

## APPENDIX

## A OTHER RELEATED WORKS

**Beyond Worst-Case Analyses of Binary Trees.** Binary trees are among the most ubiquitous pointer-based data structures. While schemes without re-balancing do obtain  $O(\log_2 n)$  time bounds in the average case, their behavior degenerates to  $\Omega(n)$  on natural access sequences such as  $1, 2, 3, \dots, n$ . To remedy this, many tree balancing schemes with  $O(\log_2 n)$  time worst-case guarantees have been proposed (Adelson-Velskii & Landis, 1963; Guibas & Sedgewick, 1978; Cormen et al., 2009).

Creating binary trees optimal for their inputs has been studied since the 1970s. Given access frequencies, the static tree of optimal cost can be computed using dynamic programs or clever greedies (Hu & Tucker, 1970; Mehlhorn, 1975b; Yao, 1982; Karpinski et al., 1996). However, the cost of such computations often exceeds the cost of invoking the tree. Therefore, a common goal is to obtain a tree whose cost is within a constant factor of the entropy of the data, multiple schemes do achieve this either on worst-case data (Mehlhorn, 1975b), or when the input follows certain distributions (Allen & Munro, 1978).

A major disadvantage of static trees is that their cost on any permutation needs to be  $\Omega(n \log_2 n)$ . On the other hand, for the access sequence  $1, 2, 3, \dots, n$ , repeatedly bringing the next accessed element to the root gives a lower cost  $O(n)$ . This prompted Allen and Munro to propose the notion of self-organizing binary search trees. This scheme was extended to splay trees by Sleator & Tarjan (1985). Splay trees have been shown to obtain many improved cost bounds based on temporal and spatial locality (Sleator & Tarjan, 1985; Cole et al., 2000; Cole, 2000; Iacono, 2005). In fact, they have been conjectured to have access costs with a constant factor of optimal on any access sequence (Iacono, 2013). Much progress has been made towards showing this over the past two decades (Demaine et al., 2009; Derryberry & Sleator, 2009; Chalermsook et al., 2019; Bose et al., 2020).

From the perspective of designing learning-augmented data structures, the dynamic optimality conjecture almost goes contrary to the idea of incorporating predictors. It can be viewed as saying that learned advice do not offer gains beyond constant factors, at least in the binary search tree setting. Nonetheless, the notion of access sequence, as well as access-sequence-dependent bounds, provides useful starting points for developing prediction-dependent search trees in online settings. In this paper, we choose to focus on bounds based on temporal locality, specifically, the working-set bound. This is for two reasons: the spatial locality of an element’s next access is significantly harder to describe compared to the time until the next access; and the current literature on spatial locality-based bounds, such as dynamic finger tends to be much more involved (Cole et al., 2000; Cole, 2000). We believe an interesting direction for extending our composite scores is to obtain analogs of the unified bound (Iacono, 2001; Bădoiu et al., 2007) for B-Trees.

**B-Trees and External Memory Model.** Parameterized B-Trees (Brodal & Fagerberg, 2003) have been studied to balance the runtime of read versus write operations, and several bounds have been shown with regard to the blocks of memory needed to be used during an operation. The optimality is discussed in both static and dynamic settings. Rosenberg and Snyder (Rosenberg & Snyder, 1981) compared the B-Tree with the minimum number of nodes (denoted as *compact*) with non-compact B-Trees and with time-optimal B-Trees. Bender et al. (Bender et al., 2016) considers keys have different sizes and gives a cache-oblivious static atomic-key B-Tree achieving the same asymptotic performance as the static B-Tree. When it comes to the dynamic setting, the trade-off between the cost of updates and accesses is widely studied (O’Neil et al., 1996; Jagadish et al., 1997; Jermaine et al., 1999; Buchsbaum et al., 2000; Yi, 2012). (Bose et al., 2008) studied the dynamic optimality of B-Trees and presented a self-adjusting B-Tree data structure that is optimal up to a constant factor when  $B$  is constant.

B-Treap were introduced by Golovin (2008; 2009) as a way to give an efficient history-independent search tree in the external memory model. These studies revolved around obtaining  $O(\log_B n)$  worst-case costs that naturally generalize Treaps. Specifically, for sufficiently small  $B$  (as compared to  $n$ ), Golovin showed a worst-case depth of  $O(\frac{1}{\alpha} \log_B n)$  with high probability, where  $B = \Omega(\ln^{1/(1-\alpha)} n)$ . The running time of this structure has recently been improved by Safavi & Seybold (2023) via a two-layer design.

The large node sizes of B-Trees interact naturally with the external memory model, where memory is accessed in blocks of size  $B$  (Brodal & Fagerberg, 2003; Vitter, 2001). The external memory model itself is widely used in data storage and retrieval Margaritis & Anastasiadis (2013), and has also been studied in conjunction with learned indices (Ferragina et al., 2020). Several previous results discuss the trade-off between update time and storage utilization (Brown, 2014; 2017; Fagerberg et al., 2019).

