Biased Skip Lists*

Amitabha Bagchi[†] Adam L. Buchsbaum[‡] Michael T. Goodrich[§]

March 30, 2004

Abstract

We design a variation of skip lists that performs well for generally biased access sequences. Given n items, each with a positive weight w_i, $1 \leq i \leq n$, the time to access item i is $O\left(1 + \log \frac{W}{w_i}\right)$, where $W = \sum_{i=1}^{n} w_i$; the data structure is dynamic. We present two instantiations of biased skip lists, one of which achieves this bound in the worst case, the other in the expected case. The structures are nearly identical; the deterministic one simply ensures the balance condition that the randomized one achieves probabilistically. We use the same method to analyze both.

1 Introduction

The primary goal of data structures research is to design data organization mechanisms that admit fast access and update operations. For a generic n-element ordered data set that is accessed and updated uniformly, this goal is typically satisfied by dictionaries that achieve $O(\log n)$-time performance for searches and updates; for example, AVL-trees [2], red-black trees [12], and (a, b)-trees [13].

Nevertheless, many dictionary applications involve sets of weighted data items that are searched and updated non-uniformly according to those weights; that is, they are *biased*. For example, most operating systems textbooks (e.g., see Stallings [23]) devote major coverage to methods for dealing with biasing in memory requests. Other recent examples of biased sets include client web server requests [11] and DNS lookups [6]. For such applications, a *biased search structure* is more appropriate—that is, a structure that achieves search times faster than $\log n$ for highly weighted items. Biased searching is also useful in auxiliary structures deployed inside other data structures [5, 10, 21].

Formally, a *biased dictionary* is a data structure that maintains an ordered set X, each element i of which has a *weight*, w_i; without loss of generality, we assume $w_i \geq 1$. The operations are as follows:

Search(X, i)**.** Determine if i is in X.

Insert(X, i)**.** Add i to X.

*An extended abstract of this work appears in *Proc. 13th Annual International Symposium on Algorithms and Computation*, *Lecture Notes in Computer Science*, v. 2518, pp. 1–13, 2002.

†Dept. of Information & Computer Science, University of California, Irvine, CA 92697-3425, bagchi@ics.uci.edu. Supported by DARPA Grant F30602-00-2-0509 and NSF Grant CCR-0098068.

‡AT&T Labs, Shannon Laboratory, 180 Park Ave., Florham Park, NJ 07932, alb@research.att.com.

§Dept. of Information & Computer Science, University of California, Irvine, CA 92697-3425, goodrich@ics.uci.edu. Supported by DARPA Grant F30602-00-2-0509 and NSF Grant CCR-0098068.

Delete(X, i)**.** Delete i from X.

Join(X_L, X_R)**.** Assuming that $i < j$ for each $i \in X_L$ and $j \in X_R$, create a new set $X = X_L \cup X_R$. The operation destroys X_L and X_R.

Split(X, i)**.** Assuming without loss of generality that $i \notin X$, create $X_L = \{j \in X : j < i\}$ and $X_R = \{j \in X : j > i\}$. The operation destroys X.

FingerSearch(X, i, j)**.** Determine if j is in X, beginning the search with a handle in the data structure to element $i \in X$.

Reweight(X, i, w'_i)**.** Change the weight of i to w'_i.

In this paper, we study efficient data structures for biased data sets subject to these operations. We desire search times that are asymptotically optimal and update times that are also efficient. For example, consider the case when w_i is the number of times item i from a set of n items is accessed over a sequence of m searches, where $m = W = \sum_{i=1}^{n} w_i$. A biased dictionary with $O\left(\log \frac{W}{w_i}\right)$ search time for the i'th item can perform this sequence in $O(m(1 - \sum_{i=1}^{n} p_i \log p_i))$ time, where $p_i = \frac{w_i}{m}$, assuming that each $w_i \geq 1$; this is asymptotically optimal [1].

