An Adaptive Packed-Memory Array

Michael A. Bender
Stony Brook University
and
Haodong Hu
Stony Brook University

The ***packed-memory array*** (***PMA***) is a data structure that maintains a dynamic set of N elements in sorted order in a $\Theta(N)$-sized array. The idea is to intersperse $\Theta(N)$ empty spaces or ***gaps*** among the elements so that only a small number of elements need to be shifted around on an insert or delete. Because the elements are stored physically in sorted order in memory or on disk, the PMA can be used to support extremely efficient range queries. Specifically, the cost to scan L consecutive elements is $O(1+L/B)$ memory transfers.

This paper gives the first ***adaptive packed-memory array*** (***APMA***), which automatically adjusts to the input pattern. Like the traditional PMA, any pattern of updates costs only $O(\log^2 N)$ amortized element moves and $O(1+(\log^2 N)/B)$ amortized memory transfers per update. However, the APMA performs even better on many common input distributions achieving only $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ amortized memory transfers. The paper analyzes ***sequential*** inserts, where the insertions are to the front of the APMA, ***hammer*** inserts, where the insertions "hammer" on one part of the APMA, ***random*** inserts, where the insertions are after random elements in the APMA, and ***bulk*** inserts, where for constant $\alpha \in [0,1]$, N^α elements are inserted after random elements in the APMA. The paper then gives simulation results that are consistent with the asymptotic bounds. For sequential insertions of roughly 1.4 million elements, the APMA has four times fewer element moves per insertion than the traditional PMA and running times that are more than seven times faster.

Categories and Subject Descriptors: D.1.0 [**Programming Techniques**]: General; E.1 [**Data Structures**]: Arrays; E.1 [**Data Structures**]: Lists, stacks, queues; E.5 [**Files**]: Sorting/searching; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms: Algorithms, Experimentation, Performance, Theory.

Additional Key Words and Phrases: Adaptive Packed-Memory Array, Cache Oblivious, Locality Preserving, Packed-Memory Array, Range Query, Rebalance, Sequential File Maintenance, Sequential Scan, Sparse Array.

1. INTRODUCTION

A classical problem in data structures and databases is how to maintain a dynamic set of N elements *in sorted order* in a $\Theta(N)$-sized array. The idea is to intersperse $\Theta(N)$ empty spaces or *gaps* among the elements so that only a small number of elements need to be shifted around on an insert or delete. These data structures effectively simulate a library bookshelf, where gaps on the shelves mean that books are easily added and removed.

Remarkably, such data structures can be efficient for any pattern of inserts/deletes. In-

Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA. Email: {bender,huhd}@cs.sunysb.edu. This research was supported in part by NSF Grants CCF 0621439/0621425, CCF 0540897/05414009, CCF 0634793/0632838, and CNS 0627645.

deed, it has been known for over two decades that the number of element moves per update is only $O(\log^2 N)$ both amortized [Itai et al. 1981] and in the worst case [Willard 1982; 1986; 1992]. Since these data structures were proposed, this problem has been studied under different names, including ***sparse arrays*** [Itai et al. 1981; Katriel 2002], ***sequential file maintenance*** [Willard 1982; 1986; 1992], and ***list labeling*** [Dietz 1982; Dietz and Sleator 1987; Dietz and Zhang 1990; Dietz et al. 1994]. The problem is also closely related to the ***order-maintenance*** problem [Dietz 1982; Tsakalidis 1984; Dietz and Sleator 1987; Bender et al. 2002].

Recently there has been renewed interest in these sparse-array data structures because of their application in I/O-efficient and cache-oblivious algorithms. The I/O-efficient and cache oblivious version of the sparse array is called the ***packed memory array (PMA)*** [Bender et al. 2000; 2005]. The PMA maintains N elements in sorted order in a $\Theta(N)$-sized array. It supports the operations insert, delete, and scan. Let B be the number of elements that fit within a memory block. To insert an element y after a given element x, when given a pointer to x, or to delete x, costs $O(\log^2 N)$ amortized element moves and $O(1+(\log^2 N)/B)$ amortized memory transfers. The PMA maintains the ***density invariant*** that in any region of size S (for S greater than some small constant value), there are $\Theta(S)$ elements stored in it. To scan L elements after a given element x, when given a pointer to x, costs $\Theta(1+L/B)$ memory transfers.

The PMA has been used in cache-oblivious B-trees [Bender et al. 2000; Bender et al. 2002; Brodal et al. 2002; Bender et al. 2004; Bender et al. 2005; Bender et al. 2006], concurrent cache-oblivious B-trees [Bender et al. 2005], cache-oblivious string B-tree [Bender et al. 2006], and scanning structures [Bender et al. 2002]. A sparse array in the same spirit as the PMA was independently proposed and used in the locality-preserving B-tree of [Raman 1999], although the asymptotic space bounds are superlinear and therefore inferior to the linear space bounds of the earlier sparse-array data structures [Itai et al. 1981; Willard 1982; 1986; 1992] and the PMA [Bender et al. 2000; 2005].

We now give more details about how to implement search in a PMA. For example, the update and scan bounds above assume that we are given a pointer to an element x; we now show how to find such a pointer. A straightforward approach is to use a standard binary search, slightly modified to deal with gaps. However, binary search does not have good data locality. As a result, binary search is not efficient when the PMA resides on disk because search requires $O(1+\log\lceil N/B\rceil)$ memory transfers. An alternative approach is to use a separate index into the array; the index is designed for efficient searches. In [Raman 1999] that index is a B-tree, and in [Bender et al. 2000; Bender et al. 2002; 2004; Bender et al. 2005] the index is some type of binary search tree, laid out in memory using a so-called van Emde Boas layout [Prokop 1999; Bender et al. 2000; 2005].

The primary use of the PMA in the literature has been for sequential storage in memory/disk of all the elements of a (cache-oblivious or traditional) B-tree. An early paper suggesting this idea was [Raman 1999]. The PMA maintains locality of reference at all granularities and consequently supports extremely efficient sequential scans/range queries of the elements. The concern with traditional B-trees is that the 2K or 4K sizes of disk blocks are too small to amortize the cost of disk seeks. Consequently, on modern disks, random block accesses are well over an order-of-magnitude slower than sequential block accesses. Thus, locality-preserving B-trees and cache-oblivious B-trees based on PMAs support range queries that run an order of magnitude faster than those of traditional B-trees [Bender et al. 2006]. Moreover, since the elements are maintained strictly in sorted order, these structures do not suffer from aging unlike most file systems and databases. The point is that traditional B-trees age: As new blocks are allocated and deallocated to the B-tree, blocks that are logically near each other, are far from each other on the disk. The result is that range-query performance suffers.

The PMA is an efficient and promising data structure, but it also has weaknesses. The main weakness is that the PMA performs relatively poorly on some common insertion patterns such as sequential inserts. For sequential inserts, the PMA performs near its worst in terms of the number of elements moved per insert. The PMA's difficulty with sequential inserts is that the insertions "hammer" on one part of the array, causing many elements to be shifted around. Although $O(\log^2 N)$ amortized elements moves and $O(1 + (\log^2 N)/B)$ amortized memory transfers is surprisingly good considering the stringent requirements on the data order, it is relatively slow compared with traditional B-tree inserts. Moreover, sequential inserts are common, and B-trees in databases are frequently optimized for this insertion pattern. It would be better if the PMA could perform near its best, not worst, in this case.

In contrast, one of the PMA's strengths is its performance on common insertion patterns such as random inserts. For random inserts, the PMA performs extremely well with only $O(\log N)$ element moves per insert and only $O(1 + (\log N)/B)$ memory transfers. This performance surpasses the guarantees for arbitrary inserts.

Results. This paper proposes an ***adaptive packed-memory array*** (abbreviated ***adaptive PMA*** or ***APMA***), which overcomes these deficiencies of the ***traditional PMA***. Our structure is the first PMA that adapts to insertion patterns and it gives the largest decrease in the cost of sparse arrays/sequential-file maintenance in almost two decades. The APMA retains the same amortized guarantees as the traditional PMA, but adapts to common insertion patterns, such as sequential inserts, random inserts, and bulk inserts, where chunks of elements are inserted at random locations in the array.

We give the following results for the APMA:

- We first show that the APMA has the "rebalance property", which ensures that any pattern of insertions cost only $O(1 + (\log^2 N)/B)$ amortized memory transfers and $O(\log^2 N)$ amortized element moves. Because the elements are kept in sorted order in the APMA, as with the PMA, scans of L elements costs $O(1 + L/B)$ memory transfers. Thus, the adaptive PMA guarantees performance at least as good as that of the traditional PMA.

We next analyze the performance of the APMA under some common insertion patterns.

- We show that for ***sequential inserts***, where all the inserts are to the front of the array, the APMA makes only $O(\log N)$ amortized element moves and $O(1 + (\log N)/B)$ amortized memory transfers.
- We generalize this analysis to ***hammer inserts***, where the inserts hammer on any single element in the array.
- We then turn to ***random inserts***, where each insert occurs after a randomly chosen element in the array. We establish that the insertion cost is again only $O(\log N)$ amortized element moves and $O(1 + (\log N)/B)$ amortized memory transfers.
- We generalize all these previous results by analyzing the case of ***bulk inserts***. In the bulk-insert insertion pattern, we pick a random element in the array and perform N^α inserts after it for $\alpha \in [0, 1]$. We show that for all values of $\alpha \in [0, 1]$, the APMA also

only performs $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ amortized memory transfers.

- We next perform simulations and experiments, measuring the performance of the APMA on these insertion patterns. For sequential insertions of roughly 1.4 million elements, the APMA has over four times fewer element moves per insertion than the traditional PMA and running times that are nearly seven times faster. For bulk insertions of 1.4 million elements, where $f(N) = N^{0.6}$, the APMA has over two times fewer element moves per insertion than the traditional PMA and running times that are over three times faster.

2. ADAPTIVE PACKED-MEMORY ARRAY

In this section we introduce the adaptive PMA. We first explain how the adaptive PMA differs from the traditional PMA. We then show that both PMAs have the same amortized bounds, $O(\log^2 N)$ element moves and $O(1+(\log^2 N)/B)$ memory transfers per insert/delete. Thus, adaptivity comes at no extra asymptotic cost.

Description of Traditional and Adaptive PMAs. We first describe how to insert into both the adaptive and traditional PMAs. Henceforth, ***PMA*** with no preceding adjective refers to either structure. When we insert an element y after an existing element x in the PMA, we look for a neighborhood around element x that has sufficiently low ***density***, that is, we look for a subarray that is not storing too many or too few elements. Once we find a neighborhood of the appropriate density, we ***rebalance*** the neighborhood by spacing out the elements, including y. In the traditional PMA, we rebalance by spacing out the elements evenly. In the adaptive PMA, we may rebalance the elements *unevenly*, based on previous insertions, that is, we leave extra gaps near elements that have recently had inserts after them.

We deal with a PMA that is too full or empty, as with a traditional hash table. Namely, we recopy the elements into a new PMA that is a constant factor larger or smaller. In this paper, this constant is stated as 2. However, the constant could be larger or smaller (say 1.2) with almost no change in running time. This is because most of the cost from element moves come from rebalances rather than from recopies.

We now give some terminology. We divide the PMA into $\Theta(N/\log N)$ ***segments***, each of size $\Theta(\log N)$, and we let the number of segments be a power of 2. We call a contiguous group of segments a ***window***. We view the PMA in terms of a tree structure, where the nodes of the tree are windows. The root node is the window containing all segments, and a leaf node is a window containing a single segment. A node in the tree that is a window of 2^i segments has two children, a left child that is the window of the first 2^{i-1} segments and a right child that is the window of the last 2^{i-1} segments.

We let the height of the tree be h, so that $2^h = \Theta(N/\log N)$ and $h = \lg N - \lg\lg N + O(1)$. The nodes at each height ℓ have an ***upper density threshold*** τ_ℓ and a ***lower density threshold*** ρ_ℓ, which together determine the acceptable density of keys within a window of 2^ℓ segments. As the node height *increases*, the upper density thresholds *decrease* and the lower density thresholds *increase*. Thus, for constant minimum and maximum densities $D_{\min}$ and $D_{\max}$, we have

$$D_{\min} = \rho_0 < \cdots < \rho_h < \tau_h < \cdots < \tau_0 = D_{\max}. \quad (1)$$

The density thresholds on windows of intermediate powers of 2 are arithmetically distributed. For example, the maximum density threshold of a segment can be set to 1.0, the maximum density threshold of the entire array to 0.5, the minimum density threshold of the entire array to 0.2, and the minimum density of a segment to 0.1. If the PMA has 32 segments, then the maximum density threshold of a single segment is 1.0, of two segments is 0.9, of four segments is 0.8, of eight segments is 0.7, of 16 segments is 0.6, and of all 32 segments is 0.5.

More formally, upper and lower density thresholds for nodes at height ℓ are defined as follows:

$$\tau_\ell = \tau_h + (\tau_0 - \tau_h)(h-\ell)/h \tag{2}$$

$$\rho_\ell = \rho_h - (\rho_h - \rho_0)(h-\ell)/h. \tag{3}$$

Moreover,

$$2\rho_h < \tau_h, \tag{4}$$

because when we double the size of an array that becomes too dense, the new array must be within the density threshold.[1] Observe that any values of τ_0, τ_h, ρ_0, and ρ_h that satisfy (1)-(4) and enable the array to have size $\Theta(N)$ will work. The important requirement is that

$$\tau_{\ell-1} - \tau_\ell = O(\rho_\ell - \rho_{\ell-1}) = O(1/\log N)\,.$$

We now give more details about how to insert element y after an existing element x. If there is enough space in the leaf (segment) containing x, then we rearrange the elements within the leaf to make room for y. If the leaf is full, then we find the closest ancestor of the leaf whose density is within the permitted thresholds and rebalance. To delete an element x, we remove x from its segment. If the segment falls below its density threshold, then, as before, we find the smallest enclosing window whose density is within threshold and rebalance. As described above, if the *entire* array is above the maximum density threshold (resp., below the minimum density threshold), then we recopy the keys into a PMA of twice (resp., half) the size.

We introduce further notation. Let ***Cap***(u_ℓ) denote the number of array positions in node u_ℓ of height ℓ. Since there are 2^ℓ segments in the node, the capacity is $\Theta(2^\ell \log N)$. Let $\text{Gaps}(u_\ell)$ denote the number of gaps, i.e., unfilled array positions in node u_ℓ. Let $\text{Density}(u_\ell)$ denote the fraction of elements actually stored in node u_ℓ, i.e., $\text{Density}(u_\ell) = 1 - \text{Gaps}(u_\ell)/\text{Cap}(u_\ell)$.

Rebalance. We ***rebalance*** a node u_ℓ of height ℓ if u_ℓ is within threshold, but we detect that a child node $u_{\ell-1}$ is outside of threshold. Any node whose elements are rearranged in the process of a rebalance is ***swept.*** Thus, we ***sweep*** a node u_ℓ of height ℓ when we detect that a child node $u_{\ell-1}$ is outside of threshold, but now u_ℓ need not be within threshold. Note that with this rebalance scheme, this tree can be implicitly rather than explicitly maintained. In this case, a rebalance consists of two scans, one to the left and one to the right of the insertion point until we find a region of the appropriate density.

In a traditional PMA we rebalance evenly, whereas in the adaptive PMA we rebalance unevenly. The idea of the APMA is to store a smaller number of elements in the leaves in

[1] There are straightforward ways to generalize (4) to further reduce space usage. Introducing this generalization here leads to unnecessary complication in presentation.

which there have been many recent inserts. However, since we must maintain the bound of $O(\log^2 N)$ amortized element moves, we cannot let the density of any child node be too high or too low.

PROPERTY 1. *(rebalance property) After a rebalance, if each node u_ℓ (except the root of the rebalancing subtree) has density within u_ℓ's parent's thresholds, then we say that the rebalance satisfies the **rebalance property**. We say that a node u_ℓ is **within balance** or **well balanced** if u_ℓ is within its parent's thresholds.*

The following theorem shows if each rebalance satisfies the rebalance property, then we achieve good update bounds. The proof is essentially that in [Bender et al. 2000; 2005], but the rebalance property applies to a wide set of rebalancing schemes.

THEOREM 1. *If the rebalance in a PMA satisfies the rebalance property, then inserts and deletes take $O(\log^2 N)$ amortized element moves and $O(1+(\log^2 N)/B)$ amortized memory transfers.*

PROOF. Let u_ℓ be a node at level ℓ. A rebalance of u_ℓ is triggered by an insert or delete that pushes one descendant node u_i at each height $i = 0,\ldots,\ell-1$ above its upper threshold τ_i or below its lower threshold ρ_i. (If this were not the case, then we would rebalance a node of a lower height than ℓ.)