## B LEARNING-AUGMENTED B-TREES

We now extend the ideas above, specifically the composite priority notions, to B-Trees in the *External Memory Model*. We show that the learning-augmented B-Treaps (Appendix B.1) obtain static optimality (Appendix B.2) and is robust to the noisy predicted scores (Appendix B.3). This model is also the basis of our analyses in online settings in Appendix C.

### B.1 LEARNING-AUGMENTED B-TREAPS

We first formalize this extension by incorporating our composite priorities with the B-Treap data structure from Golovin (2009) and introducing offsets in priorities.

**Lemma B.1** (B-Treap, Golovin (2009)). *Given the unique binary Treap  $(T, \text{priority}_x)$  over the set of items  $[n]$  with their associated priorities, and a target branching factor  $B = \Omega(\ln^{1/(1-\alpha)} n)$  for some  $\alpha > 0$ . Assuming  $\text{priority}_x$  are drawn uniformly from  $(0, 1)$ , we can maintain a B-Tree  $T^B$ , called the B-Treap, uniquely defined by  $T$ . This allows operations such as Lookup, Insert, and Delete of an item to touch  $O(\frac{1}{\alpha} \log_B n)$  nodes in  $T^B$  in expectation.*

*In particular, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , all above performance guarantees hold with high probability.*

The main technical theorem is the following:

**Theorem B.2** (Learning-Augmented B-Treap via Composite Priorities). *Denote  $\mathbf{w} = (w_1, \dots, w_n) \in (0, 1)^n$  as a score associated with each element of  $[n]$  such that  $\|\mathbf{w}\|_1 = O(1)$  and a branching factor  $B = \Omega(\ln^{1/(1-\alpha)} n)$ , there is a randomized data structure that maintains a B-Tree  $T^B$  over  $U$  such that*

1. *Each item  $x$  has expected depth  $O(\frac{1}{\alpha} \log_B(1/w_x))$ .*
2. *Insertion or deletion of item  $x$  into/from  $T$  touches  $O(\frac{1}{\alpha} \log_B(1/w_x))$  nodes in  $T^B$  in expectation.*
3. *Updating the weight of item  $x$  from  $w$  to  $w'$  touches  $O(\frac{1}{\alpha} |\log_B(w'/w)|)$  nodes in  $T^B$  in expectation.*

*We consider the following priority assignment scheme: For any  $x$  and its corresponding score  $w_x$ , we always maintain:*

$$\text{priority}_x := -\lfloor \log_2 \log_B \frac{1}{w_x} \rfloor + \delta, \delta \sim U(0, 1).$$

*In addition, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , all above performance guarantees hold with high probability  $1 - \delta$ .*

The learning-augmented B-Treap is created by applying Lemma B.1 to a partition of the binary Treap  $T$ . Each item  $x$  has a priority in the binary Treap  $T$ , defined as:

$$\text{priority}_x = -\left\lfloor \log_2 \log_B \frac{1}{w_x} \right\rfloor + \delta_x, \delta_x \sim U(0, 1), \text{ for all } x \in U. \quad (5)$$

We then partition the binary Treap  $T$  based on each item's tier. The tier of an item is defined as the absolute value of the integral part of its priority, i.e.,  $\tau_x \stackrel{\text{def}}{=} \lfloor \log_2 \log_B(1/w_x) \rfloor$ .

*Proof of Theorem B.2.* To formally construct and maintain  $T^B$ , we follow these steps:

1. Start with a binary Treap  $(T, \text{priority}_x)$  with priorities defined using equation equation 5.
2. Decompose  $T$  into sub-trees based on each item's tier, resulting in a set of maximal sub-trees with items sharing the same tier.
3. For each  $T_i$ , apply Lemma B.1 to maintain a B-Treap  $T_i^B$ .
4. Combine all the B-Treaps into a single B-Tree, such that the parent of  $\text{root}(T_i^B)$  is the B-Tree node containing the parent of  $\text{root}(T_i)$ .

Now, let's analyze the depth of each item  $x$ . Keep in mind that any item  $y$  in the same B-Tree node shares the same tier. Therefore, we can define the tier of each B-Tree node as the tier of its items.

Suppose  $x_1, x_2, \dots$  are the B-Tree nodes we encounter until we reach  $x$ . The tiers of these nodes are in non-increasing order, that is,  $\tau_{x_i} \geq \tau_{x_{i+1}}$  for any  $i$ . We'll define  $C_t$  as the number of items of tier  $t$  for any  $t$ . As per the definition (refer to equation 5), we have:

$$C_t = O(B^{2^t}), \text{ for all } t$$

Using Lemma B.1 and the fact that  $B = O(C_t^{1/2})$ ,  $t \geq 1$ , we find that the number of nodes among  $x_i$  of tier  $t$  is  $O(\frac{1}{\alpha} \log_B C_t) = O(2^t/\alpha)$  with high probability. As a result, the number of nodes touched until reaching  $x$  is, with high probability:

$$\sum_{t=0}^{\tau_x} O(2^t/\alpha) = O(2^{\tau_x}/\alpha) = O\left(\frac{1}{\alpha} \log_B \frac{1}{w_x}\right)$$

This analysis is also applicable when performing Lookup, Insert, and Delete operations on item  $x$ .

The number of nodes touched when updating an item's weight can be derived from first deleting and then inserting the item.