We therefore desire $O\left(\log \frac{W}{w_i}\right)$ search times and similar update times for general biased data (with arbitrary weights). Moreover, we seek biased structures that would be simple to implement and that do not require major restructuring operations, such as tree rotations, to achieve biasing. Tree rotations in particular make structures less amenable to augmentation, for such rotations often require the complete rebuilding of auxiliary structures stored at the affected nodes.

1.1 Related Prior Work

The study of biased data structures for weighted data is a classic topic in algorithmics. Early work includes a dynamic programming method by Knuth [14] for constructing a static biased binary search tree for items weighted by their search frequencies (see also Knuth [15]). Subsequent work focuses primarily on methods for achieving search times within a constant factor of optimal while also being able to perform updates efficiently. Most of the known methods for constructing dynamic biased data structures do so using search trees, and they differ from one another primarily in their degree of complication and whether or not their resulting time bounds are amortized, randomized, or worst case.

Sleator and Tarjan [22] introduce the theoretically elegant *splay tree* data structure, which automatically adjusts itself to achieve optimal amortized biased access times for access-frequency weights. Splay trees achieve this result by performing a large number of tree rotations after every access; they do not store any balance or weight information. From a theoretical standpoint, the splay tree is a beautiful structure, but the large number of tree rotations done after every access makes it less practically efficient than even AVL-trees in many applications [3]. These rotations also make splay trees a poor choice for augmentation with auxiliary structures at internal nodes.

Bent, Sleator, and Tarjan [4] and Feigenbaum and Tarjan [9] design biased search trees for arbitrary weights that significantly reduce, but do not eliminate, the number of tree rotations needed in order to maintain a biased search tree. They offer efficient worst-case and amortized performance of biased dictionary operations but do so with complicated implementations.

Alternately, Seidel and Aragon [20] demonstrate randomized bounds with *treaps*. Like splay trees, treaps achieve biasing by performing a large number of rotations after every access. Their data structure is elegant and efficient in practice, but its performance does not achieve bounds that are efficient in a worst-case or amortized sense.

As mentioned above, these biased search tree structures achieve their biasing using tree rotations. Pugh [19] introduces an alternative *skip list* structure, which efficiently implements an unbiased dictionary without using rotations. Skip lists store the items in series of a linked lists, which are themselves linked together in a leveled fashion. Pugh presents skip lists as a randomized structure that is easily implemented and shows that they are empirically faster than fast balanced search trees, such as AVL-trees. Searches and updates take $O(\log n)$ expected time in skip lists, with no rotations or other rebalancing needed for updates. Exploiting the relationship between skip lists and (a, b)-trees, Munro, Papadakis, and Sedgewick [18] show how to implement a deterministic version of skip lists that achieves similar bounds in the worst case through the use of simple promote and demote operations.

In terms of biased skip list structures, much less prior work exists. Mehlhorn and Näher [17] anticipated biased skip lists but claimed only a partial result and omitted details and analysis of such a structure. Roura and Martínez [16] designed an algorithm that takes a probability distribution over accesses to a dictionary and builds a static weighted skip list structure that minimizes the average access cost. Recently, Ergun *et al.* [7, 8] presented a biased skip list structure that is designed for a specialized notion of biasing, in which access to an item i takes $O(\log r(i))$ expected time, where $r(i)$ is the number of items accessed since the last time i was accessed. Their data structure is incomparable to a general biased dictionary, as each provides properties not present in the other.

1.2 Our Results

In this paper, we present a comprehensive design of a biased version of the skip list data structure. It combines techniques underlying deterministic skip lists [18] with Mehlhorn and Näher's suggestion [17]. Our methods work for arbitrarily defined item weights and provide optimal search times based on these weights (to within constant factors). Moreover, since our methods are built using the technology of skip lists, they do not employ tree rotations to achieve biasing. We present complete descriptions of all the biased dictionary operations, with time performances that compare favorably with those of the various versions of biased search trees. We give both deterministic and randomized implementations of biased skip lists. Our deterministic structure achieves worst-case running times that are similar to those of biased search trees [4, 9] but uses techniques that are arguably simpler. A node in a deterministic biased skip list is assigned an initial level based on its weight, and simple invariants govern promotion and demotion of node levels to ensure the desired access time. Our randomized structure achieves expected bounds that are similar to the respective amortized and randomized bounds of splay trees [22] and treaps [20]. Our randomized structure does not use partial rebuilding and hence does not need any amortization of its own. Table 1 juxtaposes our results against biased search trees, splay trees, and treaps.