Consider one particular such node u_i. Before the sweep of u_i's parent u_{i+1},

$$\text{Density}(u_i) > \tau_i \quad \text{or} \quad \text{Density}(u_i) < \rho_i\,.$$

After the sweep of u_{i+1}, by the rebalance property,

$$\rho_{i+1} \le \text{Density}(u_i) \le \tau_{i+1}\,.$$

Therefore we need at least

$$(\tau_i - \tau_{i+1})\text{Cap}(u_i)$$

inserts or at least

$$(\rho_{i+1} - \rho_i)\text{Cap}(u_i)$$

deletes before the next sweep of node u_{i+1}. Therefore the amortized size of a sweep of node u_{i+1} per insert into child node u_i is at most

$$\begin{aligned}\max\left\{\frac{\text{Cap}(u_{i+1})}{(\tau_i-\tau_{i+1})\text{Cap}(u_i)}, \frac{\text{Cap}(u_{i+1})}{(\rho_{i+1}-\rho_i)\text{Cap}(u_i)}\right\} &= \max\left\{\frac{2}{\tau_i-\tau_{i+1}}, \frac{2}{\rho_{i+1}-\rho_i}\right\} \\ &= O(\log N).\end{aligned}$$

When we insert an element into the PMA, we actually insert into $h = \Theta(\log N)$ such nodes u_i, one at each level in the tree. Therefore the total amortized size of a rebalance per insertion into the PMA is $O(\log^2 N)$. Thus, the amortized number of element moves per insert is $O(\log^2 N)$. Because a rebalance is composed of a constant number of sequential scans, the amortized number of memory transfers per insert is $O(1+(\log^2 N)/B)$, as promised. □

Observe that Theorem 1 applies to both insertions and deletions; in contrast, we focus only on insertions in the rest of the paper, for the sake of simplicity. However, it is likely that, with only minor modifications to the predictor, the same bounds for common insertion

Algorithm 1 Predictor.insert(x)

```
1:  if $\exists$ a cell $c$ such that $c$.element $= x$ then
2:      SWAP($c$, $c$.nextcell)                  { If $c$ is not the head pointer. }
3:      $c$.count $\leftarrow$ $c$.count $+ 1$
4:      if $c$.count $> \log N$ then
5:          tail.count $\leftarrow$ tail.count $- 1$      { When $c$.count is at the maximum $\log N \ldots$ }
6:          $c$.count $\leftarrow$ $c$.count $- 1$ { Decrease the tail's count instead of increasing $c$.count. }
7:      end if
8:  else
9:      if head.nextcell $=$ tail then
10:         tail.count $\leftarrow$ tail.count $- 1$      { Decrease tail's count when no free space. }
11:     else
12:         head $\leftarrow$ head.nextcell
13:         head.element $\leftarrow$ $x$
14:         head.count $\leftarrow$ 1
15:         head.leaf $\leftarrow$ $x$.leaf       { In other cases, create a new cell for new element. }
16:     end if
17: end if
18: if tail.count $= 0$ then
19:     tail $\leftarrow$ tail.nextcell         { The tail cell is removed when its count drops to zero. }
20: end if
```

distributions can be made to apply to deletion distributions and to distributions combining both operations. There does not seem to be any significant additional difficulties in dealing with deletions.

Prediction. In a ***predictor*** data structure we keep track of a small collection of elements, called ***marker*** elements, that directly precede elements recently inserted into the APMA. The predictor stores a pointer to those leaf nodes of the APMA (i.e., $\Theta(\log N)$-sized segments of the array) that contain marker elements. For each marker element, we count the number of recently inserted elements that directly follow the marker.

We give terminology for prediction. For an element x, let ***insert number*** $I(x)$ denote a count from 0 to $\log N$ estimating the number of inserts after x in the last $O(\log^2 N)$ inserts. The predictor is designed so that

- $I(x)$ is always an underestimate of the number of inserts, and
- $I(x)$ never grows above $\log N$.

Below, we explain why and how these properties are enforced. Furthermore, if element x is not in the predictor, then we define $I(x) = 0$.

We now define the insert number $I(u_\ell)$ of a node u_ℓ at level ℓ in the APMA. Specifically, let insert number $I(u_\ell)$ be the sum of the insert numbers of elements in u_ℓ. When rebalancing a node, we reallocate elements unevenly among its descendant leafs according to their insert numbers. The larger the insert number, the fewer elements are allocated.

We now explain how the predictor determines (1) which elements to store as marker elements and (2) what the count numbers are for each element. To do so, we explain how to implement the predictor.

The predictor is a circular linked list, stored in an array. The predictor contains $\beta \log N$ cells, for constant β. Two pointers, a head pointer and a tail pointer, indicate the front and

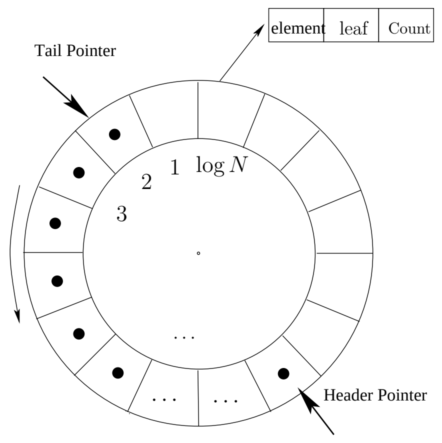

Fig. 1. The predictor. Each cell contains a marker element x, the leaf node in the APMA where x resides, and the count number $I(x)$.

the back of the linked list. Each cell in the circular linked list stores a marker element x. Associated with x are two pieces of data, (1) a pointer to the leaf node in the APMA where x currently resides and (2) the count number $I(x)$ (see Fig. 1).

When a new element x is determined to be a marker element, it is inserted into the predictor; x is inserted at the head of the linked list (where the head-pointer points). When an element x is no longer needed as a marker element, it is deleted from the predictor; before x is deleted, x will always be stored at the tail of the linked list (where the tail-pointer points).

When a new element y is inserted into the APMA after an element x, we first check whether x is a marker element (i.e., stored in the predictor). If x is a marker element, we store x and its auxiliary information one cell forward in the APMA (unless x is already at the head of the predictor). Let w be the element displaced by x. We store w (and auxiliary information) in the cell vacated by x. We also increase the element x's count number by 1 unless it is already at the maximum $O(\log N)$. Let z be the element stored in the tail of the predictor. If x is already at the maximum $O(\log N)$, then we decrement z's count number instead of incrementing x's count number. (This decrement is one reason why the count number of x is an underestimate.)

If x is not a marker element, then there are two cases. If there are empty cells in the predictor, then we store x at the head of the predictor. If there are no empty cells in the predictor, then we decrease the count number of z (the element stored in the tail) instead of storing x in the predictor. (This lack of space is another reason why the count number of x is an underestimate.)

This decrement may reduce the count number of z to 0. If so, we delete z from the predictor. A new free cell space is now available for future inserts.

The predictor algorithm is engineered to tolerate "random noise" in the insertions. By random noise, we mean that some of the insertions may not follow an insertion distribution (such as head insert, hammer insert, bulk insert, etc). Our guarantees still apply even if as much as a constant fraction of insertions are after random elements in the APMA. To understand why our predictor tolerates random noise, observe that a few arbitrary inserts will not be stored in the predictor unless the tail count drops below zero. If a poor choice of element is, in fact, stored in the predictor, it will soon be swapped to the tail if no new inserts follow.

Uneven Rebalance. Now we present the algorithm for uneven rebalance (See Algorithm 2). Assume that nodes $u_{\ell-1}$ and $v_{\ell-1}$ are left and right children of u_ℓ at level ℓ and that there are m ordered elements $\{x_1, x_2, \ldots, x_m\}$ stored in u_ℓ. The uneven rebalance performs as follows:

Algorithm 2 Rebalance.uneven(u_ℓ)

```
1: $u_{\ell-1} \leftarrow u_\ell$'s left child;
2: $v_{\ell-1} \leftarrow u_\ell$'s right child;
3: **if** ($u_{\ell-1}$ is empty) or ($v_{\ell-1}$ is empty) **then**
4:     return;
5: **end if**
6: splitnum $\leftarrow \max\{\mathrm{Cap}(u_{\ell-1})\rho_\ell, m - \mathrm{Cap}(v_{\ell-1})\tau_\ell\}$;
7: optvalue $\leftarrow \left| \frac{\sum_{j=1}^{\mathrm{splitnum}} I(x_j)}{\mathrm{Cap}(u_{\ell-1}) - \mathrm{splitnum}} - \frac{\sum_{j=\mathrm{splitnum}+1}^{m} I(x_j)}{\mathrm{Cap}(v_{\ell-1}) - (m - \mathrm{splitnum})} \right|$;
8: **for** $i =$ splitnum to $\min\{\mathrm{Cap}(u_{\ell-1})\tau_\ell, m - \mathrm{Cap}(v_{\ell-1})\rho_\ell\}$ **do**
9:     curvalue $\leftarrow \left| \frac{\sum_{j=1}^{i} I(x_j)}{\mathrm{Cap}(u_{\ell-1}) - i} - \frac{\sum_{j=i+1}^{m} I(x_j)}{\mathrm{Cap}(v_{\ell-1}) - (m - i)} \right|$;
10:    **if** optvalue $>$ curvalue **then**
11:        optvalue $\leftarrow$ curvalue;
12:        splitnum $\leftarrow i$;
13:    **end if**
14: **end for**
15: $u_{\ell-1} \leftarrow \{x_1, \ldots, x_{\mathrm{splitnum}}\}$;
16: $v_{\ell-1} \leftarrow \{x_{\mathrm{splitnum}+1}, \ldots, x_m\}$;
17: Rebalance.uneven($u_{\ell-1}$);
18: Rebalance.uneven($v_{\ell-1}$);
```

- If $I(x_i) = 0$ for all $i \in [1, m]$, then we perform an even rebalance for this node u_ℓ.
- Otherwise, we perform an uneven rebalance. Our uneven rebalance is designed so that, the bigger the insert numbers, the more gaps we leave. Specifically, we minimize the quantity

$$\left| \frac{I(u_{\ell-1})}{\mathrm{Gaps}(u_{\ell-1})} - \frac{I(v_{\ell-1})}{\mathrm{Gaps}(v_{\ell-1})} \right|, \tag{5}$$

subject to the constraint that the rebalance property must be satisfied. When we rebalance, we ***split at an element*** x_i, meaning that we put elements $\{x_1, \ldots, x_i\}$ in $u_{\ell-1}$ and $\{x_{i+1}, \ldots, x_m\}$ in $v_{\ell-1}$. The objective is to find the index i to minimize

$$\left| \frac{\sum_{j=1}^{i} I(x_j)}{\mathrm{Cap}(u_{\ell-1}) - i} - \frac{\sum_{j=i+1}^{m} I(x_j)}{\mathrm{Cap}(v_{\ell-1}) - (m-i)} \right|, \tag{6}$$

subject to the constraints that the densities of both left child and right child are within parent's threshold, i.e.,

$$i \in \left[\mathrm{Cap}(u_{\ell-1})\rho_\ell, \mathrm{Cap}(u_{\ell-1})\tau_\ell\right], \tag{7}$$

$$i \in \left[m - \mathrm{Cap}(v_{\ell-1})\tau_\ell, m - \mathrm{Cap}(v_{\ell-1})\rho_\ell\right]. \tag{8}$$

- We recursively allocate elements in $u_{\ell-1}$ and $v_{\ell-1}$'s child nodes and proceed down the tree until we reach the leaves. Once we know the number of elements in each leaf, we rebalance u_ℓ in one scan.

For example, in the insert-at-head case, the insert numbers of right descendants are always 0. Thus, minimizing the simplified objective quantity $|I(u_{\ell-1})/\mathrm{Gaps}(u_{\ell-1})|$ means maximizing $\mathrm{Gaps}(u_{\ell-1})$.

Now we show how to implement the rebalance so that there is no asymptotic overhead in the bookkeeping for the rebalance. Specifically, the number of element moves in the uneven rebalance is dominated by the size of the rebalancing node, as described in the following theorem:

THEOREM 2. *To rebalance a node u_ℓ at level ℓ unevenly requires $O(Cap(u_\ell))$ operations and $O(1 + Cap(u_\ell)/B)$ memory transfers.*

PROOF. There are three steps to rebalancing a node u_ℓ unevenly. First, we check the predictor to obtain the insert numbers of the elements located in all descendant nodes of u_ℓ. Because the size of the predictor is $O(\log N)$, this step takes $O(\log N)$ operations and $O(1 + (\log N)/B)$ memory transfers. Second, we recursively determine the number of elements to be stored in u_ℓ's children, grandchildren, etc., down to descendent leaves. Naively, this procedure uses $O(\ell \mathrm{Cap}(u_\ell))$ operations and $O(1 + \ell \mathrm{Cap}(u_\ell)/B)$ memory transfers; below we show how to perform this procedure in $O(\mathrm{Cap}(u_\ell))$ operations and $O(1 + \mathrm{Cap}(u_\ell)/B)$ memory transfers. Third, we scan the node u_ℓ putting each element into the correct leaf node. Thus, this last step also takes $O(\mathrm{Cap}(u_\ell))$ operations and $O(1 + \mathrm{Cap}(u_\ell)/B)$ memory transfers.

We now show how to implement the second step efficiently. We call the elements in the predictor ***weighted*** elements and the remaining elements ***unweighted***. Recall that only weighted elements have nonzero insert numbers. In the first step, we obtain all information about which elements are weighted. Then, we start the second step, which is recursive. At the first recursive level, we determine which elements are allocated to the left and right children of u_ℓ, i.e., we find the index i minimizing (6). At first glance, it seems necessary to check all indices i in order to get the minimum, which takes $O(\mathrm{Cap}(u_\ell))$ operations, but we can do better. Observe that when the index i is in a sequence of unweighted elements between two weighted elements, the numerator in (6) does not change. Only the denominator changes, and it does so continuously. So in order to minimize (6) at the first recursive level, it is not necessary to check all elements in node u_ℓ. It is enough to check which two contiguous weighted elements the index i is between such that (6) is minimized. Since there are at most $O(\log N)$ weighted elements, the number of operations at each recursive level is at most $O(\log N)$. Furthermore, because there are ℓ recursive levels, the number of operations in the whole recursive step is at most $O(\ell \log N)$, which is less than $O(\mathrm{Cap}(u_\ell))$. By storing these weighted elements contiguously during the rebalance, we obtain $O(1 + \mathrm{Cap}(u_\ell)/B)$ memory transfers. □

3. ANALYSIS OF SEQUENTIAL AND HAMMER INSERTIONS

In this section we first analyze the adaptive PMA for the sequential insert pattern, where inserts occur at the front of the PMA. Then we generalize the result to hammer inserts.

For sequential inserts, we prove the following theorem:

THEOREM 3. *For sequential inserts, the APMA has $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ amortized memory transfers.*

We give some notation. In the rest of this section, we assume that u_ℓ is the leftmost node at level ℓ and $v_{\ell-1}$ is the right child of u_ℓ. Recall that leaves have height 0. Suppose that we insert N elements in the front of an array of size cN ($c > 1$). Since we always insert elements at the front, rebalances occur only at the leftmost node u_ℓ ($0 \le \ell \le h$). If we know the number of sweeps of u_ℓ in the process of inserting these N elements, then we also know the total number of moves.

In order to bound the number of sweeps at each level, we need more notation. For $\kappa \le \ell$, let $\mathcal{N}_\kappa(\ell,i)$ be the number of sweeps of the leftmost node u_κ at level κ between the $(i-1)$th sweep and the ith sweep of node u_ℓ. We imagine a virtual parent node u_{h+1} of the root node u_h, where u_{h+1} has size $2cN$. Thus, the time when the root node u_h reaches its upper threshold τ_h, after we insert N elements, is the time when the virtual parent node performs the first rebalance. Thus, $\mathcal{N}_\kappa(h+1,1)$ is the number of sweeps of node u_κ at level κ during the insertion of these N elements ($0 \le \kappa \le h$). Since each sweep of u_κ costs $2^\kappa \log N$ moves, the total number of moves is:

$$\sum_{\kappa=0}^{h} \mathcal{N}_\kappa(h+1,1)2^\kappa \log N.$$

This quantity is the sum of the sweep costs at each level, until the virtual node needs its first rebalance. Thus, the amortized number of element moves is

$$\frac{1}{N}\sum_{\kappa=0}^{h} \mathcal{N}_\kappa(h+1,1)2^\kappa \log N. \tag{9}$$

Sequential Inserts with Only Upper Thresholds. For pedagogical reasons, we now consider the simpler case of a PMA with no lower-bound thresholds and show that Theorem 3 holds in this special case. This lack of lower-bound thresholds makes it significantly easier to achieve the bounds from Theorem 3. By providing this simpler analysis we give insight into the origin of Theorem 3's bounds and why the subsequent analysis is more complicated.