□

## B.2 STATIC OPTIMALITY

In this section, we show that with our priority assignment, the learning-augmented B-Treaps are statically optimal. Let  $x(1), x(2), \dots, x(m)$  represent an access sequence of length  $m$ . We define the relative frequency of each item  $x$  as  $p_x \stackrel{\text{def}}{=} \frac{|\{i \in [m] \mid x(i)=x\}|}{m}$ . A static B-Tree is statically optimal if the depth of item  $x$  is:

$$\text{depth}(x) = O\left(\log_B \frac{1}{p_x}\right), \text{ for all } x \in [n]$$

As a corollary of Theorem B.2, we show that if we are given the relative frequency  $p_x$ , the learning-augmented B-Treaps with our priority assignment achieves *Static Optimality* with weights  $w_x \stackrel{\text{def}}{=} p_x$ .

**Corollary B.3** (Static Optimality for B-Treaps). *Given the relative frequency  $p_x$  of each item  $x \in [n]$ , and a branching factor  $B = \Omega(\ln^{1.1} n)$ , there exists a randomized data structure that maintains a B-Tree  $T^B$  over  $[n]$  such that each item  $x$  has an expected depth of  $O(\log_B 1/p_x)$ . That is,  $T^B$  achieves Static Optimality, meaning the total number of nodes touched is  $O(OPT_B^{\text{static}})$  in expectation, where:*

$$OPT_B^{\text{static}} \stackrel{\text{def}}{=} m \cdot \sum_{x \in [n]} p_x \log_B \frac{1}{p_x} \quad (6)$$

Furthermore, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , all above performance guarantees hold with high probability.

## B.3 ROBUSTNESS GUARANTEES

In practice, we would not have access to the relative frequency  $p_x$ . Instead, we will have an inaccurate prediction  $q_x$ . Let  $\mathbf{p}$  and  $\mathbf{q}$  be the probability distribution over  $[n]$  such that  $\mathbf{p}(x) = p_x$ ,  $\mathbf{q}(x) = q_x$ . In this section, we will show that B-Treap performance is robust to the error. Specifically, we analyze the performance under various notions of error in the prediction. The notions listed here are the ones used for learning discrete distributions (refer to Canonne (2020) for a comprehensive discussion).

**Corollary B.4** (Kullback—Leibler (KL) Divergence). *If we are given a density prediction  $\mathbf{q}$  such that  $d_{KL}(\mathbf{p}; \mathbf{q}) = \sum_x p_x \ln(p_x/q_x) \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + \frac{\epsilon m}{\ln B}\right)$$

*Proof.* Given the inaccurate prediction  $\mathbf{q}$ , the total number of touched nodes in  $T^B$  is

$$O\left(\sum_x m \cdot p_x \log_B \frac{1}{q_x}\right) = O\left(\sum_x m \cdot p_x \log_B \frac{1}{p_x} + m \cdot \sum_x p_x \log_B \frac{p_x}{q_x}\right) = O(OPT_B^{\text{static}} + m \frac{d_{KL}(\mathbf{p}; \mathbf{q})}{\ln B})$$

□

**Corollary B.5** ( $\chi^2$ ). *If we are given a density prediction  $\mathbf{q}$  such that  $\chi^2(\mathbf{p}; \mathbf{q}) = \sum_x (p_x - q_x)^2 / q_x \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + \frac{\epsilon m}{\ln B}\right)$$

*Proof.* The corollary follows from [Corollary B.5](#) and the fact  $d_{KL}(\mathbf{p}; \mathbf{q}) \leq \chi^2(\mathbf{p}; \mathbf{q}) \leq \epsilon$ .  $\square$

**Corollary B.6** ( $L_\infty$  Distance). *If we are given a density prediction  $\mathbf{q}$  such that  $\|\mathbf{p} - \mathbf{q}\|_\infty \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + m \log_B(1 + \epsilon n)\right)$$

*Proof.* For item  $x$  with its marginal probability smaller than  $1/1000n$ , its expected depth in the B-Treap is  $O(\log_B n)$  using either  $p_x$  or  $q_x$  as its score. If item  $x$ 's marginal probability is at least  $1/1000n$ , the  $L_\infty$  distance implies that

$$\frac{p_x}{\tilde{p}_x} = 1 + \frac{p_x - \tilde{p}_x}{\tilde{p}_x} \leq 1 + \frac{\epsilon}{1/(1000n)} = 1 + 1000(1 + \epsilon n)$$

Therefore, item  $x$ 's expected depth in the B-Treap with score  $\mathbf{q}$  is roughly

$$O\left(\log_B \frac{1}{p_x} + \log_B \frac{p_x}{q_x}\right) \leq O\left(\log_B \frac{1}{p_x} + \log_B(1 + \epsilon n)\right)$$

The corollary follows.  $\square$

**Corollary B.7** ( $L_2$  Distance). *If we are given a density prediction  $\mathbf{q}$  such that  $\|\mathbf{p} - \mathbf{q}\| \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + m \log_B(1 + \epsilon n)\right)$$