In Section 2, we define our deterministic biased skip list structure, and in Section 3 we describe how to perform updates efficiently in this structure. In Section 4 we describe a simple, randomized variation of biased skip lists and analyze its performance. We conclude in Section 5.

Table 1: Time bounds for biased data structures. In all bounds, W is the total weight of all items before the operation; $V(i,j) = \sum_{k=i}^{j} w_k$. For each table entry, E, the associated time bound is $O(1+E)$.

	Biased Search Trees [4]	Splay Trees [22] amort.	Treaps [20] rand.	Biased Skip Lists w.c. & rand.
Search(X,i)	$\log \frac{W}{w_i}$ amort./w.c.	$\log \frac{W}{w_i}$	$\log \frac{W}{w_i}$	$\log \frac{W}{w_i}$
Insert(X,i)	$\log \frac{W+w_i}{\min(w_{i-}+w_{i+},w_i)}$ amort. $\log \frac{W}{w_{i-}+w_{i+}} + \log \frac{W+w_i}{w_i}$ w.c.	$\log \frac{W}{\min(w_{i-},w_{i+})} + \log \frac{W+w_i}{w_i}$	$\log \frac{W+w_i}{\min(w_{i-},w_i,w_{i+})}$	$\log \frac{W+w_i}{\min(w_{i-},w_i,w_{i+})}$
Delete(X,i)	$\log \frac{W}{w_i}$ amort. $\log \frac{W}{w_i} + \log \frac{W-w_i}{w_{i-}+w_{i+}}$ w.c.	$\log \frac{W}{w_i} + \log \frac{W-w_i}{w_{i-}}$	$\log \frac{W+w_i}{\min(w_{i-},w_i,w_{i+})}$	$\log \frac{W+w_i}{\min(w_{i-},w_i,w_{i+})}$
Join(X_L,X_R)	$\log \frac{W_L+W_R}{w_{L_{\max}}+w_{R_{\min}}}$ w.c.	$\log \frac{W_L+W_R}{w_{L_{\max}}}$	$\log \frac{W_L}{w_{L_{\max}}} + \log \frac{W_R}{w_{R_{\min}}}$	$\log \frac{W_L}{w_{L_{\max}}} + \log \frac{W_R}{w_{R_{\min}}}$
Split(X,i)	$\log \frac{W}{w_{i-}+w_{i+}}$ amort./w.c.	$\log \frac{W}{\min(w_{i-},w_{i+})}$	$\log \frac{W_L}{w_{L_{\max}}} + \log \frac{W_R}{w_{R_{\min}}}$	$\log \frac{W}{\min(w_{i-},w_{i+})}$
Reweight(X,i,w_i')	$\log \frac{\max(W,W')}{\min(w_i,w_i')}$ amort. $\log \frac{W}{w_i} + \log \frac{W'}{w_i'}$ w.c.		$\log \frac{\max(w_i,w_i')}{\min(w_i,w_i')}$	$\log \frac{W'}{\min(w_i,w_i')}$
FingerSearch(X,i,j)			$\log \frac{V(i,j)}{\min(w_i,w_j)}$	$\log \frac{V(i,j)}{\min(w_i,w_j)}$

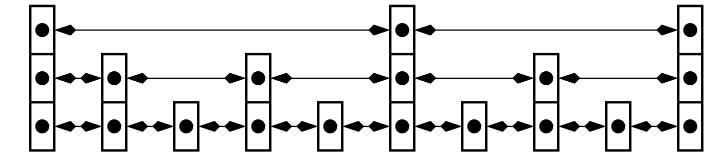

Figure 1: A skip list for the set $X = \{1, 5, 10, 22, 50, 60, 75, 80\}$.