LEMMA 4. *For sequential inserts, the APMA with no lower-bound thresholds has $O(\log N)$ amortized element moves and $O(1+\log N/B)$ amortized memory transfers.*

PROOF. Recall that $\mathcal{N}_\kappa(\ell,1)$ is the number of sweeps of the leftmost child u_κ at level κ until ancestor node u_ℓ performs its first rebalance. Observe that just before u_ℓ performs the first rebalance, $u_{\ell-1}$ reaches its threshold $\tau_{\ell-1}$. We want to find the number of sweeps of u_κ before $u_{\ell-1}$ reaches its upper threshold $\tau_{\ell-1}$.

We decompose this process into two phases. Phase 1 ends before the first rebalance of node $u_{\ell-1}$ when we have $\tau_{\ell-2}2^{\ell-2}$ elements in the left child $u_{\ell-2}$ of $u_{\ell-1}$ and 0 elements in the right child $v_{\ell-2}$ of $u_{\ell-1}$ (see Fig. 2). According to our uneven-rebalance strategy, since

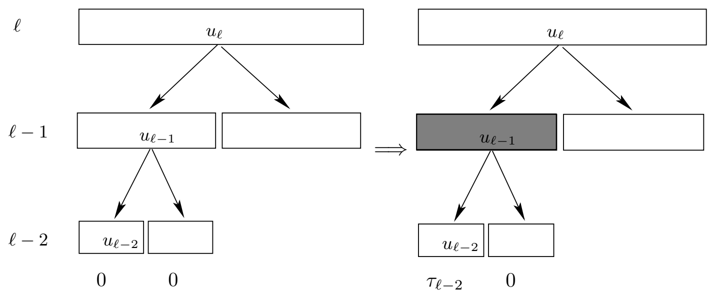

Fig. 2. In the simple case, Phase 1 of node u_ℓ starts from Density$(u_{\ell-2}) = 0$ (left) and ends at Density$(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

all inserts are to the left child, we allocate $\tau_{\ell-1}2^{\ell-2}$ elements to $v_{\ell-2}$ and $(\tau_{\ell-2}-\tau_{\ell-1})2^{\ell-2}$ elements to $u_{\ell-2}$ at the end of Phase 1, i.e., we give the maximum allowed number of elements to the right child. Now we consider Phase 2, which takes place between the first rebalance and the second rebalance of $u_{\ell-1}$ (see Fig. 3). Since the right child $v_{\ell-2}$ of $u_{\ell-1}$ already has density $\tau_{\ell-1}$, when $u_{\ell-2}$ reaches its threshold $\tau_{\ell-2}$ again, the density of $u_{\ell-1}$ is $(\tau_{\ell-2}+\tau_{\ell-1})/2 > \tau_{\ell-1}$ at the end of Phase 2, which is above its upper threshold.

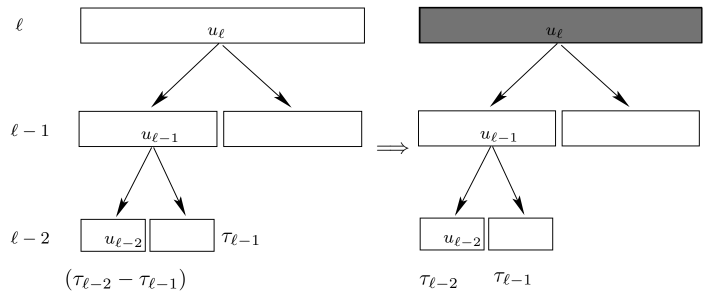

Fig. 3. In the simple case, Phase 2 of node u_ℓ starts from Density$(u_{\ell-2}) = \tau_{\ell-2}-\tau_{\ell-1}$ (left) and ends at Density$(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

To summarize, the first time that we rebalance $u_{\ell-1}$ is when we move elements from $u_{\ell-2}$ into $v_{\ell-2}$. This rebalance is triggered because $u_{\ell-2}$ is above its threshold. The next time $u_{\ell-2}$ goes above its threshold $\tau_{\ell-2}$, $u_{\ell-1}$ is also above its threshold $\tau_{\ell-1}$, and we trigger the first rebalance of u_ℓ. Thus, there are at most two sweeps of node $u_{\ell-1}$ before it reaches its threshold $\tau_{\ell-1}$. That is

$$\mathcal{N}_\kappa(\ell,1) \le \mathcal{N}_\kappa(\ell-1,1) + \mathcal{N}_\kappa(\ell-1,2). \tag{10}$$

To calculate (9), we first show that $\mathcal{N}_\kappa(\ell-1,2) < \mathcal{N}_\kappa(\ell-1,1)$. Recall that $\mathcal{N}_\kappa(\ell-1,2)$ is the number of sweeps of the leftmost child u_κ at level κ between ancestor node $u_{\ell-1}$'s first sweep (rebalance) and second sweep. The above inequality is true because at the end of both phases $u_{\ell-2}$ reaches its threshold, but the first phase starts with $u_{\ell-2}$ having density 0 (an empty data structure), and the second phase starts with $u_{\ell-2}$ having density $\tau_{\ell-2}-\tau_{\ell-1}$. Thus, by plugging $\mathcal{N}_\kappa(\ell-1,2) < \mathcal{N}_\kappa(\ell-1,1)$ in (10), we have the recurrence

$$\mathcal{N}_\kappa(\ell,1) \le 2\mathcal{N}_\kappa(\ell-1,1).$$

The amortized number of element moves is

$$\begin{aligned}\frac{1}{N}\sum_{\kappa=0}^{h}\mathcal{N}_\kappa(h+1,1)2^{\kappa}\log N &= \sum_{\kappa=0}^{h}\mathcal{N}_\kappa(h+1,1)2^{\kappa-h}\\ &\le \sum_{\kappa=0}^{h}\left[2^{h-\kappa+1}\mathcal{N}_\kappa(\kappa,1)\right]2^{\kappa-h}\\ &= \sum_{\kappa=0}^{h}2 = O(\log N).\end{aligned}$$

□

Sequential Inserts in APMA with Lower and Upper Thresholds. We now consider the general case of a PMA with both the lower- and upper-bound thresholds and are ready to prove Theorem 3.

PROOF OF THEOREM 3: The proof is a generalization of the proof of Lemma 4; we bound $\mathcal{N}_\kappa(\ell,1)$, the number of sweeps of the leftmost child u_κ at level κ until the ancestor node u_ℓ performs the first rebalance. The difficulty with both the lower- and upper-bound thresholds is that we must decompose the time before the first rebalance of u_ℓ into more than 2 phases, and thus we obtain a more complicated recurrence to solve. We decompose this process into three phases. ***Phase** i **of node*** u_ℓ $(1 \le i \le 3)$, starts after the $(i-1)$th sweep of $u_{\ell-1}$ and ends at the ith sweep of $u_{\ell-1}$. At the end of the last phase, u_ℓ performs its first rebalance, which is the third sweep of $u_{\ell-1}$. Thus, we have at most three sweeps of node $u_{\ell-1}$ before the first rebalance of u_ℓ:

$$\mathcal{N}_\kappa(\ell,1) \le \mathcal{N}_\kappa(\ell-1,1)+\mathcal{N}_\kappa(\ell-1,2)+\mathcal{N}_\kappa(\ell-1,3).$$

Now we prove the above claim analyzing the densities in each phase.

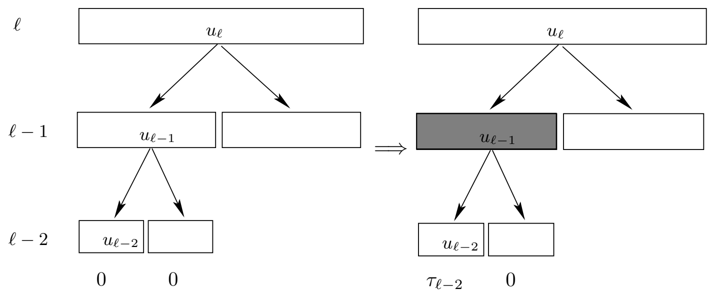

Fig. 4. Phase 1 of node u_ℓ starts from Density$(u_{\ell-2}) = 0$ (left) and ends at Density$(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

I) We consider the densities of child nodes $u_{\ell-2}$ and $v_{\ell-2}$ of $u_{\ell-1}$ at the end of Phase 1. The first rebalance of $u_{\ell-1}$ occurs (see Fig. 4) when $u_{\ell-2}$ reaches its upper threshold $\tau_{\ell-2}$. For sequential inserts, we allocate as many free spaces as possible to $u_{\ell-2}$, while ensuring that $u_{\ell-2}$ and $v_{\ell-2}$ have densities between $\rho_{\ell-1}$ and $\tau_{\ell-1}$. Thus, after the first rebalance, which happens after $\tau_{\ell-2}\mathrm{Cap}(u_{\ell-2})$ inserts, we have densities:

$$\begin{aligned}\mathrm{Density}(u_{\ell-2}) &= \rho_{\ell-1},\\ \mathrm{Density}(v_{\ell-2}) &= \tau_{\ell-2}-\rho_{\ell-1}.\end{aligned}$$

It is immediate that the density setting of $u_{\ell-2}$ is legal; we now explain why the above density setting of $v_{\ell-2}$ is legal, i.e., satisfies the rebalance property. Notice that $\rho_{\ell-1} \le \tau_{\ell-2}-\rho_{\ell-1} \le \tau_{\ell-1}$, since $2\rho_{\ell-1} \le \tau_{\ell-1} < \tau_{\ell-2}$ by (1) and (4) and $\tau_{\ell-2}-\tau_{\ell-1} = O(1/\log N) < \rho_{\ell-1}$ by (1) and (2).

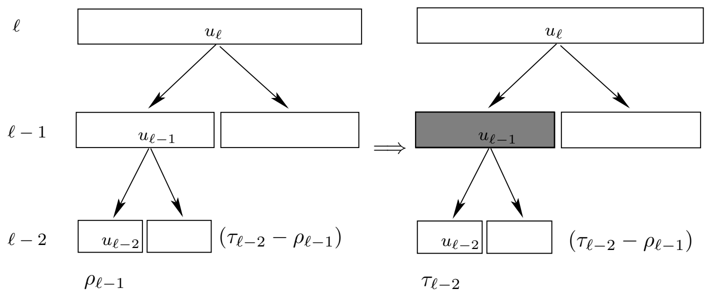

Fig. 5. Phase 2 of node u_ℓ starts from $\mathrm{Density}(u_{\ell-2}) = \rho_{\ell-1}$ (left) and ends at $\mathrm{Density}(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

II) We now consider the densities of child nodes $u_{\ell-2}$ and $v_{\ell-2}$ at the end of Phase 2. When $u_{\ell-2}$ reaches its threshold again, Phase 2 of node u_ℓ ends (see Fig. 5). After $u_{\ell-1}$ does the second rebalance, which happens after $(\tau_{\ell-2}-\rho_{\ell-1})\mathrm{Cap}(u_{\ell-2})$ inserts, we have densities:

$$\begin{aligned}\mathrm{Density}(u_{\ell-2}) &= 2\tau_{\ell-2}-\rho_{\ell-1}-\tau_{\ell-1},\\ \mathrm{Density}(v_{\ell-2}) &= \tau_{\ell-1}.\end{aligned}$$

It is immediate that the density setting of $v_{\ell-2}$ is legal; we now show that the density setting of $u_{\ell-2}$ is legal. Notice that $\rho_{\ell-1} < 2\tau_{\ell-2}-\rho_{\ell-1}-\tau_{\ell-1} < \tau_{\ell-1}$, because $2\rho_{\ell-1} < \tau_{\ell-2} < \tau_{\ell-2}+(\tau_{\ell-2}-\tau_{\ell-1})$ by (1) and (4) and $2(\tau_{\ell-2}-\tau_{\ell-1}) = O(1/\log N) < \rho_{\ell-1}$ by (1) and (2).

III) Now we consider the densities of child nodes $u_{\ell-2}$ and $v_{\ell-2}$ at the end of Phase 3. When $u_{\ell-2}$ reaches its threshold a third time, which happens after $(\tau_{\ell-1}-\tau_{\ell-2}+\rho_{\ell-1})\mathrm{Cap}(u_{\ell-2})$ inserts, Phase 3 of node u_ℓ ends (see Fig. 6). When $u_{\ell-1}$ does the third sweep, the density of $u_{\ell-1}$ is $(\tau_{\ell-2}+\tau_{\ell-1})/2 > \tau_{\ell-1}$, so $u_{\ell-1}$ is above threshold. Thus, the end of Phase 3 is the first rebalance of u_ℓ.

Thus, there are at most three sweeps of $u_{\ell-1}$ before the first rebalance of u_ℓ, that is,

$$\mathcal{N}_\kappa(\ell,1) \le \mathcal{N}_\kappa(\ell-1,1)+\mathcal{N}_\kappa(\ell-1,2)+\mathcal{N}_\kappa(\ell-1,3). \tag{11}$$

We cannot simply use the bound $\mathcal{N}_\kappa(\ell,1) \le 3\mathcal{N}_\kappa(\ell-1,1)$ for our analysis, since this bound naively leads to $O(N^{\log(3/2)})$ amortized moves, which is far from our goal of $O(\log N)$.

To establish our bound, we prove the following recurrences for Phase 2 and Phase 3:

$$\mathcal{N}_\kappa(\ell-1,2) \le 2\mathcal{N}_\kappa(\ell-2,2), \tag{12}$$

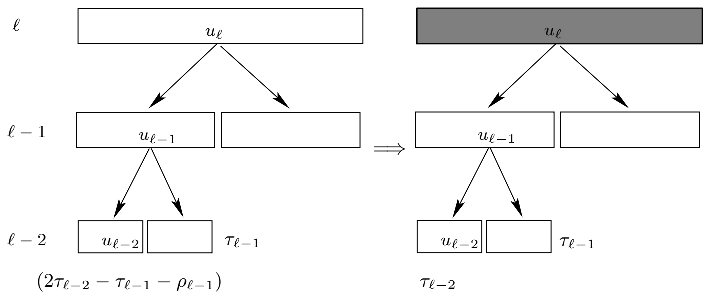

Fig. 6. Phase 3 of node u_ℓ starts from Density$(u_{\ell-2}) = 2\tau_{\ell-2} - \tau_{\ell-1} - \rho_{\ell-1}$ (left) and ends at Density$(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

and

$$\mathcal{N}_\kappa(\ell-1,3) \leq \mathcal{N}_\kappa(\ell-1,2). \tag{13}$$

Solving (11), (12), and (13) will yield the desired bound.

We already showed (11); now we show (12). We proceed by breaking Phase 2 into two subphases. The first subphase begins when Phase 2 begins, i.e., after the first rebalance of $u_{\ell-1}$, and it ends after the next sweep of $u_{\ell-2}$. The second subphase begins when the first subphase ends, and it ends after the next another sweep of $u_{\ell-2}$. We will show that at the end of Subphase 2, $u_{\ell-2}$ is above threshold, meaning that Subphase 2 ends with a sweep of $u_{\ell-1}$, i.e., Phase 2 ends as well.