*Proof.* This claim follows from [Corollary B.6](#) and the fact  $\|\mathbf{p} - \mathbf{q}\|_\infty \leq \|\mathbf{p} - \mathbf{q}\|_2 \leq \epsilon$ .  $\square$

**Corollary B.8** (Total Variation). *If we are given a density prediction  $\mathbf{q}$  such that  $d_{TV}(\mathbf{p}, \mathbf{q}) = 0.5\|\mathbf{p} - \mathbf{q}\|_1 \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + m \log_B(1 + \epsilon n)\right)$$

*Proof.* This claim follows from [Corollary B.6](#) and the fact  $\|\mathbf{p} - \mathbf{q}\|_\infty \leq \|\mathbf{p} - \mathbf{q}\|_1 \leq 2\epsilon$ .  $\square$

**Corollary B.9** (Hellinger Distance). *If we are given a density prediction  $\mathbf{q}$  such that  $d_H(\mathbf{p}, \mathbf{q}) = 0.5\|\sqrt{\mathbf{p}} - \sqrt{\mathbf{q}}\|_2 \leq \epsilon$ , the total number of touched nodes is*

$$O\left(OPT_B^{\text{static}} + m \log_B(1 + \epsilon n)\right)$$

*Proof.* This claim follows from [Corollary B.6](#) and the fact  $\|\mathbf{p} - \mathbf{q}\|_\infty \leq 2\sqrt{2}d_H(\mathbf{p}, \mathbf{q}) \leq 2\sqrt{2}\epsilon$ .  $\square$

## C DYNAMIC LEARNING-AUGMENTED SEARCH TREES

In this section, we investigate the properties of dynamic B-trees that permit modifications concurrent with sequence access. Prioritizing items that are anticipated to be accessed in the near future to reside at lower depths within the tree can significantly reduce access times. Nonetheless, updating the B-trees introduces additional costs. The overarching goal is to minimize the composite cost, which includes both the access operations across the entire sequence and the modifications to the B-trees. We specifically concentrate on the study of locally dynamic B-trees, which are characterized by the restriction that tree modifications are limited solely to the adjustment of priorities for the items being accessed.

In [Appendix C.1](#), we give the total cost guarantees for any locally dynamic B-trees. In [Appendix C.2](#), we establish the robustness guarantees in the context of imprecise priority scores, which may be given from a learning oracle. In [Appendix C.3](#), we demonstrate that the dynamic learning-augmented B-trees with a specific priority based on the working set size — the number of distinct items requested between two consecutive accesses — achieves the working set property. Full details are included in the appendix. Finally, in [Appendix C.4](#), we analyze the general dynamic B-trees with general time-varying priorities.

### C.1 LOCALLY DYNAMIC B-TREES

Our objective is to maintain a data structure that minimizes the total cost of accessing the sequence  $S$  given the time-varying score  $w(i) \in (0, 1)^n, i \in [m]$  associated to each item. Here, we focus on the dynamic B-trees that update the priorities of only the items being accessed at any given moment, leaving the priorities of all other items unchanged. We refer to these as *locally dynamic B-trees*.

Given  $n$  items, denoted as  $[n] = \{1, \dots, n\}$ , and a sequence of access sequence  $\mathbf{X} = (x(1), \dots, x(m))$ , where  $x(i) \in [n]$ . At time  $i \in [m]$ , there exists some time-dependent score  $w_{ij}$  associated with each item  $j \in [n]$ . Let  $\mathbf{w}(i) = (w_{i,1}, \dots, w_{i,n}) \in (0, 1)^n$  be the time-varying score vector. The score  $\mathbf{w}(i)$  is defined to be *locally changed* if it only differs from the previous score vector at the index of the item being accessed. In other words, at each time  $i \in \{1, 2, \dots, n\}$ , we have  $w_{i,j} = w_{i-1,j}$  for any  $j \neq x(i)$ . The *locally dynamic B-Treap* is then defined as a B-Treap whose priorities are updated according to the locally changed score. For any vector  $\mathbf{w}$ , we write  $\log \mathbf{w}$  as the vector taking the element-wise log on  $\mathbf{w}$ . We give the guarantees in [Theorem C.1](#).

**Theorem C.1** (Locally-Dynamic B-Treap with Given Priorities). *Given the locally changed scores  $\mathbf{w}(i) \in (0, 1)^n, i \in [m]$  satisfying  $\|\mathbf{w}(i)\|_1 = O(1)$  and a branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree  $T^B$  over  $[n]$  such that when accessing the item  $x(i)$  at time  $i$ , the expected depth of item  $x(i)$  is  $O(\log_B \frac{1}{w_{i,x(i)}})$ . The expected total cost for processing the whole access sequence  $\mathbf{X}$  is*

$$\text{cost}(\mathbf{X}, \mathbf{w}) = O\left(n \log_B n + \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}}\right).$$

Moreover, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .

The proof is an application of [Theorem B.2](#), where the priority function dynamically changes as time goes on, rather than the *Static Optimality* case where the priority is fixed beforehand.