2 Biased Skip Lists

A *skip list* [19] S is a dictionary data structure storing an ordered set X, the items of which we number 1 through $|X|$. Each item $i \in X$ has a key, x_i, and a corresponding *node* in the skip list of some integral *height*, $h_i \geq 0$. The *height* of S is $H(S) = \max_{i \in X} h_i$. The *depth*, d_i, of item i is $H(S) - h_i$. We use the terms item, node, and key interchangeably where convenient; the context clarifies any ambiguity. We assume without loss of generality that the keys in X are unique: $x_i < x_{i+1}$, $1 \leq i < |X|$.

Each node i is implemented by a linked list or array of length $h_i + 1$, which we refer to as the *tower* for that node. The *level-j successor* of a node i is the least node $\ell > i$ of height $h_\ell \geq j$; i.e., no node $i < k < \ell$ has height $h_k \geq j$. Symmetrically, the *level-j predecessor* of node i is the greatest node $\ell < i$ of height $h_\ell \geq j$. For node i and each $0 \leq j \leq h_i$, the j'th element of the node contains pointers to the j'th elements of the level-j successor and predecessor of i. Two distinct nodes $x < y$ are called *consecutive* if and only if $h_z < \min(h_x, h_y)$ for all $x < z < y$. A *plateau* is a maximal sequence of consecutive nodes of equal height.

For convenience we assume sentinel nodes of height $H(S)$ at the beginning (with key $-\infty$) and end (with key ∞) of S; in practice, this assumption is not necessary. We orient the pointers so that the skip list stores items in left-to-right order, and the node levels progress bottom to top. See Figure 1.

To search for an item with key K we start at level $H(S)$ of the left sentinel. When searching at level i from some node we follow the level-i links to the right until we find a key matching K or a pair of nodes j, k such that k is the level-i successor of j and $x_j < K < x_k$. We then traverse one level down and continue the search at level $i - 1$ from node j. The search ends with success if we find a node with key K, or failure if we find nodes j and k as above on level 0. See Figure 2.

We describe a deterministic, biased version of skip lists. In addition to a key, x_i, each item $i \in X$ has a *weight*, w_i; without loss of generality, we assume $w_i \geq 1$. We define the *rank* of item i as $r_i = \lfloor \log_a w_i \rfloor$, where a is a constant to be defined shortly.

Definition 2.1 *For every a and b such that $1 < a \leq \left\lfloor \frac{b}{2} \right\rfloor$, an (a, b)-*biased skip list* is one in which each item has height $h_i \geq r_i$ and the following invariants hold.*

(I1) *For all $0 \leq i \leq H(S)$, there are never more than b consecutive items of height i.*

(I2) *For each node x and all $r_x < i \leq h_x$, there are at least a nodes of height $i - 1$ between x and any consecutive node of height at least i.*

To derive exact bounds for the case when an item does not exist in the skip list we modify the structure to eliminate redundant pointers. For every pair of adjacent items $i, i + 1$, we set the

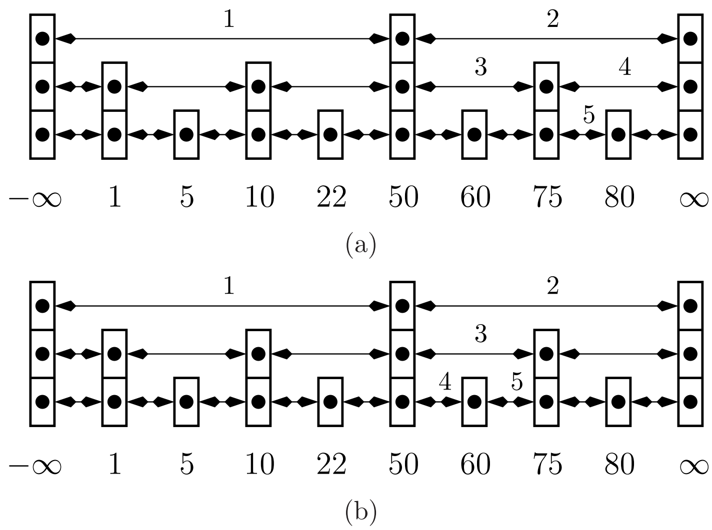

Figure 2: (a) Searching for key 80 in the skip list of Figure 1. Numbers over the pointers indicate the order in which they are traversed. (b) Similarly, searching for key 65.