- At the beginning of Subphase 1, node $u_{\ell-3}$ has density $\rho_{\ell-2}$ by the rebalance property. (Since insertions are at the beginning of the array, we want $u_{\ell-3}$ to be as sparse as possible, and the rebalance property says that after a rebalance Density$(u_{\ell-3}) \geq \rho_{\ell-2}$.) The sweep of $u_{\ell-2}$ is triggered once the density of $u_{\ell-3}$ reaches $\tau_{\ell-3}$ (see Fig. 7). At the end of Subphase 1, after $(\tau_{\ell-3}-\rho_{\ell-2})Cap(u_{\ell-3})$ inserts, the density of $u_{\ell-3}$ and $v_{\ell-3}$ are:

$$\begin{aligned}\text{Density}(u_{\ell-3}) &= 2\rho_{\ell-1} - \rho_{\ell-2} + \tau_{\ell-3} - \tau_{\ell-2},\\ \text{Density}(v_{\ell-3}) &= \tau_{\ell-2}.\end{aligned}$$

  It is immediate that the density of $v_{\ell-3}$ is legal; we show that the density of $u_{\ell-3}$ is legal too. Notice that $\rho_{\ell-2} < 2\rho_{\ell-1} - \rho_{\ell-2} + \tau_{\ell-3} - \tau_{\ell-2} < \tau_{\ell-2}$, because $2\rho_{\ell-2} < 2\rho_{\ell-1}$ and $\tau_{\ell-2} < \tau_{\ell-3}$ by (1) and $2\rho_{\ell-1} < \tau_{\ell-1} < \tau_{\ell-2}$ and $\tau_{\ell-3} - \tau_{\ell-2} = O(1/\log N) < \rho_{\ell-2}$ by (1) and (4).
  We now show that the number of sweeps of u_κ in Subphase 1 is equal to $\mathcal{N}_\kappa(\ell-2,2)$. Observe that Subphase 1 is exactly Phase 2 of the node $u_{\ell-1}$ because they both start with the node $u_{\ell-3}$ having density $\rho_{\ell-2}$ and end with the node $u_{\ell-3}$ having density $\tau_{\ell-3}$. Although in Subphase 1 and Phase 2 of node $u_{\ell-1}$, node $v_{\ell-3}$ has different densities, this difference does not matter because the density of $v_{\ell-3}$ does not affect when Subphase 1 and Phase 2 of node $u_{\ell-1}$ end.
- At the beginning of Subphase 2, $u_{\ell-3}$ has density $2\rho_{\ell-1} - \rho_{\ell-2} + \tau_{\ell-3} - \tau_{\ell-2} > \rho_{\ell-2}$, and the subsequent sweep of $u_{\ell-2}$ is triggered once the density of $u_{\ell-3}$ reaches $\tau_{\ell-3}$ again (see Fig. 8). Since the density of $v_{\ell-3}$ is $\tau_{\ell-2}$, the density of $u_{\ell-2}$ is $(\tau_{\ell-3} + \tau_{\ell-2})/2 > \tau_{\ell-2}$ at the end of Subphase 2, so $u_{\ell-2}$ is above its upper threshold. Thus, the end of Subphase 2 is the sweep of $u_{\ell-1}$.

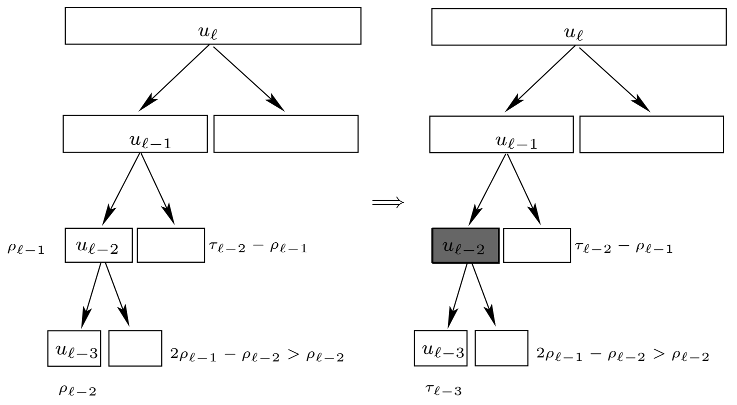

Fig. 7. Subphase 1 starts from Density$(u_{\ell-3}) = \rho_{\ell-2}$ (left) and ends at Density$(u_{\ell-3}) = \tau_{\ell-3}$ (right). The shaded region is rebalanced.

We now prove that the number of sweeps of u_κ in Subphase 2 is less than $\mathcal{N}_\kappa(\ell-2,2)$, because both Subphase 2 and Phase 2 of node $u_{\ell-1}$ end with node $u_{\ell-3}$ reaching its upper threshold $\tau_{\ell-3}$, but Subphase 2 starts with node $u_{\ell-3}$ having density greater than $\rho_{\ell-2}$ while Phase 2 of node $u_{\ell-1}$ starts with node $u_{\ell-3}$ having density $\rho_{\ell-2}$.

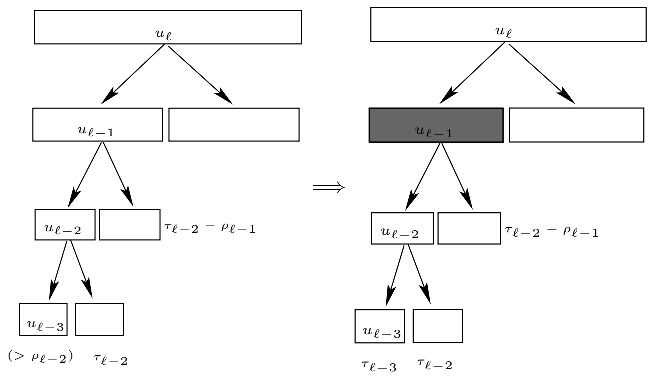

Fig. 8. Subphase 2 starts from Density$(u_{\ell-3}) \geq \rho_{\ell-2}$ (left) and ends at Density$(u_{\ell-3}) = \tau_{\ell-3}$ (right). The shaded region is rebalanced.

Thus, there are at most two subphases in Phase 2 of node u_ℓ and each subphase has the number of sweeps of node u_κ at most $\mathcal{N}_\kappa(\ell-2,2)$, which shows (12). Since Recurrence (12) has the base case $\mathcal{N}_\kappa(\kappa,2) = 1$, we obtain the solution

$$\mathcal{N}_\kappa(\ell-1,2) \leq 2^{\ell-\kappa-1}. \tag{14}$$

Now we establish the recurrence in (13). Both Phase 2 and Phase 3 end with node $u_{\ell-2}$ reaching its upper threshold $\tau_{\ell-2}$, while Phase 3 starts with the node $u_{\ell-2}$ having density $2\tau_{\ell-2}-\tau_{\ell-1}-\rho_{\ell-1} > \rho_{\ell-1}$. Phase 2 starts with node $u_{\ell-2}$ having density $\rho_{\ell-1}$.

We now establish the desired bound. Plugging (14) and (13) into (11), we have

$$\begin{aligned}\mathcal{N}_\kappa(\ell,1) &\le \mathcal{N}_\kappa(\ell-1,1)+\mathcal{N}_\kappa(\ell-1,2)+\mathcal{N}_\kappa(\ell-1,3)\\ &\le \mathcal{N}_\kappa(\ell-1,1)+2\mathcal{N}_\kappa(\ell-1,2)\\ &\le \mathcal{N}_\kappa(\ell-1,1)+2\cdot 2^{\ell-\kappa-1}\\ &\le 2^{\ell-\kappa+1}. \end{aligned} \tag{15}$$

Finally, the amortized number of moves is

$$\frac{1}{N}\sum_{\kappa=0}^{h}\mathcal{N}_\kappa(h+1,1)2^\kappa \log N = \sum_{\kappa=0}^{h}\mathcal{N}_\kappa(h+1,1)2^{\kappa-h}$$
$$\le \sum_{\kappa=0}^{h}(2^{h-\kappa+2})2^{\kappa-h} = \sum_{\kappa=0}^{h}4 = O(\log N).$$

Observe that after any insert the elements are moved from a contiguous group, and the moves can be performed with a constant number of scans. Therefore the amortized number of memory transfers is $O(1+(\log N)/B)$. □

Hammer Inserts. We now consider the hammer insertion distribution, where we always insert the elements at the same rank. We show that the analysis from sequential insertion distribution (Theorem 3) applies here.

THEOREM 5. *When inserted elements have fixed rank (hammer inserts), the APMA has $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ amortized memory transfers.*

PROOF. In the hammer-insert case, we always insert new elements after a given element x. Notice that in the rebalancing subtree rooted at u_ℓ, there is a unique path from the leaf node containing the element x to the root node u_ℓ. Let node u_i ($i \le \ell$) be the ancestor of x at level i, and let v_i be u_i's sibling. An important difference between this proof and the proof of Theorem 3 is that u_{i-1} and sibling v_{i-1} may now be either left or right children of u_i for $i<\ell$.

Recall that, as in the proof of Theorem 3, for level $\kappa \le \ell$, $\mathcal{N}_\kappa(\ell,t)$ is the number of sweeps of the leftmost node u_κ at level κ between the $(t-1)$th sweep and the tth sweep of node u_ℓ.

Intuitively, we want to use a similar argument as in the proof of Theorem 3, to show that $\mathcal{N}_\kappa(\ell,1)$ is bounded as in (15), up to a constant factor, that is, for constant β,

$$\mathcal{N}_\kappa(\ell,1) \le \beta 2^{\ell-\kappa+1}.$$

This approach comes close to working, but requires a much more technical generalization. In particular, as we show, Recurrences (11) and (13) still hold, but there is one value of $i+1$ below which Recurrence (12) might not.

In the following, we explain why there may exist a node u_{i+1} below which Recurrence (12) does not hold. Then we explain that

$$\mathcal{N}_\kappa(i+1,2) = O(2^{i+1-\kappa}),$$

which is the same as the solution of Recurrence (12) up to a constant factor. Finally, we explain why the analysis from Theorem 3 still applies even when there exists such a node u_{i+1}.

We first explain why there may exist a node u_{i+1} for which Recurrence (12) does not hold. To do so, we examine the density of the child u_i after the first sweep of u_{i+1} and demonstrate that Density(u_i) can be different with sequential inserts and hammer inserts. With sequential inserts, a rebalance tries to put as few elements as possible in u_i and as many elements as possible in v_i without disobeying the upper and lower density thresholds. With hammer inserts, we also want u_i to be as sparse as possible while still maintaining the rebalance property.

But now we have an additional constraint, the ***hammer constraint***, that node x must remain in u_i. What we mean by this additional constraint is the following. Suppose that u_i is a left child, and v_i is a right child. In a rebalance we try to put as few elements as possible in u_i and as many elements as possible in v_i. But if the last element in u_i is x then we cannot reduce the density of u_i any further — the next element to move into v_i is x, but then v_i becomes u_i.

To summarize, there are two cases in which hammer inserts may differ from sequential inserts. The first case is when u_i is a left child and x is the rightmost element in u_i after a sweep of u_{i+1}. The second case is when u_i is a right child and x is the leftmost element in u_i after a sweep of u_{i+1}. In both cases Recurrence (12) may not hold for u_{i+1}. (If x is not in one of these two positions at the end of a rebalance, then the critical constraint is the rebalance property rather than the hammer constraint, as with sequential inserts.)

We now explain that in both cases, the number of sweeps of u_κ between the first sweep and the second sweep of u_{i+1}, $\mathcal{N}_\kappa(i+1,2)$, still has the solution $O(2^{i+1-\kappa})$. When node u_i is a right child and x is the leftmost element in u_i, the bound follows from the analysis in Theorem 3 because the insert pattern of u_i matches the sequential-insert case. The difficult case is when u_i is a left child and x is the rightmost element in u_i after the first sweep of u_{i+1}. We call this the ***tail-insert case***. This case corresponds to a stage beginning after any sweep of u_{i+1} when the element x is the rightmost element in u_i and ending when node u_i reaches its upper threshold, i.e., at the next sweep of u_{i+1}. We call this interval the ***tail-insert stage of*** u_i. Below, we give a bound on the number of sweeps of u_κ in the tail-insert case.

We use the following claim:

CLAIM 6. *Consider the tail-insert stage of u_i: the stage starts after one sweep of u_{i+1} and ends just before the next sweep of u_{i+1}, and x is the rightmost element in u_i at the beginning of the stage. Then the number of sweeps of node u_κ during the stage is $O(2^{i-\kappa})$.*

We prove the above claim in the appendix. The proof is similar to Theorem 3, but significantly more technical.

Finally, we show why, given Claim 6, the analysis from Theorem 3 applies to hammer inserts. Recurrence (12) is true above an intermediate node u_i, that is,

$$\mathcal{N}_\kappa(\ell-1,2) \le 2^{\ell-i-2}\mathcal{N}_\kappa(i+1,2).$$

Moreover, by Claim 6,

$$\mathcal{N}_\kappa(i+1,2) \le \beta 2^{i+1-\kappa}$$

for some constant β at node u_i. Therefore,

$$\mathcal{N}_\kappa(\ell-1,2) \le \beta 2^{\ell-\kappa-1}.$$

Thus, the solution for Recurrence (11) is

$$\mathcal{N}_\kappa(\ell, 1) \le 2^{\ell-\kappa+1}\beta,$$

and the theorem follows.

□

4. ANALYSIS FOR RANDOM AND BULK INSERTIONS

In the previous section we analyze the sequential and hammer insertion distributions, where the inserts hammer on one part of the PMA. In this section we first analyze random insertion distribution, where we insert after random elements in the array. Then we generalize all of these distributions and consider the bulk insertion distribution.

The bulk insertion distribution for function N^α, $0 \le \alpha \le 1$, is defined as follows: pick a random element and insert N^α elements after it; then pick another element and repeat. Bulk insert generalizes all distributions seen so far: For $\alpha = 0$, we have random inserts, and for $\alpha = 1$, we have sequential or hammer inserts.

Random Inserts. We now give the performance for the traditional PMA and APMA with random inserts. In the traditional PMA or APMA, each insertion causes only a small number of elements to be moved or triggers a recopying of the entire array.

THEOREM 7 [ITAI ET AL. 1981; BENDER ET AL. 2004]. *Consider random insertions into a traditional PMA or APMA, in which each new element is inserted after a random element in the PMA or APMA. Whenever the density of the entire array is below the maximum density threshold, then each insert causes $O(\log N)$ element moves and $O(1+(\log N)/B)$ memory transfers with high probability, i.e., probability polynomially small in N. Specifically, each insert causes $O(\alpha \log N)$ element moves and $O(1+\alpha(\log N)/B)$ memory transfers with probability at least $1-1/N^\alpha$.*

Even simpler rebalance schemes perform well under random inserts, as shown in [Itai et al. 1981; Bender et al. 2004]. Publications [Itai et al. 1981; Bender et al. 2004] show that there are $O(\log N)$ moves with high probability for random inserts, even with the following simple rebalance procedure: When we insert an element y after an element x, we simply push the elements to the right or left to make room for y. The maximum number of elements moved is $O(\log N)$ with high probability. Thus, for the traditional PMA, as long as the density thresholds in the leaves is a constant less than 1, we need no big rebalances in the tree.

Bulk Inserts. For bulk inserts, we have the following theorem:

THEOREM 8. *For bulk inserts with $f(N) = N^\alpha$ ($0 \le \alpha \le 1$), the APMA achieves $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ memory transfers.*

The intuition for Theorem 8 is as follows: Conceptually, we divide the virtual tree into a top tree with $\Theta(N/(f(N)\log N))$ leaves, each of which is the root of a bottom tree T with $\Theta(f(N))$ leaves, i.e., $\Theta(f(N)\log N)$ array positions. Thus, we split the virtual tree at height $h' = \lceil \alpha \log N \rceil$. Bulk inserts can be analyzed by looking at the process as a combination of random and hammer inserts: random inserts in the top tree A with big leaf nodes of size $f(N)\log N$ and hammer inserts in a bottom tree T of size $f(N)\log N$. In an insertion, we

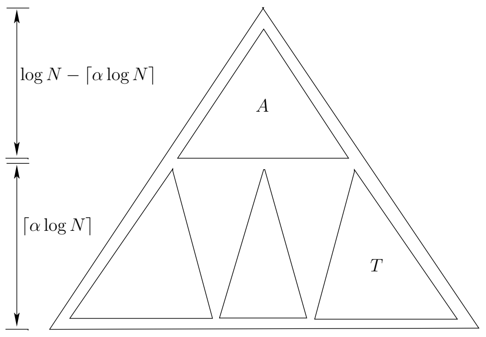
$\log N - \lceil \alpha \log N \rceil$
A
$\lceil \alpha \log N \rceil$
T

Fig. 9. An illustration showing the tree divided at height $\lceil \alpha \log N \rceil$.

randomly choose a leaf node of top tree A and do a hammer insert at the bottom subtree of the chosen leaf node of A.

We first show that $f(N) = N^\alpha$ $(0 \leq \alpha \leq 1)$ hammer inserts into T costs $O(\log N)$ amortized moves when all the nodes are well balanced. Then, we explain that these $f(N)$ inserts trigger at most one rebalance in the top tree A. Thus, from the point of view of A, there is a big element of size $f(N)$ inserted, and this big insert costs $O(\log N)$ amortized moves in the leaf node.