*Proof of Theorem C.1.* Initially, we set the priority for all items to be 1, and insert all items into the Treap. For any time  $i \in [n]$ , for  $j \in [n]$  such that  $w_{i-1,j} \neq w_{i,j}$ , we set

$$\text{priority}_j^{(i)} := -\lfloor \log_4 \log_B \frac{1}{w_{i,j}} \rfloor + \delta_{ij}, \delta_{ij} \sim U(0, 1).$$

Since  $\|\mathbf{w}(i)\|_1 = O(1), i \in [m]$ , by [Theorem B.2](#), the expected depth of item  $s(i)$  is  $O(\log_B \frac{1}{w_{i,x(i)}})$ . The total cost for processing the sequence consists of both accessing  $x(i)$  and updating the priorities. The expected total cost for all the accesses is

$$O\left(\sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}}\right).$$

Then we will calculate the cost to update the Treap. Since the priority of an item only changes when it is accessed. Updating the priority of  $x(i)$  from  $w_{i-1,x(i)}$  to  $w_{i,x(i)}$  has cost

$$O\left(\left|\log_B \frac{w_{i-1,x(i)}}{w_{i,x(i)}}\right|\right).$$

Hence we can bound the expected total cost for maintaining the Treap by

$$O\left(n \log_B n + \sum_{i=2}^m \left|\log_B \frac{w_{i-1,x(i)}}{w_{i,x(i)}}\right|\right) = O\left(n \log_B n + 2 \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}}\right).$$

Together the expected total cost is

$$O\left(n \log_B n + \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}}\right).$$

The high probability bound follows similarly as [Theorem B.2](#).  $\square$

## C.2 ROBUSTNESS GUARANTEES

We have shown that given time-varying scores associated with each item  $w(i)$ , there exists a B-Tree that gives us the total cost in terms of the scores. In this section, we address scenarios in which precise score  $w(i)$  are not accessible. Utilizing a learning oracle that predicts the logarithm of the score, we demonstrate that the total cost incorporates an additive term corresponding to the mean absolute error (MAE) of the logarithm of the scores  $\sum_{i=1}^m |\log_B w_{i,x(i)} - \log_B \tilde{w}_{i,x(i)}|$ . We predict the logarithm of the score instead of itself to better capture the scale of it.

**Theorem C.2** (Locally Dynamic B-Treap with Predicted Scores). *Given the predicted locally changed scores  $\tilde{w}(i) \in (0, 1)^n$  satisfying  $\|\tilde{w}(i)\|_1 = O(1)$ ,  $\tilde{w}_{i,j} \geq 1/\text{poly}(n)$  and a branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree over the  $n$  keys such that the expected total cost for processing the whole access sequence  $\mathbf{X}$  is*

$$\text{cost}(\mathbf{X}, \tilde{\mathbf{w}}) = \text{cost}(\mathbf{X}, \mathbf{w}) + O\left(\sum_{i=1}^m |\log_B w_{i,x(i)} - \log_B \tilde{w}_{i,x(i)}|\right).$$

Moreover, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .

*Proof.* We apply Theorem C.1 with the predicted score  $\tilde{w}(i)$ , and get the expected total loss is

$$\begin{aligned} \text{cost}(\mathbf{X}, \tilde{\mathbf{w}}) &= O\left(n \log_B n + \sum_{i=1}^m \log_B \frac{1}{\tilde{w}_{i,x(i)}}\right) \\ &\leq \text{cost}(\mathbf{X}, \mathbf{w}) + O\left(\sum_{i=1}^m \left|\log_B \frac{1}{\tilde{w}_{i,x(i)}} - \log_B \frac{1}{w_{i,x(i)}}\right|\right) \\ &= \text{cost}(\mathbf{X}, \mathbf{w}) + O\left(\sum_{i=1}^m |\log_B w_{i,x(i)} - \log_B \tilde{w}_{i,x(i)}|\right). \end{aligned}$$

□

## C.3 WORKING SET PROPERTY

In data structures, the working set is the collection of data that a program uses frequently over a given period. This concept is important because it helps us understand how a program interacts with memory and thus enables us to design more efficient data structures and algorithms. For example, if a program is sorting a list, the working set might be the elements of the list it is comparing and swapping right now. The size of the working set can affect how fast the program runs. A smaller working set can make the program run faster because it means the program doesn't need to reach out to slower parts of memory as often. In other word, if we know which parts of a data structure are used most, we can organize the data or even the memory in a way that makes accessing these parts faster, which can speed up the entire program.

In this section, we construct dynamic learning-augmented B-treaps that achieve the working set property. We define the *working-set size* as the number of distinct items accessed between two consecutive accesses. Correspondingly, we design a time-varying score, *working-set score*, as the reciprocal of the square of one plus working-set size. We will show that the working-set score is locally changed and there exists a data structure that achieves the *working-set property*, which states that the time to access an element is a logarithm of its working-set size. The formal definition of the working-set size and the main theorems in this section are presented as follows.