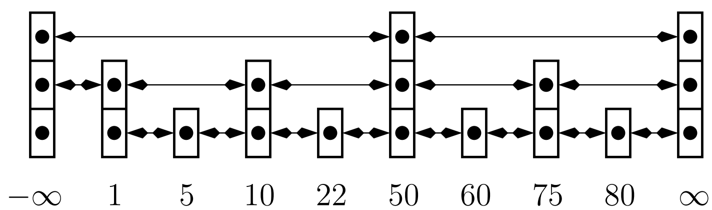

Figure 3: The skip list of Figure 1 with the unnecessary level-0 pointers between $-\infty$ and 1 set to nil.

pointers between them on levels 0 through $\min(h_i, h_{i+1})-1$ to nil. (See Figure 3.) When searching for an item $i \notin X$, we assert failure immediately upon reaching a nil pointer. It suffices, in fact, to ensure only that the pointers between them on level $\min(h_i, h_{i+1}) - 1$ are nil; the pointers below this level remain undefined.

Throughout the remainder of the paper, we define $W = \sum_{i \in X} w_i$ to be the weight of S before any operation. For any key i, we denote by i^- the item in X with largest key less than i, and by i^+ the item in X with smallest key greater than i. The main result of our definition of biased skip lists is summarized by the following lemma, which bounds the depth of any node.

Lemma 2.2 (Depth Lemma) *The depth of any node i in an (a,b)-biased skip list is $O\left(\log_a \frac{W}{w_i}\right)$.*

Before we prove the depth lemma, consider its implication on the *access time* for key i, defined to be the time it takes to find i in S if $i \in X$ or to find the pair i^-, i^+ in S if $i \notin X$.

Corollary 2.3 (Access Lemma) *The access time for any key i in an (a,b)-biased skip list is $O\left(1 + b \log_a \frac{W}{w_i}\right)$ if $i \in X$ and $O\left(1 + b \log_a \frac{W}{\min(w_{i^-}, w_{i^+})}\right)$ if $i \notin X$.*

Proof. By **(I1)**, at most $b+1$ pointers are traversed at any level during a search. Because a search stops upon reaching the first nil pointer, the Depth Lemma thus implies the result. $\square$

It is important to note that while all the bounds we prove rely on W, the data structure itself need not maintain this value.

To prove the depth lemma, observe that the number of items of any given rank that can appear at higher levels decreases geometrically by level. Define $N_i = |\{x : r_x = i\}|$ and $N'_i = |\{x : r_x \leq i \wedge h_x \geq i\}|$.

Lemma 2.4 $N'_i \leq \sum_{j=0}^{i} \frac{1}{a^{i-j}} N_j$.

Proof. We prove the lemma by induction. The base case, $N'_0 = N_0$, is true by definition. For $i > 0$, **(I2)** implies that

$$\begin{aligned} N'_{i+1} &\leq N_{i+1} + \left\lfloor \frac{1}{a} N'_i \right\rfloor \\ &\leq N_{i+1} + \frac{1}{a} N'_i \end{aligned}$$

which, together with the induction hypothesis, proves the lemma. $\square$

Intuitively, this implies that a node promoted to a higher level is supported by enough weight associated with items at lower levels. Define $W_i = \sum_{r_x \leq i} w_x$.

Corollary 2.5 $W_i \geq a^i N'_i$.