We prove the following lemma for $f(N) = N^\alpha$.

LEMMA 9. *Consider inserting $f(N) = N^\alpha$ elements after a fixed element x in subtree T of size $N^\alpha \log N$. Suppose that at the beginning of these insertions, each node in T is well balanced. Then, the amortized number of moves is $O(\log N)$ and the amortized number of memory transfers is $O(1 + (\log N)/B)$.*

PROOF. We first show that all sweeps during the insertions of N^α elements occur in subtree T. Because the root node is well balanced, the density of the root is at most $\tau_{h'+1}$. Thus, before root $u_{h'}$ goes outside of its upper threshold, we can insert at least $(\tau_{h'} - \tau_{h'+1})(N^\alpha \log N) = \Theta(N^\alpha)$ elements without triggering sweeps above level h'.

Now we give some assumptions and notation. For simplicity we assume that there are sequential insertions within T. (We know from the proof of Theorem 5 that sequential inserts and hammer inserts have the same analysis except at one level of the recurrence relations.) Now we denote the leftmost node in T at level ℓ as u_ℓ. As in the proof of Theorem 5, we use $\mathcal{N}_\kappa(\ell, i)$ to denote the number of sweeps of node u_κ at level κ between the $(i-1)$th and ith sweep of u_ℓ. Thus, the amortized number of element moves is at most

$$\frac{1}{N^\alpha} \sum_{\kappa=0}^{h'} \mathcal{N}_\kappa(h'+1, 1) 2^\kappa \log N. \tag{16}$$

We bound (16) by considering the worst case when all u_ℓ have density as high as $\tau_{\ell+1}$, $0 \leq \ell \leq h'$. The time when u_ℓ does its first rebalance is the time when $u_{\ell-1}$ reaches its upper threshold $\tau_{\ell-1}$. This period can be decomposed into two phases, as before:

- Phase 1 of node u_ℓ starts with node $u_{\ell-2}$ having density $\tau_{\ell-1}$ and node $v_{\ell-2}$ having density $\tau_{\ell+1}$. Phase 1 ends with node $u_{\ell-2}$ having density $\tau_{\ell-2}$. Thus, after the first rebalance of $u_{\ell-1}$ (see Fig. 10), which occurs after $(\tau_{\ell-2} - \tau_{\ell-1})\mathrm{Cap}(u_{\ell-2})$ inserts, we have densities:

$$\begin{aligned}\text{Density}(u_{\ell-2}) &= \tau_\ell,\\ \text{Density}(v_{\ell-2}) &= \tau_{\ell-1}.\end{aligned}$$

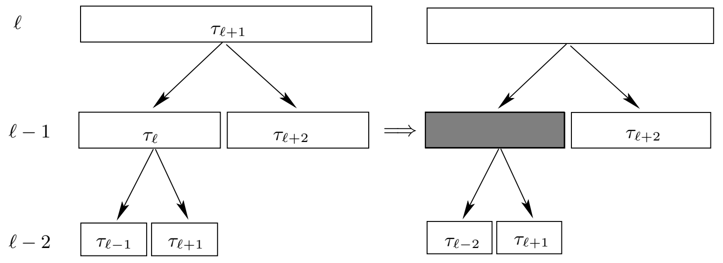

Fig. 10. Phase 1 of node u_ℓ starts from $\text{Density}(u_{\ell-2}) = \tau_{\ell-1}$ (left) and ends at $\text{Density}(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

- Phase 2 of node u_ℓ starts with node $u_{\ell-2}$ having density τ_ℓ and ends with node $u_{\ell-2}$ having density $\tau_{\ell-2}$. When node $u_{\ell-1}$ does its second sweep (see Fig. 11), which occurs after $(\tau_{\ell-2} - \tau_\ell)\text{Cap}(u_{\ell-2})$ inserts, the density of node $u_{\ell-1}$ is $(\tau_{\ell-2} + \tau_{\ell-1})/2 > \tau_{\ell-1}$, so node $u_{\ell-1}$ is above its threshold. Thus, the end of Phase 2 is the first rebalance of node u_ℓ.

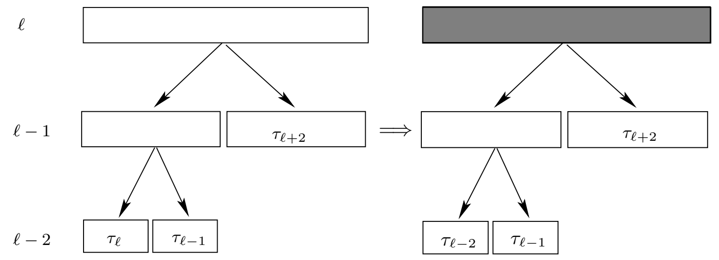

Fig. 11. Phase 2 of node u_ℓ starts from $\text{Density}(u_{\ell-2}) = \tau_\ell$ (left) and ends at $\text{Density}(u_{\ell-2}) = \tau_{\ell-2}$ (right). The shaded region is rebalanced.

Thus, we have recurrence $\mathcal{N}_\kappa(\ell,1) \le \mathcal{N}_\kappa(\ell-1,1) + \mathcal{N}_\kappa(\ell-1,2)$. However, we cannot use the straightforward bound $\mathcal{N}_\kappa(\ell,1) \le 2\mathcal{N}_\kappa(\ell-1,1)$ as we did in Lemma 4. When we try to use this bound, we obtain the solution $\mathcal{N}_\kappa(\ell,1) \le 2^{\ell-\kappa+1}$. Thus, we obtain an amortized number of moves

$$\begin{aligned}\frac{1}{N^\alpha}\sum_{\kappa=0}^{h'} \mathcal{N}_\kappa(h'+1,1)2^\kappa \log N &\le \frac{1}{N^\alpha}\sum_{\kappa=0}^{h'} 2^{h'-\kappa+2}2^\kappa \log N\\ &= O(\log^2 N),\end{aligned}$$

which is greater than our goal of $O(\log N)$. Instead, we need a tighter analysis.

Now we analyze $\mathcal{N}_\kappa(\ell-1,2)$ in more detail. The bound $\mathcal{N}_\kappa(\ell-1,2)$ is the number of sweeps of node u_κ at level κ between the first and second sweeps of $u_{\ell-1}$. After the first rebalance of $u_{\ell-1}$, we have Density$(v_{\ell-2}) = \tau_{\ell-1}$ and Density$(v_{\ell-3}) = \tau_{\ell-2}$ according to our rebalance strategy for the sequential-insert pattern, i.e., both $v_{\ell-2}$ and $v_{\ell-3}$ already have densities as high as their parents' upper thresholds (see Fig. 12). The time when $u_{\ell-1}$ does its next sweep is the time when $u_{\ell-2}$ reaches its threshold. Because $v_{\ell-3}$ has density $\tau_{\ell-2}$, this is also the first time when $u_{\ell-2}$ does its next sweep, and because $v_{\ell-4}$ has density $\tau_{\ell-3}$, this is also the first time when $u_{\ell-3}$ does its next sweep, i.e., both $\mathcal{N}_{\ell-2}(\ell-1,2)$ and $\mathcal{N}_{\ell-3}(\ell-1,2)$ are 1. This process continues a number of levels down the tree to be determined below, but not to the leaves.

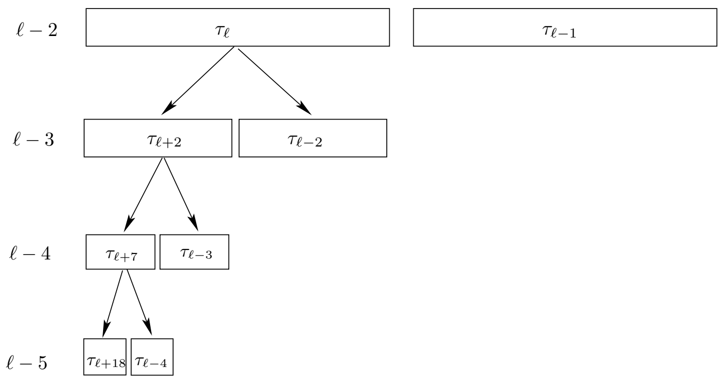

Fig. 12. The densities of node u_ℓ's descendants at the beginning of Phase 2 of node u_ℓ.

The process does not continue to the leaves because after the first rebalance of $u_{\ell-1}$, the density of each leftmost child is decreasing from top to bottom. Thus, at some level $\ell-j$, node $u_{\ell-j}$ may be so sparse that there are not enough elements to fill its right child $v_{\ell-j-1}$ to density $\tau_{\ell-j}$. Specifically, we claim that as long as Density$(u_{\ell-j}) \geq \tau_h$, then we can fill $v_{\ell-j-1}$ to density $\tau_{\ell-j}$. Because

$$\begin{aligned} \text{Density}(u_{\ell-j}) &\geq \tau_h \\ &\geq (\tau_h + \rho_h)/2 \\ &= (\tau_{\ell-j} + \rho_{\ell-j})/2 \end{aligned}$$

by (2) and (3), we can fill $v_{\ell-j-1}$ to density $\tau_{\ell-j}$ while keeping the density of $u_{\ell-j-1}$ great than $\rho_{\ell-j}$. Thus, there is only one sweep of $u_{\ell-j}$ in Phase 2 of u_ℓ, i.e., $\mathcal{N}_{\ell-j}(\ell-1,2) = 1$.

We now calculate the lowest level x such that Density$(u_x) \geq \tau_h$. First, we give the densities of the nodes above level x after the first rebalance of $u_{\ell-1}$.

CLAIM 10. *For level $\ell - j > x$,*

$$\text{Density}(u_{\ell-j}) = \tau_{\ell+3\cdot 2^{j-2}-j-1}. \tag{17}$$

The proof of this claim is by induction on j. The base case is $j = 2$. Eq. (17) is satisfied because Density$(u_{\ell-2}) = \tau_\ell$. Now assume that the claim is true for level $\ell - j$ and all levels above. We show that the claim is also true for level $\ell - j - 1$. Because $\ell - j > x$,

$\text{Density}(u_{\ell-j}) \geq (\tau_{\ell-j}+\rho_{\ell-j})/2$. Thus, we can fill $v_{\ell-j-1}$ to density $\tau_{\ell-j}$. Thus, we obtain

$$\begin{aligned}
\text{Density}(u_{\ell-j-1}) &= 2\text{Density}(u_{\ell-j}) - \text{Density}(v_{\ell-j-1}) \\
&= 2\tau_{\ell+3\cdot 2^{j-2}-j-1} - \tau_{\ell-j} \\
&= \tau_{\ell+3\cdot 2^{j-1}-j-2}.
\end{aligned}$$

So (17) is true for level $\ell-j-1$.

Now we need solve the inequality

$$\tau_{\ell+3\cdot 2^{j-2}-j-1} \geq \tau_h \tag{18}$$

to determine x. Ineq. (18) is equivalent to

$$3\cdot 2^{j-2} - j - 1 \leq h - \ell.$$

Because $\ell \leq h' = \alpha \log N$ for some fixed constant α, $j = \lg\lg N - O(1)$. That is, the lowest level that x can be is $\ell - \lg\lg N + \lambda_\alpha$, where λ_α is a constant that depends only on α. Thus, we have formula

$$\mathcal{N}_\kappa(\ell-1,2) = 1 \tag{19}$$

for $\ell - 1 \geq \kappa \geq \ell - \lg\lg N + \lambda_\alpha$.

For those levels lower than $\ell - \lg\lg N + \lambda_\alpha$, we use simple but straightforward bounds: each sweep of a node costs at most two sweeps of its left child, assuming that each node is within balance. Thus, we have formula

$$\mathcal{N}_\kappa(\ell-1,2) \leq 2^{\ell-\lg\lg N+\lambda_\alpha-\kappa}, \tag{20}$$

for $0 \leq \kappa \leq \ell - \lg\lg N + \lambda_\alpha$.

Combining (19) and (20), we obtain

$$\mathcal{N}_\kappa(\ell-1,2) \leq \lceil 2^{\ell-\lg\lg N+\lambda_\alpha-\kappa} \rceil. \tag{21}$$

Now we are ready to bound $\mathcal{N}_\kappa(\ell,1)$ by using (21):

$$\begin{aligned}
\mathcal{N}_\kappa(\ell,1) &\leq \mathcal{N}_\kappa(\ell-1,1) + \mathcal{N}_\kappa(\ell-1,2) \\
&\leq \mathcal{N}_\kappa(\kappa,1) + \sum_{i=\kappa}^{\ell-1} \mathcal{N}_\kappa(i,2) \\
&= 1 + \sum_{i=\kappa}^{\ell-1} \mathcal{N}_\kappa(i,2) \\
&\leq 1 + \sum_{i=\kappa}^{\ell-1} \lceil 2^{i-\lg\lg N+\lambda_\alpha-\kappa} \rceil \\
&\leq \ell - \kappa + 1 + \sum_{i=\kappa+\lg\lg N-\lambda_\alpha}^{\ell-1} 2^{i-\lg\lg N+\lambda_\alpha-\kappa} \\
&\leq \ell - \kappa + 1 + 2^{\ell-\lg\lg N+\lambda_\alpha-\kappa}.
\end{aligned}$$

Finally, we establish that the amortized number of movements for these N^α elements is at most

$$\begin{aligned}
&\frac{1}{N^\alpha}\sum_{\kappa=0}^{h'} \mathcal{N}_\kappa(h',1)2^\kappa \log N \\
\leq\ & \frac{1}{N^\alpha}\sum_{\kappa=0}^{\lceil \alpha \log N\rceil} (h'-\kappa+1)2^\kappa \log N + \frac{1}{N^\alpha}\sum_{\kappa=0}^{\lceil \alpha \log N\rceil} 2^{h'-\lg\lg N+\lambda_\alpha-\kappa}2^\kappa \log N \\
=\ & O(\log N).
\end{aligned}$$

Now we bound the number of memory transfers. Observe that after any insert, the elements moved from a contiguous group, and the moves can be performed with a constant number of scans. Therefore the amortized number of memory transfer is $O(1+(\log N)/B)$. □

Based on Lemma 9, Theorem 8 is proved as follows.

PROOF OF THEOREM 8: We consider each bottom subtree T. Suppose that an ancestor of the root of T does a rebalance. Then the root of T has density at most $\tau_{h'+1}$. Thus, we can insert at least $(\tau_{h'}-\tau_{h'+1})\Theta(N^\alpha \log N) = \Theta(N^\alpha)$ elements without triggering sweeps above level h', i.e., inserting N^α elements in T triggers at most one rebalance in top subtree A.

Now we consider a ***round*** of N^α inserts into some bottom subtree T. We show that there are $O(\log N)$ amortized element moves in the APMA. Recall that we use the predictor to store recent inserts. For the first N^α inserts, the predictor only uses one cell. When the next N^α inserts start to hammer, the predictor uses the second cell to store new elements. After the count number in the second cell reaches $\log N$, which means there are $\log N$ new elements at the second position, the count number in the first cell begins to decrease. Thus, at most $2\log N$ inserts remove the first cell, meaning that the hammer-insert pattern starts after the first $2\log N$ inserts. Thus, we divide the N^α inserts in the round into two parts: the first $2\log N$ ones and the $N^\alpha - 2\log N$ subsequent ones. This is one dividing point.

The second dividing point is when some insert triggers a rebalance in the top subtree A. We assume the second dividing point is after the first one. The alternative is similar to the following analysis, although somewhat easier. These two dividing points split the round into three parts. We analyze the cost of the rebalance in the bottom subtree T for these parts as follows:

1. The rebalance cost for the first part, the insertion of the first $2\log N$ elements, is at most $3N^\alpha \log N$. To see why, observe that there exists a node u' of size N^α, such that these $2\log N$ elements trigger at most one rebalance above u', by an argument similar to that above. This rebalance is within T, and therefore costs at most $N^\alpha \log N$. Thus, the total cost is the cost of this rebalance, at most $N^\alpha \log N$, plus the cost of the rebalances below u', at most $(2\log N - 1)N^\alpha$.
2. The second part is from the $(2\log N)$th element insert to the element insert triggering the rebalance in the top subtree A. The total cost is at most the worst-case cost in Lemma 9, which is $O(N^\alpha \log N)$.
3. The third part is from the element insert triggering the rebalance in the top subtree A to the last element insert of these N^α elements. From Lemma 9, the cost is less than the cost to insert all N^α elements in subtree T whose ancestor did the rebalance, which is $O(N^\alpha \log N)$.