**Definition C.3** (Previous and Next Access  $\text{prev}(i, x)$  and  $\text{next}(i, x)$ ). *Let  $\text{prev}(i, x)$  be the previous access of item  $x$  at or before time  $i$ , i.e.,  $\text{prev}(i, x) := \max\{i' \leq i \mid x(i') = x\}$ . Let  $\text{next}(i, x)$  to be the next access of item  $x$  after time  $i$ , i.e.,  $\text{next}(i, x) := \min\{i' > i \mid x(i') = x\}$ .*

**Definition C.4** (Working-set Size  $\text{work}(i, x)$ ). *Define the working-set Size  $\text{work}(i, x)$  to be the number of distinct items accessed between the previous access of item  $x$  at or before time  $i$  and the next access of item  $x$  after time  $i$ . That is,*

$$\text{work}(i, x) \stackrel{\text{def}}{=} |\{x(\text{prev}(i, x) + 1), \dots, x(\text{next}(i, x))\}|.$$

If  $x$  does not appear after time  $i$ , we define  $\text{work}(i, x) := n$ .



**Theorem C.5** (Dynamic B-Treaps with Working-set Priority). *With the working-set size  $\text{work}(i, x)$  known and the branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree  $T^B$  over  $[n]$  with the priorities assigned as*

$$\text{priority}(i, x) = -\lfloor \log_2 \log_B(1 + \text{work}(i, x))^2 \rfloor + U(0, 1).$$

*Upon accessing the item  $x$  at time  $i$ , the expected depth of item  $x$  is  $O(\log_B(1 + \text{work}(i, x)))$ . The expected total cost for processing the whole access sequence  $\mathbf{X}$  is*

$$\text{cost}(\mathbf{X}, \text{priority}) = O\left(n \log_B n + \sum_{i=1}^m \log_B(1 + \text{work}(i, x))\right).$$

*In particular, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .*

**Remark.** Consider two sequences with length  $m$ ,  $\mathbf{X}_1 = (1, 2, \dots, n, 1, 2, \dots, n, \dots, 1, 2, \dots, n)$ ,  $\mathbf{X}_2 = (1, 1, \dots, 1, 2, 2, \dots, 2, \dots, n, n, \dots, n)$ . Two sequences have the same total cost if we have a fixed score. However,  $\mathbf{X}_2$  should have less cost because of its repeated pattern. Given the frequency  $\text{freq}$  as a time-invariant priority, by [Corollary B.3](#), the optimal static costs are

$$\text{cost}(\mathbf{X}_1, \text{freq}) = \text{cost}(\mathbf{X}_2, \text{freq}) = O(m \log_B n).$$

But for the dynamic B-Trees, with the working-set score, we calculate both costs from [Theorem C.5](#) as

$$\text{cost}(\mathbf{X}_1, \omega) = O(m \log_B(n + 1)), \quad \text{cost}(\mathbf{X}_2, \omega) = O(n \log_B n + m \log_B 3).$$

This means that our proposed priority can better capture the timing pattern of the sequence and thus can even do better than the optimal static setting.

The main idea to prove [Theorem C.5](#) is to show that (1) the working-set size is locally changed and (2) the corresponding priority satisfies the regularity conditions in [Theorem C.1](#). To complete the proof, we introduce the interval-set size  $\text{interval}(i, x)$ . See [Figure 11](#) as an illustration.

**Definition C.6** (Interval-set Size  $\text{interval}(i, x)$ ). *Define the Interval-set Size  $\text{interval}(i, x)$  to be the number of distinct items accessed between time  $i$  and the next access of item  $x$  after time  $i$ . That is,*

$$\text{interval}(i, x) := |\{x(i+1), \dots, x(\text{next}(i, x))\}|.$$

*If  $x$  does not appear after time  $i$ , we define  $\text{interval}(i, x) := n$ .*

Furthermore, we define the working-set score as follows.

**Definition C.7** (Working-set Score  $\omega(i, x)$ ). *Define the time-varying priority as the reciprocal of the square of one plus working-set size. That is,*

$$\omega(i, x) = \frac{1}{(1 + \text{work}(i, x))^2}$$

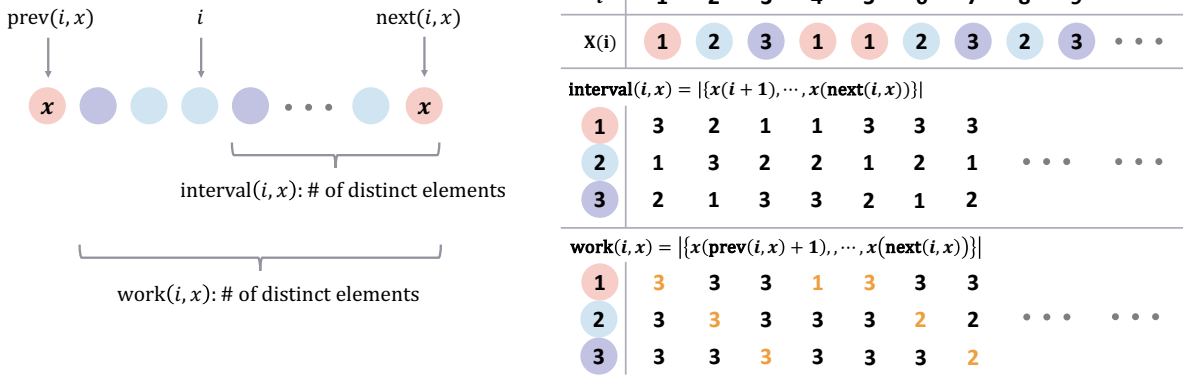
Next, we will show that the interval set priority is  $O(1)$  for any time  $i \in [m]$  in [Lemma C.8](#). The proof has three steps. Firstly, the interval-set size at time  $i$  is always a permutation of  $[n]$ . Secondly, for any  $i \in [m]$ ,  $x \in [n]$ , the working-set size is always no less than the interval-set size. Therefore, for any  $i \in [m]$ , the  $l_1$  norm of working-set score vector  $\omega(i) \stackrel{\text{def}}{=} (\omega(i, 1), \dots, \omega(i, n))$  can be upper bounded by  $\sum_{j=1}^n 1/(1+j)^2 = O(1)$ .