Proof. By definition,

$$\begin{aligned} W_i &\geq \sum_{j=0}^{i} a^j N_j \\ &= a^i \sum_{j=0}^{i} \frac{1}{a^{i-j}} N_j. \end{aligned}$$

Lemma 2.4 yields the result. $\square$

Define $R = \max_{x \in X} r_x$. Any nodes with height exceeding R must have been promoted from lower levels to maintain the invariants. **(I2)** thus implies that $H(S) \leq R + \log_a N'_R$, and therefore the maximum possible depth of an item i is $d_i \leq H(S) - r_i \leq R + \log_a N'_R - r_i$.

By Corollary 2.5, $W = W_R \geq a^R N'_R$. Therefore $\log_a N'_R \leq \log_a W - R$. Hence, $d_i \leq \log_a W - r_i$. The Depth Lemma follows, because $\log_a w_i - 1 < r_i \leq \log_a w_i$. $\square$

(I1) and **(I2)** resemble the invariants defining (a,b)-skip lists [18], but **(I2)** is stronger than their analogue. In fact, **(I2)** is stronger than necessary to prove the Depth Lemma. It would suffice for a node of height h exceeding its rank, r, to be supported by at least a items to each side only at level $h-1$, not at every level between r and $h-1$. The stronger invariant is easier to maintain, however; the update procedures in the next section rely on the support occurring at every level.

3 Updating Biased Skiplists

We present and analyze deterministic procedures to update biased skip lists.

First, we define the profile of an item i. For $h_{i^-} \leq j \leq H(S)$, let L^i_j be the level-j predecessor of i; for $h_{i^+} \leq j \leq H(S)$, let R^i_j be the level-j successor of i. Define the ordered sets $PL(i) = \Big(j : h_{L^i_j} = j,\ h_{i^-} \leq j \leq H(S)\Big)$ and $PR(i) = \Big(j : h_{R^i_j} = j,\ h_{i^+} \leq j \leq H(S)\Big)$. $PL(i)$ (rsp., $PR(i)$) is the set of distinct heights of the nodes to the left (rsp., right) of i. We call the ordered set $\Big(L^i_j : j \in PL(i)\Big) \cup \Big(R^i_j : j \in PR(i)\Big)$ the *profile* of i. We call the subset of predecessors the *left profile* and the subset of successors the *right profile* of i. For example, in Figure 1, $PL(60) = (3)$; $PR(60) = (2, 3)$; the left profile of 60 is (50); and the right profile of 60 is $(75, \infty)$.

The profile definitions assume $i \in S$, but they are also precise when $i \notin S$, in which case they apply to the node that would contain key i. Given node i (if $i \in S$) or nodes i^- and i^+ (if $i \notin S$), we can trace i's profile back from lowest-to-highest nodes by starting at i^- (rsp., i^+) and, at any node x, iteratively finding its level-$(h_x + 1)$ predecessor (rsp., successor), until we reach the left (rsp., right) sentinel.

3.1 Inserting An Item

The following procedure inserts a new item with key i into an (a, b)-biased skip list S. We assume that i does not already exist in the skip list, or else we discover it in Step 1.

Procedure Insert(S, i)

1. Search S for i to discover the pair i^-, i^+.

2. Create a new node of height r_i to store i, and insert it between i^- and i^+ in S, tracing through i's profile to splice predecessors and successors as in a standard skip list [19].

3. Restore **(I2)**, if necessary. Any node x in the left (sym., right) profile of i might need to have its height demoted, because i might interrupt a sequence of consecutive nodes of height less than h_x, leaving fewer than a to its left (sym., right). In this case, x is demoted to the next lower height in the profile (or r_x, whichever is higher).

   More precisely, for j in turn from h_{i^-} up through r_i, if $j \in PL(i)$, consider node $u = L^i_j$. If **(I2)** is violated at node u, then demote u to height r_u if $u = i^-$ and otherwise to height $\max(j', r_u)$, where j' is the predecessor of j in $PL(i)$; let h'_u be the new height of u. If the demotion violates **(I1)** at level h'_u, then, among the $k \in (b, 2b]$ consecutive items of height h'_u, promote the $\lfloor \frac{k}{2} \rfloor$'th node (in order) to height $h'_u + 1$. (See Figure 4.) Iterate at the next j. Symmetrically process right profile of i.