Thus, without counting the rebalance cost in the top subtree A, the average cost for each round is $O(N^{\alpha}\log N)/N^{\alpha} = O(\log N)$. If we can show that the average cost in the top subtree A is also $\log N$, then the theorem is proved.

From the view point of top subtree A, the bulk insert is similar to random inserts of "big elements" of size N^{α} in A, because big element triggers at most one rebalance in A and a leaf node of size $N^{\alpha}\log N$ is a black box that has $O(\log N)$ amortized moves. So the bulk insert is: randomly choose a leaf node in A, a black-box operation to insert N^{α} elements in the leaf node, each with $O(\log N)$ moves. If the leaf node reaches its threshold, then a rebalance is triggered at most once in A. Thus, as in Theorem 7, we have $O(\log N)$ element moves in the top subtree A. As before, the memory-transfer bound follows because all rebalances are to contiguous groups of elements. □

5. EXPERIMENTAL RESULTS

In this section we describe our simulation and experimental study. We show that our results are consistent with the asymptotic bounds from the previous sections and suggest the constants involved. We also demonstrate that the bookkeeping for the adaptive structure has little computational overhead.

We ran our experiments as follows: For each insert pattern, we began with an empty array and added elements until the array contained roughly 1.4 million elements. We began our measurements once the array had size at least $100,000$. We recorded the amortized number of element moves per insert as well as the running times. We considered the sequential, hammer, random, and bulk insertion distributions from the previous sections. We also added noise to the distributions, combining, for example, the hammer and random distributions, showing that the predictor is resilient to this noise. Each graph plots the intermediate data points in a single run.

We ran our experiments on a Pentium 4 CPU 3.0GHZ, with 2GB of RAM, running Windows XP professional, and a 100G ATA disk drive. Our file contained up to 2^{21} keys, and the total memory used was up to 1.4 GB. We implemented a search into the PMA as a simple binary search. The binary search was appropriate since our experiments were small enough that they did not involve paging to disk. Consequently, the search time was dominated by the insertion time into the PMA.

The adaptive PMA is ultimately targeted for used in cache-oblivious and locality preserving B-trees, where the search time becomes relatively more expensive because the data structures do not fit in main memory. In this case the binary search will be too slow because it lacks sufficient data locality. (The number of memory transfers for the PMA insert is $O(1+(\log N)/B)$, which is dominated by the cost of a binary search, $O(\log\lceil N/B\rceil)$, as well as the optimal external-memory search cost, $O(1+\log_B N)$.) Thus, our next round of experiments on larger data sets is to be run with the objective of speeding up inserts in the cache-oblivious B-tree.

Sequential inserts. We first compared the adaptive and traditional PMAs on sequential insertions. For sequential inserts of roughly 1.4 million elements, the APMA has four times fewer element moves per insertion than the traditional PMA and running times that are nearly seven times faster.

Fig. 13 shows the average number of element moves in the PMAs. The x-axis indicates the number of inserted elements up to 1.4 million. The y-axis indicates the number of element moves divided by $\lg N$. For both the adaptive and traditional PMA, we choose

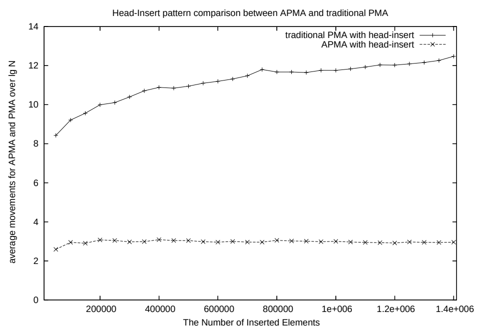

Fig. 13. Sequential inserts: average moves per insert divided by $\lg N$. The array size grows to two million and 1.4 million elements are inserted.

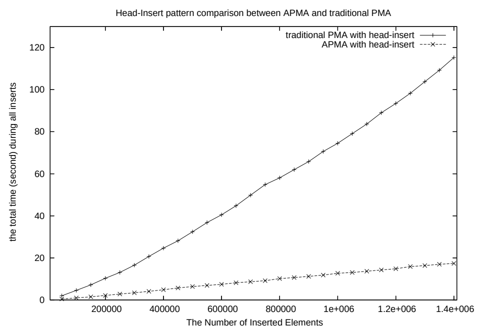

Fig. 14. Sequential inserts: the running time to insert up to 1.4 million elements.

the upper and lower density thresholds as follows: $\tau_0 = 0.92$, $\tau_h = 0.7$, $\rho_h = 0.3$, and $\rho_0 = 0.08$. In our experiments, we double when the array gets too full. Thus, before doubling, the array has density over 0.7 and after, the array has density over 0.35. (By increasing the array size by only a $(1+\varepsilon)$-factor for constant ε, we can make the density of the entire array at least $(1+\varepsilon)\rho_h$ with only a small additive increase in the number of elements moved. Thus, we can have an array whose density is always arbitrary close to 70% full.) The roughly flat line shows the performance of the APMA. These experiments suggest that the constant in front of the $\lg N$ (see Theorem 3) is roughly 2.5 for the density thresholds chosen. Because we are measuring number of element moves, these results are machine independent. Fig. 14 gives the running times for our experiment. Observe that the APMA runs almost 7 times faster even though the amortized number of element moves is only 4 times smaller. Hence, the overhead for the adaptive PMA is small. We suspect that this decrease has to do with caching issues; the APMA has a smaller working set than the traditional PMA.

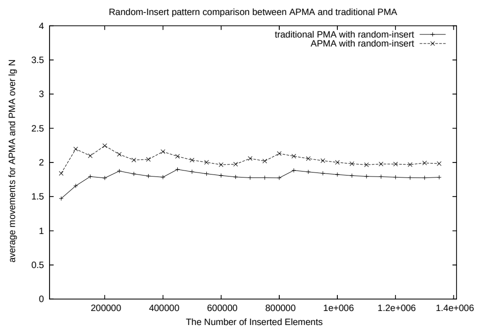

Fig. 15. Random inserts: average moves per insert divided by $\lg N$. The array size grows to two million and 1.4 million elements are inserted.

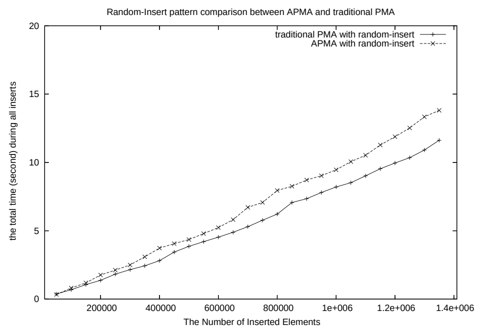

Fig. 16. Random inserts: the running time to insert up to 1.4 million elements.

Random inserts. For random insertions the traditional PMA performs slightly better than the APMA because there is seemingly no advantage in uneven rebalalances and because the traditional PMA has less overhead. For random insertions of 1.4 million elements with the same density thresholds and axes as in Figures 13 and 14, both the adaptive and traditional PMAs have the same asymptotic performance (see Theorem 7). The traditional PMA's constant seem to be less than 10% smaller. Figures 15 and 16 show that both the amortized number of element moves and the running times are comparable, with the traditional PMA performing slightly better, as expected. Fig. 16 indicates that the bookkeeping overhead for the APMA is small.

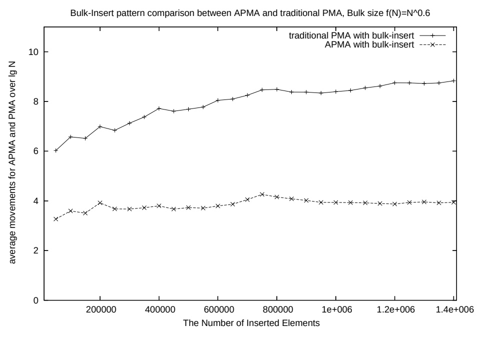

Fig. 17. Bulk inserts: average moves per insert divided by $\lg N$. The array size grows to two million and 1.4 million elements are inserted.

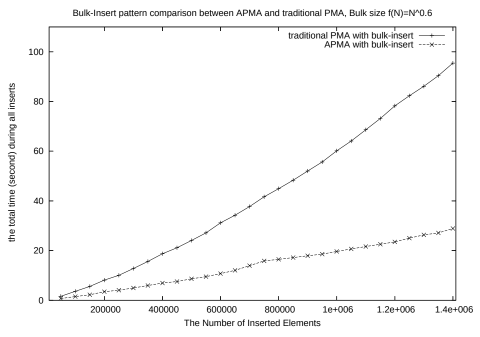

Fig. 18. Bulk inserts: the running time to insert up to 1.4 million elements.

Bulk inserts. We next investigated the bulk-insert distribution, comparing both the adaptive and traditional PMAs. For bulk insertions of 1.4 million elements, the APMA has roughly 2.3 times fewer element moves per insertion than the traditional PMA and running times that are over 3.4 times faster. Fig. 17 shows the average number of elements moves in the PMAs with the same thresholds as in Fig. 13 and bulk parameter $N^{0.6}$. The roughly flat line shows the performance of the APMA. These experiments suggest that the constant in front of the $\lg N$ (see Theorem 8) is roughly 4 for the chosen density thresholds and bulk parameter. Fig. 18 shows the running times of the traditional and adaptive PMAs.

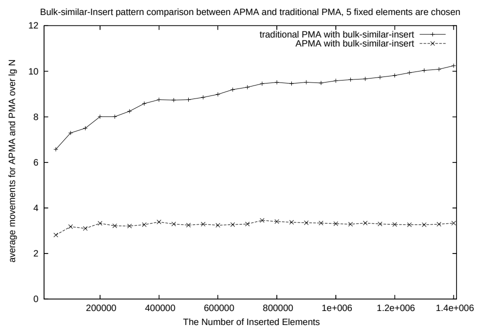

Fig. 19. Multiple sequential inserts: average moves per insert divided by $\lg N$. The array size grows to two million and 1.4 million elements are inserted.

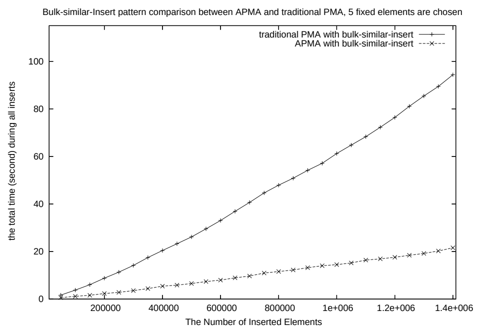

Fig. 20. Multiple sequential inserts: the running time to insert up to 1.4 million elements.

Multiple sequential inserts. We next consider a distribution that performs sequential inserts into multiple parts of the array at once. We first choose R random elements and then insert one element at a time after one of these chosen elements. As long as the number of chosen elements R is less than the number of elements stored in the predictor, most predictions are good and the performance of APMA remains $O(\log N)$. Figures 19 and 20 compare the performance of the traditional and adaptive PMAs when we choose 5 fixed elements. The APMA in this case has a performance only slightly worse than that in the sequential-insert case while tradition PMA still performs much worse.

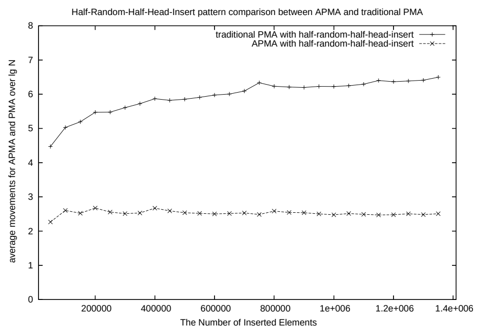

Fig. 21. Half random, half sequential inserts: average moves per insert divided by $\lg N$. The array size grows to two million and 1.4 million elements are inserted.

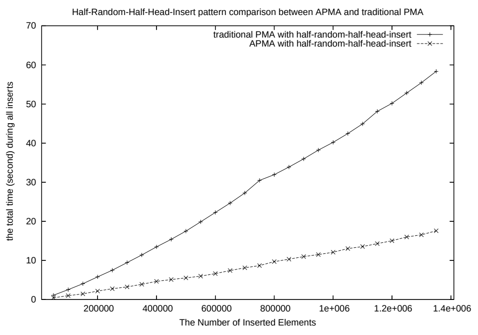

Fig. 22. Half random, half sequential inserts: the running time to insert up to 1.4 million elements.

Half random and half sequential inserts. Finally, we analyze a distribution that adds noise to sequential inserts. We decide randomly whether to insert a new element at the front of the PMA or after a random element. Thus, roughly half of the inserted elements form random noise. Figures 21 and 22 compare the performance of the traditional PMA and APMA. The roughly flat curve in Fig. 21 is the performance of APMA, which is slightly worse than that in random inserts and better than that in sequential inserts, while the performance of traditional PMA is about 3 times worse than that of random inserts.

6. CONCLUSION

We introduced an adaptive packed-memory array. The adaptive PMA guarantees a performance at least as good as that of the traditional PMA, while simultaneously adapting to common insertion distributions. Thus, the adaptive PMA always achieves at most $O(\log^2 N)$ amortized element moves and $O(1+(\log^2 N)/B)$ memory transfers per update, but it achieves only $O(\log N)$ amortized element moves and $O(1+(\log N)/B)$ memory transfers for sequential inserts, hammer inserts, random inserts, and bulk inserts. Our simulations and experiments are consistent with these asymptotic bounds. Several open problems remain. For example, can we show some type of working-set property for an adaptive PMA? Perhaps such an investigation will require study into the design of other predictors. The next step in this research is to use the adaptive PMA in a cache-oblivious B-tree and to measure the speedup obtained for updates.

ACKNOWLEDGMENTS

We are grateful to Ziyang Duan for proposing an early uneven-rebalance strategy. We are grateful to Bradley Kuszmaul for suggesting that we focus on sequential inserts. We thank Yue Wang for important discussions and for reading early drafts of this paper. Finally, we thank our anonymous referees for many important comments that significantly improved our presentation.

REFERENCES

BENDER, M. A., COLE, R., DEMAINE, E. D., AND FARACH-COLTON, M. 2002. Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In *Proc. 10th Annual European Symposium on Algorithms (ESA)*. Lecture Notes in Computer Science, vol. 2461. 139–151.

BENDER, M. A., DEMAINE, E. D., AND FARACH-COLTON, M. 2000. Cache-oblivious B-trees. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS)*. 399–409.

BENDER, M. A., DEMAINE, E. D., AND FARACH-COLTON, M. 2005. Cache-oblivious B-trees. *SIAM Journal on Computing 35,* 2, 341–358.

BENDER, M. A., DUAN, Z., IACONO, J., AND WU, J. 2002. A locality-preserving cache-oblivious dynamic dictionary. In *Proc. of the 13th Annual Symposium on Discrete Mathematics (SODA)*. 29–38.

BENDER, M. A., DUAN, Z., IACONO, J., AND WU, J. 2004. A locality-preserving cache-oblivious dynamic dictionary. *Journal of Algorithms 3,* 2, 115–136.

BENDER, M. A., FARACH-COLTON, M., AND KUSZMAUL, B. 2006. Cache-oblivious string B-trees. In *Proc. 25th Symposium on Principles of Database Systems (PODS)*. 233–242.

BENDER, M. A., FARACH-COLTON, M., AND MOSTEIRO, M. 2004. Insertion sort is $O(n \log n)$. In *Proc. 3rd International Conference on Fun with Algorithms (FUN)*. 16–23.

BENDER, M. A., FINEMAN, J. T., GILBERT, S., AND KUSZMAUL, B. C. 2005. Concurrent cache-oblivious B-trees. In *Proc. 17th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 228–237.

BRODAL, G. S., FAGERBERG, R., AND JACOB, R. 2002. Cache oblivious search trees via binary trees of small height. In *Proc. 13th Annual Symposium on Discrete Algorithms (SODA)*. 39–48.

DIETZ, P. F. 1982. Maintaining order in a linked list. In *Proc. Symposium on the Theory of Computing (STOC)*. 122–127.

DIETZ, P. F., SEIFERAS, J. I., AND ZHANG, J. 1994. A tight lower bound for on-line monotonic list labeling. In *Proc. 4th Scandinavian Workshop on Algorithm Theory (SWAT)*. Lecture Notes in Computer Science, vol. 824. 131–142.

DIETZ, P. F. AND SLEATOR, D. D. 1987. Two algorithms for maintaining order in a list. In *Proc. 19th Annual Symposium on Theory of Computing (STOC)*. 365–372.

DIETZ, P. F. AND ZHANG, J. 1990. Lower bounds for monotonic list labeling. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*. Lecture Notes in Computer Science, vol. 447.