**Lemma C.8** (Norm Bound for Working-set Score). *Fix any timestamp  $i \in [m]$ ,*

$$\sum_{j=1}^n \omega(i, j) = O(1).$$

*Proof.* We first show that at any time  $i \in [m]$ , the interval-set size is a permutation of  $[n]$ . By definition,  $\text{interval}(i, x)$  is the number of items  $y$  such that  $\text{next}(i, y) \leq \text{next}(i, x)$ . Let  $\pi_1, \dots, \pi_n$  be a permutation of all items  $[n]$  in the order of increasing  $\text{next}(i, x)$ . Then for any item  $x$ ,  $\text{interval}(i, x)$  is the index of  $x$  in  $\pi$ . So the sum of the reciprocal of squared  $\text{interval}(i, x)$  is upper bounded by

$$\sum_{j=1}^n \frac{1}{(1 + \text{interval}(i, j))^2} = \sum_{j=1}^n \frac{1}{(1 + j)^2} = \Theta(1).$$



**Figure 11:** An example of  $n = 3$ ,  $X = (1, 2, 3, 1, 1, 2, 3, \dots)$ . For any  $i, x$ ,  $\text{work}(i)$  is a permutation of  $[n]$ ;  $\text{interval}(i, x) \geq \text{work}(i, x)$ ;  $\text{interval}(i, x)$  changes only when  $x(i) = x$  (highlighted in orange).

Secondly, recall that  $\text{prev}(i, x) \leq i$ , and hence for any  $i \in [m]$ ,  $x \in [n]$ ,  $\text{work}(i, x) \geq \text{interval}(i, x)$ . So we have the upper bound for working-set score as follows.

$$\sum_{j=1}^n \omega(i, j) = \sum_{j=1}^n \frac{1}{(1 + \text{work}(i, x))^2} \leq \sum_{j=1}^n \frac{1}{(1 + \text{interval}(i, j))^2} = O(1).$$

□

Since we have shown the working-set score has constant  $l_1$  norm and in each timestamp, we only update one item's priority. We are ready to prove and show the efficiency of the corresponding B-Treap.

*Proof of Theorem C.5.* By Lemma C.8, we know  $\|\omega(i)\| = O(1)$ , for all  $i \in [m]$ . Also by definition of  $\text{work}(i, x)$ , for any  $x \in [n]$ ,  $\text{work}(i-1, x) \neq \text{work}(i, x)$  only when  $x(i) = x$ . So for each time  $i$ , at most one item (i.e.,  $x(i)$ ) changes its priority. So we apply Theorem C.1 with  $w_{i,j} = \omega(i, x)$ ,  $i \in [m]$ ,  $x \in [n]$ , and get the total cost

$$\text{cost}(X, \omega) = O\left(n \log_B n + \sum_{i=1}^m \log_B(1 + \text{work}(i, x(i)))\right)$$

□

Furthermore, we use the following theorem to show the robustness of the results when the scores are inaccurate. This is a direct corollary of Theorem C.2.

**Theorem C.9** (Locally-Dynamic B-Treaps with Predictions). *Given the predicted locally changed working-set score  $\tilde{\omega}(i) \in (0, 1)^n$  satisfying  $\|\tilde{\omega}(i)\|_1 = O(1)$ ,  $\tilde{\omega}_{i,j} \geq 1/\text{poly}(n)$  and the branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree over the  $n$  keys such that the expected total cost for processing the whole access sequence  $X$  is*

$$\text{cost}(X, \tilde{\omega}) = \text{cost}(X, \omega) + O\left(\sum_{i=1}^m |\log_B \omega_{i, x(i)} - \log_B \tilde{\omega}_{i, x(i)}|\right).$$

In particular, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .

#### C.4 GENERAL RESULTS FOR DYNAMIC B-TREES

In this section, we give the results for general dynamic B-trees. We first construct the dynamic B-Treaps and give the guarantees when we have access to the real-time priorities for each item in Appendix C.4.1. Then we analyze the dynamic B-trees given the estimation the time-varying priorities in Appendix C.4.2. We use the same notation in Appendix C.