4. Restore **(I1)**, if necessary. Starting at node i and level $j = r_i$, if node i violates **(I1)** at level j, then, among the $b + 1$ consecutive items of height j, promote the $\lfloor \frac{b+1}{2} \rfloor$'th node (in order), u, to height $j + 1$, and iterate at node u and level $j + 1$. Continue until the violations stop. (See Figure 5.)

Lemma 3.1 *Invariants **(I1)** and **(I2)** are true after* **Insert(S, i)**.

Proof. Assume the invariants were true before the insertion. To the left of node i, **(I2)** can be violated at most once at each level $j \in PL(i) \cap [h_{i^-}, r_i]$, where node i might split a sequence of consecutive nodes of height less than j. Consider the least j and the associated node $u = L^i_j$ where

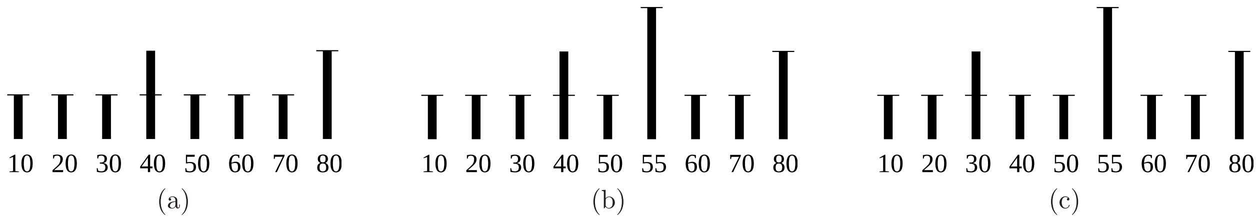

Figure 4: (a) A (2,4)-biased skip list. Nodes are drawn to reflect their heights; hatch marks indicate ranks. Pointers between nodes are omitted. (b) After the insertion of 55 with rank 3, node 40 violates **(I2)**. (c) After the demotion of node 40 and compensating promotion of node 30.

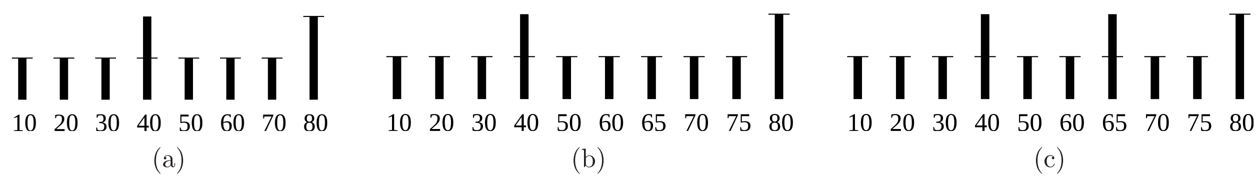

Figure 5: (a) The (2,4)-biased skip list of Figure 4(a). (b) **(I1)** is violated by the insertion of 65 and 75 with rank 1 each. (c) After promoting node 65.

such a violation occurs. Demoting u in Step 3 restores **(I2)** at level j, by the assumption that **(I2)** was true before the operation. The demotion might cause there to be more than b consecutive nodes of height h'_u around u, however, in which case the promotion of the node in the middle restores **(I1)**. By definition of profile, h'_u is the only height at which this demotion might incur a compensating promotion. Since $a \leq \lfloor \frac{b}{2} \rfloor$, this compensating promotion cannot violate **(I2)**. By induction, iterating Step 3 up to level r_i restores **(I2)** to the left of i. Symmetrically argue for the nodes to the right of i.