ITAI, A., KONHEIM, A. G., AND RODEH, M. 1981. A sparse table implementation of priority queues. In *Proc. 8th Internationl Colloquium on Automata, Languages, and Programming (ICALP)*. Lecture Notes in Computer Science, vol. 115. 417–431.

KATRIEL, I. 2002. Implicit data structures based on local reorganizations. M.S. thesis, Technion – Israel Inst. of Tech., Haifa.

PROKOP, H. 1999. Cache-oblivious algorithms. M.S. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

RAMAN, V. 1999. Locality-preserving dictionaries: theory and application to clustering in databases. In *Proc. 18th Symposium on Principles of Database Systems (PODS)*. 337–345.

TSAKALIDIS, A. K. 1984. Maintaining order in a generalized linked list. *Acta Informatica 21,* 1, 101–112.

WILLARD, D. 1982. Maintaining dense sequential files in a dynamic environment (extended abstract). In *Proc. 14th Annual Symposium on Theory of Computing (STOC)*. 114–121.

WILLARD, D. E. 1986. Good worst-case algorithms for inserting and deleting records in dense sequential files. In *Proc. International Conference on Management of Data (SIGMOD)*. 251–260.

WILLARD, D. E. 1992. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Information and Computation 97,* 2, 150–204.

APPENDIX

PROOF OF CLAIM 6: We give more details of what happens during the tail-insert stage. During the tail-insert stage, new elements are inserted after x, the rightmost element of u_i at the beginning of the stage. At the end of the stage, node u_i reaches its upper threshold, which triggers the next sweep of node u_{i+1}. Observe that sweeps occurring during the tail-insert stage do not involve v_i, u_i's sibling. This is because the tail-insert stage ends when u_i reaches its upper threshold, which triggers the sweep of u_{i+1}. We will bound the number of sweeps of u_κ during the tail-insert stage of u_i.

Below, we show that it suffices to prove Claim 6 when the tail-insert stage begins with $\text{Density}(u_i) = \rho_{i+1}$. To do so, we show that the fewer elements there are in u_i at the start of the tail-insert stage, the more sweeps there will be of u_κ (descendant of u_i) during the stage. That is, the number of sweeps of u_κ is maximized when node u_i starts with density ρ_{i+1}, the lowest density possible after a sweep of u_{i+1}.

We now explain why the worst case is when $\text{Density}(u_i) = \rho_{i+1}$. Recall that x is in u_κ, and since all inserts are after x, they are all in u_κ. If node u_i has a low density at the beginning of the stage, then more elements can be inserted after x and into u_κ without triggering a sweep of u_{i+1}, which means that there are more sweeps of u_κ during the stage.

We present additional notation. We define $\mathcal{C}_\kappa(i,t)$ to be the number of sweeps of u_κ between the $(t-1)$th and the tth sweep of u_i since the beginning of the tail-insert stage.[2] We define ***Phase*** t ***of*** u_i to be the phase starting after the $(t-1)$th sweep of u_i and ending at the tth sweep of u_i since the beginning of the tail-insert stage. Thus, by the above two definitions, the number of sweeps of u_κ in the Phase t of u_i equals $\mathcal{C}_\kappa(i,t)$. To simplify the proof, we constrain the density thresholds τ_0, τ_h, ρ_0, and ρ_h as follows:

$$\tau_0 - \tau_h = \rho_h - \rho_0 \quad \text{and} \quad \tau_0 \le 5\rho_0. \tag{22}$$

For example, setting $\rho_0 = 0.16$, $\rho_h = 0.32$, $\tau_h = 0.64$ and $\tau_0 = 0.8$ satisfies (1)-(4) and (22). Therefore, by (2) and (3), for any $0 \le \ell \le h$ we obtain

$$\tau_\ell + \rho_\ell = \tau_0 + \rho_0 = \tau_h + \rho_h \quad \text{and} \quad \tau_\ell \le 5\rho_\ell. \tag{23}$$

Observe that for any choice of constants ρ_0 and τ_0, there exists a constant β, such that $\tau_0 \le \beta\rho_0$. In this proof, we adopt the constraint that $\tau_0 \le 5\rho_0$ for the sake of relative simplicity; we will explain why the results also carry through if we choose some bigger constant instead.

To establish Claim 6, we decompose the sequence of insertions before the first sweep of u_{i+1} since the beginning of the tail-insert stage into phases of u_i, as defined above. By the similar analysis to that of Theorem 3, we show that there are at most three phases of node u_i before the first sweep of u_{i+1}.

Now we prove that there are at most three phases of u_i in the tail-insert stage of u_i; we do so by analyzing the densities in each phase.

I) Consider the densities of child nodes u_{i-1} and v_{i-1} of node u_i at the end of Phase 1 of u_i. The first sweep of u_i occurs, when u_{i-1} reaches its threshold τ_{i-1} (see Fig. 23). After the first sweep of u_i (which is the beginning of Phase 2), we claim that the marker

[2]Thus, $\mathcal{C}_\kappa(i,t)$ is defined analogously to $\mathcal{N}_\kappa(i,t)$, except that we begin counting from the beginning of the tail-insert stage rather than from the first insert into the APMA.

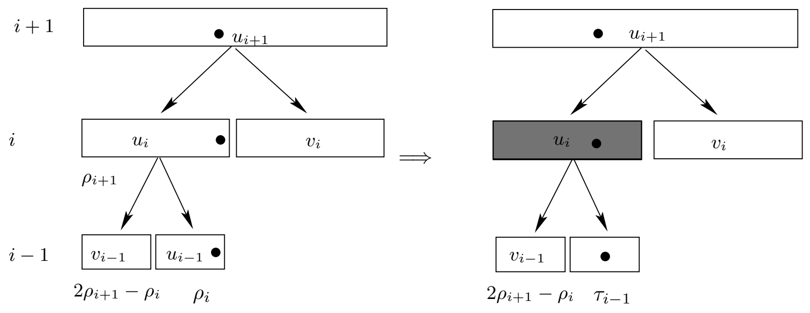

Fig. 23. Phase 1 of u_i starts from Density$(u_{i-1}) = \rho_i$ (left) and ends at Density$(u_{i-1}) = \tau_{i-1}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of the phase is shaded.

element x is either the leftmost element of the right child of u_i or the rightmost element of the left child of u_i.

We now prove this claim. Notice that the number of elements in u_i before x is $2\rho_{i+1}\text{Cap}(u_{i-1})$ and the number of elements in u_i after x is $(\tau_{i-1} - \rho_i)\text{Cap}(u_{i-1})$. To see why, observe that by assumption the phase begins when Density$(u_i) = \rho_{i+1}$. Since all inserts are after x, the number of elements before x stays the same. A rebalance of node u_i is triggered when u_{i-1} reaches its threshold, after $(\tau_{i-1} - \rho_i)\text{Cap}(u_{i-1})$ elements have been inserted.

It is legal for u_{i-1} and v_{i-1} to contain $2\rho_{i+1}\text{Cap}(u_{i-1})$ elements and $(\tau_{i-1} - \rho_i)\text{Cap}(u_{i-1})$ elements, and therefore the sweep at level $i-1$ is constrained by the hammer constraint (not density constraints). Marker element x is always stored in the child having the smaller density (by the hammer constraint). Thus, if there are more elements before x than after x, then x is in the right child of u_i (u_{i-1} is a right child). Otherwise, x is in the left child of u_i (u_{i-1} is a left child).

In the first case, when x is the leftmost element of the right child of u_i, the insert pattern into u_{i-1} in Phase 2 is exactly the head-insert case. Thus, by Theorem 3, the number of sweeps of u_κ in Phase 2 is given by $\mathcal{C}_\kappa(i,2) = O(2^{i-\kappa})$.

In the following we consider the second case, when x is the rightmost element of the left child of u_i. Thus, after the first sweep of u_i, by the hammer constraint, we have the the following densities:

$$\begin{aligned} \text{Density}(u_{i-1}) &= 2\rho_{i+1}, \\ \text{Density}(v_{i-1}) &= \tau_{i-1} - \rho_i. \end{aligned}$$

II) Now (for the above second case) we consider the densities of child nodes u_{i-1} and v_{i-1} of node u_i at the end of Phase 2. The second sweep of u_i occurs when u_{i-1} reaches its upper threshold again (see Fig. 24). Recall that at the beginning of the phase, we chose to put the marker element x in the child of u_i having the smaller density, and since we are in the second case, this was the left child of u_i. Thus, by the hammer constraint,

$$2\rho_{i+1} < \tau_{i-1} - \rho_i\,. \tag{24}$$

When u_{i-1} reaches its threshold, the number of elements after x in u_i is the number of elements in u_{i-1} after x (the new elements inserted in Phase 2) plus the number of

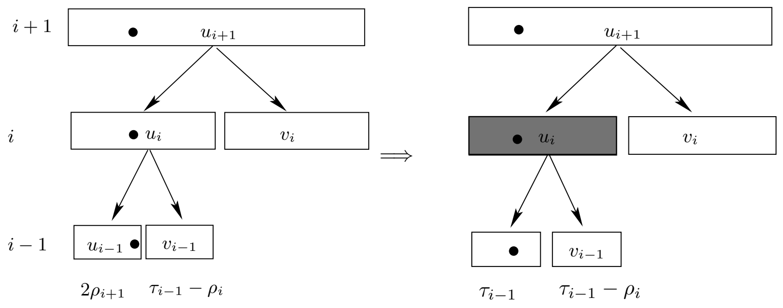

Fig. 24. Phase 2 of u_i starts from $\text{Density}(u_{i-1}) = 2\rho_{i+1}$ (left) and ends at $\text{Density}(u_{i-1}) = \tau_{i-1}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of the phase is shaded.

elements in v_{i-1}, i.e.,

$$(\tau_{i-1} - 2\rho_{i+1})\text{Cap}(u_{i-1}) + (\tau_{i-1} - \rho_i)\text{Cap}(v_{i-1}). \tag{25}$$

Observe that (25) is greater than $\tau_{i-1}\text{Cap}(v_{i-1})$ by (24). Therefore, the second sweep of u_i is constrained by the rebalance property, not the hammer constraint. In particular, after the second sweep of u_i, node v_{i-1} has density τ_i, the upper threshold of its parent u_i; node u_{i-1} has (the remaining) density $\tau_{i-1} + (\tau_{i-1} - \rho_i) - \tau_i$, which equals $\tau_{i-1} - \rho_{i-1}$ by (23). Thus, after the second sweep, we have the following densities:

$$\begin{aligned} \text{Density}(u_{i-1}) &= \tau_{i-1} - \rho_{i-1}, \\ \text{Density}(v_{i-1}) &= \tau_i. \end{aligned}$$

III) We now consider the densities of child nodes u_{i-1} and v_{i-1} of node u_i at the end of Phase 3. (We focus on the above second case in the following, but the first case is now essentially the same.) The third sweep of u_i occurs when u_{i-1} reaches its threshold for a third time (see Fig. 25). When u_i does the third sweep, the density of u_i is $(\tau_{i-1} + \tau_i)/2 > \tau_i$, so u_i is above its upper threshold. Thus, the end of Phase 3 is the first sweep of u_{i+1} since the beginning of the tail-insert stage.

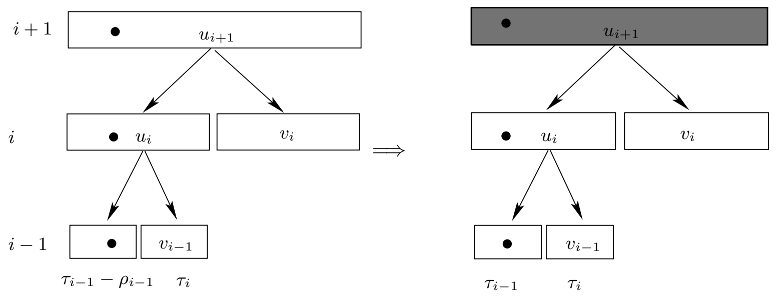

Fig. 25. Phase 3 of u_i starts from $\text{Density}(u_{i-1}) = \tau_{i-1} - \rho_{i-1}$ (left) and ends at $\text{Density}(u_{i-1}) = \tau_{i-1}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of the phase is shaded.

We have therefore shown that (for the second case) there are at most three sweeps of u_i before the first sweep of u_{i+1}, that is,

$$\mathcal{C}_\kappa(i+1,1) \leq \mathcal{C}_\kappa(i,1) + \mathcal{C}_\kappa(i,2) + \mathcal{C}_\kappa(i,3)\,. \tag{26}$$

For the first case, we have the similar recurrence

$$\mathcal{C}_\kappa(i+1,1) \leq \mathcal{C}_\kappa(i,1) + O(2^{i-\kappa}) + \mathcal{C}_\kappa(i,3)\,. \tag{27}$$

As we will show in (28), Recurrence (27) in the first case is actually bounded by Recurrence (26). In the rest of this appendix, we only need focus on (26).

Until now, the proof has been similar to the proof of Theorem 3. However, if we continue to decompose Phase 2, we find that in the worst case there are three subphases. Furthermore, we cannot use the recurrence $\mathcal{C}_\kappa(i,3) \leq \mathcal{C}_\kappa(i,2)$ to prove our bound as in Theorem 3, because the recurrence is true but too weak.

To establish our bound, we instead prove the following recurrences for Phases 2 and 3:

$$\mathcal{C}_\kappa(i,2) \leq \mathcal{C}_\kappa(i-1,1) + \mathcal{C}_\kappa(i-3,1) + O(2^{i-\kappa})\,, \tag{28}$$

and

$$\mathcal{C}_\kappa(i,3) \leq \mathcal{C}_\kappa(i-3,1) + O(2^{i-\kappa})\,. \tag{29}$$

Before we establish Recurrences (28) and (29), we prove the following claim, which describes a subphase in both Phases 2 and 3:

CLAIM 11. *Consider a tail-insert stage of u_{i-2} starting at* Density$(u_{i-2}) = 4\rho_{i+1}$ *and ending when node u_{i-2} reaches its upper threshold. The number of sweeps of u_κ during this stage is at most $\mathcal{C}_\kappa(i-3,1)$.*

PROOF OF CLAIM 11: We first give the densities of nodes u_{i-3}, u_{i-4}, v_{i-3}, and v_{i-4} at the

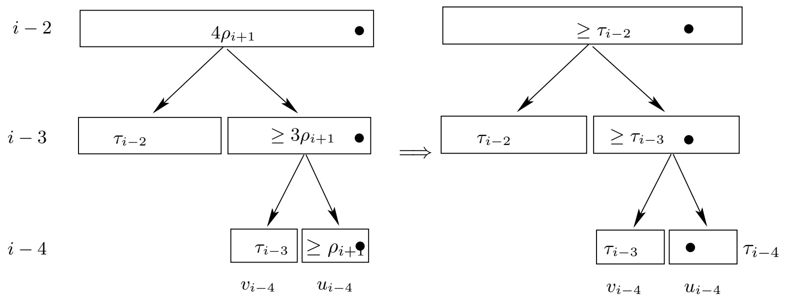

Fig. 26. The tail-insert stage of u_{i-2} starts from Density$(u_{i-4}) \geq \rho_{i+1}$ (left) and ends at Density$(u_{i-4}) = \tau_{i-4}$ (right). The marker element x is indicated by a black dot. At the end of the tail-insert stage of u_{i-2}, node u_{i-1} is rebalanced.

beginning of the tail-insert stage of u_{i-2}. We show that the rebalance is constrained by the upper density thresholds of v_{i-3} and v_{i-4}, that is, at the beginning of the tail-insert stage, Density$(v_{i-3}) = \tau_{i-2}$ and Density$(v_{i-4}) = \tau_{i-3}$.