#### C.4.1 DYNAMIC B-TREAP WITH GIVEN PRIORITIES

**Theorem C.10** (Dynamic B-Treap with Given Priorities). *Given the time-varying scores  $w(i) \in (0, 1)^n$ ,  $i \in [m]$  satisfying  $\|w(i)\|_1 = O(1)$  and a branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree  $T^B$  over  $[n]$  such that when accessing the item  $x(i)$  at time  $i$ , the expected depth of item  $x(i)$  is  $O(\log_B \frac{1}{w_{i,x(i)}})$ . The expected total cost for processing the whole access sequence  $\mathbf{X}$  is*

$$\text{cost}(\mathbf{X}, w) = O \left( n \log_B n + \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}} + \sum_{i=2}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{w_{i-1,j}} \right| \right).$$

In particular, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .

*Proof.* Initially, we set the priority for all items to be 1, and insert all items into the Treap. For any time  $i \in [m]$ , for  $j \in [n]$  such that  $w_{i-1,j} \neq w_{i,j}$ , we set

$$\text{priority}_j^{(i)} := -\lfloor \log_4 \log_B \frac{1}{w_{i,j}} \rfloor + \delta_{ij}, \delta_{ij} \sim U(0, 1).$$

Since  $\|w(i)\|_1 = O(1)$ ,  $i \in [m]$ , by [Theorem B.2](#), the expected depth of item  $s(i)$  is  $O(\log_B \frac{1}{w_{i,x(i)}})$ . The total cost for processing the sequence consists of both accessing  $x(i)$  and updating the priorities. The expected total cost for all the accesses is

$$O \left( \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}} \right).$$

Then we will calculate the cost to update the Treap. Updating the priority of  $j$  from  $w_{i-1,j}$  to  $w_{i,j}$  has cost  $O(|\log_B(w_{i-1,j}/w_{i,j})|)$ . Hence we can bound the expected total cost for maintaining the Treap by

$$O \left( n \log_B n + \sum_{i=2}^m \sum_{j=1}^n \left| \log_B \frac{w_{i-1,j}}{w_{i,j}} \right| \right).$$

Together the expected total cost is

$$O \left( n \log_B n + \sum_{i=1}^m \log_B \frac{1}{w_{i,x(i)}} + \sum_{i=2}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{w_{i-1,j}} \right| \right).$$

The high probability bound follows similarly as [Theorem B.2](#).  $\square$

**Remark.** The total cost for processing the access sequence has three terms. The first two terms are the same as in the static optimality bound, while the third term is incurred from updating the scores. Hence, here is a trade-off between the costs of updating items and the benefits from the time-varying scores. Moreover, the locally-dynamic B-trees can avoid the high cost of keeping updating the scores because only one score is changed per time.

#### C.4.2 DYNAMIC B-TREAP WITH PREDICTED PRIORITIES

In this section, we give the guarantees for the dynamic B-Treaps with predicted priorities learned by a machine learning oracle. Similar as in [Appendix C.4.2](#), we here predict  $\log_B \frac{1}{w_{i,j}}$  to better capture the scale of the scores. And we will find that the total cost using the B-Trees using the predicted scores is equal to the cost using the accurate priorities plus an additive error that is linear in the mean absolute error of our prediction scores:

$$\sum_{i=1}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i,j}} \right|.$$

**Theorem C.11** (Dynamic B-Treap with Predicted Scores). *Given the predicted time-varying scores  $\tilde{w}(i) \in (0, 1)^n$  satisfying  $\|\tilde{w}(i)\|_1 = O(1)$ ,  $\tilde{w}_{i,j} \geq 1/\text{poly}(n)$  and a branching factor  $B = \Omega(\ln^{1.1} n)$ , there is a randomized data structure that maintains a B-Tree over the  $n$  keys such that the expected total cost for processing the whole access sequence  $\mathbf{X}$  is*

$$\text{cost}(\mathbf{X}, \tilde{w}) = \text{cost}(\mathbf{X}, w) + O \left( \sum_{i=1}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i,j}} \right| \right)$$

In particular, if  $B = O(n^{1/2-\delta})$  for some  $\delta > 0$ , the guarantees hold with probability  $1 - \delta$ .

*Proof.* We apply [Theorem C.10](#) with score  $\tilde{w}$ , and get the expected depth of  $x(i)$  is

$$O\left(\log_B \frac{1}{\tilde{w}_{i,j}}\right).$$

The expected total cost is

$$\begin{aligned} \text{cost}(\mathbf{X}, \tilde{w}) &= O\left(n \log_B n + \sum_{i=1}^m \log_B \frac{1}{\tilde{w}_{i,x(i)}} + \sum_{i=2}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i-1,j}} \right| \right) \\ &= \text{cost}(\mathbf{X}, w) + O\left(\sum_{i=1}^m \left| \log_B \frac{1}{w_{i,x(i)}} - \log_B \frac{1}{\tilde{w}_{i,x(i)}} \right| \right) \\ &\quad + O\left(\sum_{i=2}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i,j}} \right| + \sum_{i=1}^{m-1} \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i,j}} \right| \right) \\ &= \text{cost}(\mathbf{X}, w) + O\left(\sum_{i=1}^m \sum_{j=1}^n \left| \log_B \frac{1}{w_{i,j}} - \log_B \frac{1}{\tilde{w}_{i,j}} \right| \right) \end{aligned}$$

□