The tail-insert stage of u_{i-2} begins after a sweep of u_{i-2}, and therefore by the rebalance property

$$\text{Density}(v_{i-3}) \leq \tau_{i-2} \quad \text{and} \quad \text{Density}(v_{i-4}) \leq \tau_{i-3}\,.$$

From (23), we obtain

$$\text{Density}(v_{i-3}) \le 5\rho_{i-2} \quad \text{and} \quad \text{Density}(v_{i-4}) \le 5\rho_{i-3}\,.$$

From (1), we obtain

$$\text{Density}(v_{i-3}) \le 5\rho_{i+1} \quad \text{and} \quad \text{Density}(v_{i-4}) \le 5\rho_{i+1}\,. \tag{30}$$

Now we bound the densities of u_{i-3} and u_{i-4}. The number of elements in u_{i-3} is the number of elements in u_{i-2} minus the number of elements in v_{i-3} (and similarly for u_{i-4}), that is,

$$\text{Density}(u_{i-3}) \;=\; 2\text{Density}(u_{i-2}) - \text{Density}(v_{i-3})\,, \tag{31}$$

$$\text{Density}(u_{i-4}) \;=\; 2\text{Density}(u_{i-3}) - \text{Density}(v_{i-4})\,. \tag{32}$$

From (30), we obtain

$$\text{Density}(u_{i-3}) \ge 8\rho_{i+1} - 5\rho_{i+1} = 3\rho_{i+1}\,. \tag{33}$$

Now from (30) and (33),

$$\text{Density}(u_{i-4}) \ge 6\rho_{i+1} - 5\rho_{i+1} = \rho_{i+1}\,. \tag{34}$$

Inequalities (33) and (34) show that at the beginning of the stage, the densities of u_{i-3} and u_{i-4} are above the lower bound thresholds ρ_{i-2} and ρ_{i-3}, respectively, which means that v_{i-3} and v_{i-4} are at their parents' upper thresholds, i.e., $\text{Density}(v_{i-3}) = \tau_{i-2}$ and $\text{Density}(v_{i-4}) = \tau_{i-3}$.

We now explain that when node u_{i-4} reaches its upper threshold, then u_{i-2} also reaches its upper threshold (see Fig. 26). This is because when $\text{Density}(u_{i-4}) = \tau_{i-4}$, we already have $\text{Density}(v_{i-4}) = \tau_{i-3}$. Therefore, u_{i-3} is above its upper threshold. We already have $\text{Density}(v_{i-3}) = \tau_{i-2}$, and therefore u_{i-2} is also above its upper threshold.

Therefore, the number of sweeps of u_κ in the tail-insert stage of u_{i-2} is equal to the number of sweeps of u_κ in the tail-insert stage of u_{i-4} (since u_{i-4} is the rightmost grandchild of u_{i-2}; see Fig. 26). By the definition of the tail-insert stage, the number of sweeps of u_κ in the tail-insert stage of u_{i-4} (which starts with $\text{Density}(u_{i-4}) \ge \rho_{i+1}$) is less than $\mathcal{C}_\kappa(i-3,1)$ (the number of sweeps of u_κ in the tail-insert stage of u_{i-4} that starts with $\text{Density}(u_{i-4}) = \rho_{i-3}$). $\square$

Now we are ready to prove (29). To do so, we give the densities of the sibling nodes u_{i-2} and v_{i-2} at the beginning of Phase 3. Recall that Phase 3 starts with node u_{i-1} having density $\tau_{i-1} - \rho_{i-1}$, v_{i-1} having density τ_i, and the marker element x residing in u_{i-1}. Since the number of elements before x does not change, node u_{i-1} thus has $2\rho_{i+1}\text{Cap}(u_{i-1})$ elements before x and $(\tau_{i-1} - \rho_{i-1} - 2\rho_{i+1})\text{Cap}(u_{i-1})$ elements (the remaining elements) after x.

We now show that the number of elements after x is smaller than the number of elements before x in node u_{i-1}. Because $\tau_{i-1} \le 5\rho_{i-1}$ by (23), we obtain

$$\tau_{i-1} - \rho_{i-1} - 2\rho_{i+1} \le 4\rho_{i-1} - 2\rho_{i+1}.$$

From $\rho_{i-1} < \rho_{i+1}$ by (1), we have

$$\tau_{i-1} - \rho_{i-1} - 2\rho_{i+1} \le 2\rho_{i+1}\,. \tag{35}$$

Equation (35) says that the number of elements after x is smaller than the number of elements before x. Thus, the marker element x resides in the right child of u_{i-1}, which is u_{i-2}.

We now break Phase 3 of u_i into subphases and bound the number of sweeps of u_κ in the subphases. Subphase t of Phase 3 of u_i is the period between the $(t-1)$th and tth sweeps of u_{i-1}.

Now there are two cases. Case A is that node v_{i-2} has density τ_{i-1}, i.e., this level is constrained by the rebalance property. Then we only have one subphase in Phase 3 of u_i because when u_{i-2} reaches its upper threshold τ_{i-2}, then its parent u_{i-1} has density $(\tau_{i-1}+\tau_{i-2})/2 > \tau_{i-1}$, which means the end of Phase 3.

In the following, we consider Case B when the sweep at level $i-2$ is constrained by the hammer constraint. In Case B, we decompose Phase 3 into two subphases as follows:

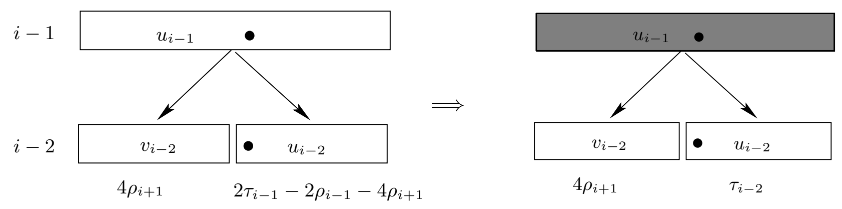

Fig. 27. Subphase 1 of Phase 3 starts from Density$(u_{i-2}) = 2\tau_{i-1}-2\rho_{i-1}-4\rho_{i+1}$ (left) and ends at Density$(u_{i-2}) = \tau_{i-2}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of Subphase 1 is shaded.

- We consider the densities of u_{i-2} and v_{i-2} at the beginning and end of Subphase 1 (see Fig. 27). At the beginning of Subphase 1, because of the hammer constraint, the density of the left child v_{i-2} is $4\rho_{i+1}$ (since the number of elements before x is always $\rho_{i+1}\mathrm{Cap}(u_i)$ – see the beginning of the appendix) and the density of the right child u_{i-2} is $2\tau_{i-1}-2\rho_{i-1}-4\rho_{i+1}$ (the remaining elements in node u_{i-1}). At the end of Subphase 1, node u_{i-2} reaches its upper threshold τ_{i-2}.

  Notice that during Subphase 1, the marker element x is the first element in node u_{i-2} and thus within u_{i-2} we have the head-insert case. Therefore, by Theorem 3, there are $O(2^{i-2-\kappa})$ sweeps of u_κ in Subphase 1.
- We now consider the densities of u_{i-2} and v_{i-2} at the beginning and end of Subphase 2 (see Fig. 28). The beginning of Subphase 2 is right after the sweep of node u_{i-1}. By the rebalance property, the density of the right child at the beginning of Subphase 2 is τ_{i-1} because before the sweep its density was τ_{i-2} ($> \tau_{i-1}$). After the sweep of node u_{i-1}, the marker element x moves to the left child of u_{i-1}. Therefore, the left child becomes node u_{i-2} and Density$(u_{i-2}) = 4\rho_{i+1} + (\tau_{i-2}-\tau_{i-1})$.

  Subphase 2 ends when node u_{i-2} reaches its upper threshold τ_{i-2}. Because the density of v_{i-2} is already at parent u_{i-1}'s threshold τ_{i-1}, the end of Subphase 2 is the end of Phase 3.

  We now prove that the number of sweeps of u_κ in Subphase 2 is less than $\mathcal{C}_\kappa(i-3,1)$, the number of sweeps from Claim 11. Both Subphase 2 and the tail-insert stage of u_{i-2} in Claim 11 end when node u_{i-2} reaches its threshold τ_{i-2}.

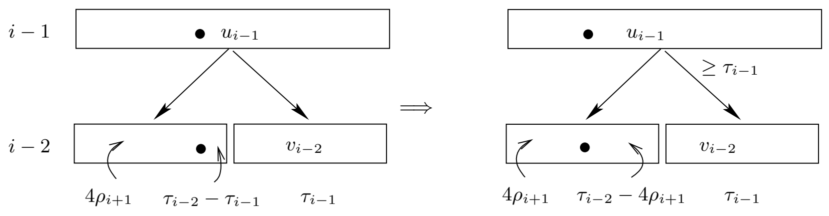

Fig. 28. Subphase 2 of Phase 3 starts from Density$(u_{i-2}) = 4\rho_{i+1}+\tau_{i-2}-\tau_{i-1}$ (left) and ends at Density$(u_{i-2}) = \tau_{i-2}$ (right). The marker element x is indicated by a black dot. At the end of Subphase 2, node u_i, the parent of u_{i-1}, is rebalanced.

However, Subphase 2 starts with more elements after the marker element x than does the tail-insert stage of u_{i-2} and the same number of elements before the marker element x. In particular, Subphase 2 has $4\rho_{i+1}\mathrm{Cap}(u_{i-2})$ elements before and $(\tau_{i-2}-\tau_{i-1})\mathrm{Cap}(u_{i-2})$ elements after x. In contrast, the tail-insert stage of u_{i-2} has no elements after and $4\rho_{i+1}\mathrm{Cap}(u_{i-2})$ elements before x.
Thus, the number of sweeps of u_κ in Subphase 2 is at most the number of sweeps of u_κ in the tail-insert stage of u_{i-2} because fewer elements can be inserted into u_{i-2} before u_{i-2}'s upper threshold is reached.

In summary, there are at most two subphases in Phase 3 and the number of sweeps of u_κ in these two subphases is at most $\mathcal{C}_\kappa(i-3,1)$ plus $O(2^{i-2-\kappa})$, which establishes (29).

We now prove (28). To do so, we decompose Phase 2 of u_i into three subphases, and we analyze the densities of u_{i-2} and v_{i-2} in each subphase.

- We consider the densities of u_{i-2} and v_{i-2} at the beginning and end of Subphase 1 (see Fig. 29). At the beginning of Subphase 1, Density$(u_{i-2}) = \rho_{i-1}$ and Density$(v_{i-2}) = 4\rho_{i+1}-\rho_{i-1}$ by the rebalance property.
  Here and below we assume that $4\rho_{i+1}-\rho_{i-1}\leq\tau_{i-1}$. The alternative, that $4\rho_{i+1}-\rho_{i-1}>\tau_{i-1}$, is the simple case. Then Density$(v_{i-2}) = \tau_{i-1}$. As a consequence, there are only two subphases in Phase 2 of u_i, and the recurrence is simpler.
  Subphase 1 ends with the density of u_{i-2} reaching its upper threshold τ_{i-2}. The number of sweeps of u_κ in Subphase 1 is exactly equal to $\mathcal{C}_\kappa(i-1,1)$ because both of them start at Density$(u_{i-2}) = \rho_{i-1}$ and end with Density$(u_{i-2}) = \tau_{i-2}$.

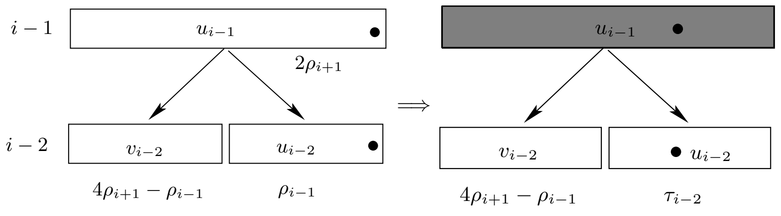

Fig. 29. Subphase 1 of Phase 2 starts from Density$(u_{i-2}) = \rho_{i-1}$ (left) and ends at Density$(u_{i-2}) = \tau_{i-2}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of Subphase 1 is shaded.

- We next consider the densities of u_{i-2} and v_{i-2} at the beginning and end of Subphase 2 (see Fig. 30). The beginning of Subphase 2 is right after the rebalance of

u_{i-1}. Notice that there are $4\rho_{i+1}\text{Cap}(u_{i-2})$ elements before the marker element x and $(\tau_{i-2} - \rho_{i-1})\text{Cap}(u_{i-2})$ elements after x. Because $\tau_{i-2} \leq 5\rho_{i-2}$ by (23), we obtain

$$\tau_{i-2} - \rho_{i-1} \leq 5\rho_{i-2} - \rho_{i-1}.$$

Because $\rho_{i-2} < \rho_{i-1} < \rho_{i+1}$ by (1), we have

$$\tau_{i-2} - \rho_{i-1} < 4\rho_{i+1}. \tag{36}$$

Equation (36) says that the number of elements after x is less than the number of elements before x in node u_{i-1}. Therefore, the marker element x will be in the right child of u_{i-1} after the sweep. By the same argument as in Phase 3, we assume the sweep at level $i-2$ is constrained by the hammer constraint. Otherwise, $\text{Density}(v_{i-2}) = \tau_{i-1}$, and there are only two subphases in Phase 2.

Thus, we consider the case that v_{i-2} is still below its parent's threshold, i.e., Phase 2 needs a third subphase before it finishes.

We now bound the number of sweeps of u_κ in Subphase 2. Since the marker element x is the leftmost element in u_{i-2}, and thus within u_{i-2} we have the head-insert case. Therefore, by Theorem 3, there are $O(2^{i-2-\kappa})$ sweeps of u_κ in Subphase 2.

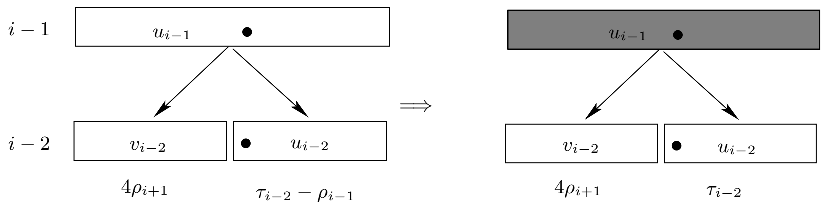

Fig. 30. Subphase 2 of Phase 2 starts from $\text{Density}(u_{i-2}) = \tau_{i-2} - \rho_{i-1}$ (left) and ends at $\text{Density}(u_{i-2}) = \tau_{i-2}$ (right). The marker element x is indicated by a black dot. The region that is rebalanced at the end of Subphase 2 is shaded.

- Finally, we consider the densities of u_{i-2} and v_{i-2} at the beginning and end of Subphase 3 (see Fig. 31). Subphase 3 is same as Subphase 2 of Phase 3. By the same argument, the number of sweeps of u_κ in Subphase 3 is $C_\kappa(i-3,1)$.

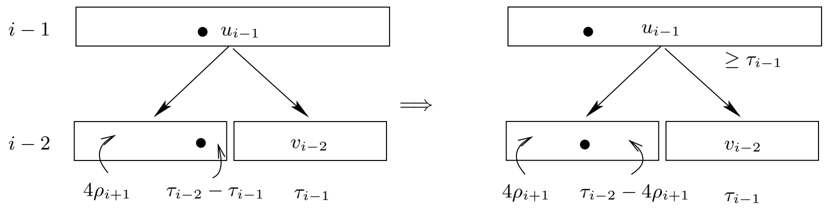

Fig. 31. Subphase 3 of Phase 2 starts from $\text{Density}(u_{i-2}) = 4\rho_{i+1} + \tau_{i-2} - \tau_{i-1}$ (left) and ends at $\text{Density}(u_{i-2}) = \tau_{i-2}$ (right). The marker element x is indicated by a black dot. At the end of Subphase 3, node u_i is rebalanced.

In summary, there are at most three subphases in Phase 2 and the number of sweeps in these three subphases is at most $\mathcal{C}_\kappa(i-1,1)$ plus $O(2^{i-3-\kappa})$ plus $\mathcal{C}_\kappa(i-3,1)$, which establishes (28).

We can now prove our desired bound. Plugging (28) and (29) into (26), we obtain

$$\mathcal{C}_\kappa(i+1,1) \leq \mathcal{C}_\kappa(i,1) + \mathcal{C}_\kappa(i-1,1) + 2\mathcal{C}_\kappa(i-3,1) + O(2^{i-\kappa})\,.$$

We prove our bound by induction. Assume $\mathcal{C}_\kappa(j,1) \leq \beta 2^{j-\kappa}$ for $j \leq i$ and the constant in $O(2^{i-\kappa})$ is α. If we choose β bigger than 4α, then

$$\begin{aligned}\mathcal{C}_\kappa(i+1,1) &\leq \beta 2^{i-\kappa} + \beta 2^{i-1-\kappa} + 2\beta 2^{i-3-\kappa} + \alpha 2^{i-\kappa} \\ &= \frac{7}{4}\beta 2^{i-\kappa} + \alpha 2^{i-\kappa} \\ &\leq \beta 2^{i+1-\kappa}.\end{aligned}$$

Therefore, $\mathcal{C}_\kappa(i+1,1) \leq \beta 2^{i+1-\kappa}$ is true for all $i > 0$, as claimed.

□