Step 4 restores **(I1)** at level r_i if it was violated by the insertion of node i. Promoting the node in the middle cannot violate **(I2)**, because $a \leq \lfloor \frac{b}{2} \rfloor$. The promotion might violate **(I1)** at the next level, however, in which case the iteration of Step 4 restores the invariant. □

Theorem 3.2 *Inserting an item i in an (a, b)-biased skip list takes* $O\left(1 + b \log_a \frac{W + w_i}{\min(w_{i^-}, w_i, w_{i^+})}\right)$ *time.*

Proof. Lemma 3.1 proves correctness of **Insert**(S, i).

By the Depth and Access Lemmas, Steps 1 and 2 take $O\left(1 + b \log_a \frac{W + w_i}{\min(w_{i^-}, w_i, w_{i^+})}\right)$ time. If $\min(h_{i^-}, h_{i^+}) \leq r_i$, Step 3 performs $O(b)$ work at each level between $min(h_{i^-}, h_{i^+})$ and r_i; Step 4 performs $O(b)$ work at each level from r_i through $H(S)$. Again applying the Depth Lemma yields the result. □

3.2 Deleting An Item

Deletion is the inverse of insertion.

Procedure Delete(S, i)

1. Search S to discover i.
2. Find the immediate neighbors i^- and i^+. Remove i, and splice predecessors and successors as required.
3. Restore **(I1)**, if necessary. (Removing i might unite sequences of consecutive nodes into sequences of length exceeding b.) For j in turn from $\min(h_{i^-}, h_{i^+})$ up through $h_i - 1$, if removing i violates **(I1)** at level j, consider the $k \in (b, 2b]$ consecutive nodes of height j, and promote the $\lfloor \frac{k}{2} \rfloor$'th among them to height $j+1$. Iterate at the next j.
4. Restore **(I2)**, if necessary. (Removing i might decrease the length of the sequence of consecutive nodes of height h_i to $a-1$, in which case one of the delineating towers might need to be demoted, and so on from there.) Starting at node i and the least $j \in PL(i)$ greater than h_i, if **(I2)** is violated at node $u = L^i_j$, then demote u to height $\max(h_i, r_u)$; let h'_u be the new height of u. For each j' in turn from h'_u up through h_u (the old height of u), if the demotion violates **(I1)** at level j', then, among the $k \in (b, 2b]$ consecutive items of height j', promote the $\lfloor \frac{k}{2} \rfloor$'th among them to height j'. Iterate, checking for an **(I2)** violation at the old height h_u, continuing until the violations stop. Symmetrically process the right profile of i.

Lemma 3.3 *Invariants **(I1)** and **(I2)** are true after* **Delete(**S, i**)**.

Proof. Assume the invariants were true before the deletion. **(I1)** can be violated at most once at each level $\min(h_{i^-}, h_{i^+}) \leq j < h_i$, where the removal of node i might unite two previously separate sequences of consecutive nodes of height j. Step 3 restores **(I1)** at each such level j. Promoting the node in the middle cannot violate **(I2)**, because $a \leq \lfloor \frac{b}{2} \rfloor$, nor can the promotion propagate to higher levels, because the previous existence of node i at level j satisfied **(I1)**.

(I2) can be violated by the removal of i or the subsequent demotion of a predecessor (rsp., successor). By the assumption that **(I2)** held before the operation, any node so violating **(I2)** need be demoted no farther than the height of the preceding node in the left (rsp., right) profile of i (or i in the case of i^- (rsp., i^+)). As in Step 3 of **Insert**, the demotion might require compensating promotions, each of which cannot percolate higher. Step 4 thus restores **(I2)**. □

Theorem 3.4 *Deleting an item i from an (a, b)-biased skip list takes $O\left(1 + b \log_a \frac{W}{\min(w_{i^-}, w_i, w_{i^+})}\right)$ time.*

Proof. Lemma 3.3 proves correctness of **Delete(**S, i**)**.

By the Depth and Access Lemmas, Steps 1 and 2 take $O\left(1 + b \log_a \frac{W}{\min(w_{i^-}, w_i, w_{i^+})}\right)$ time. If $\min(h_{i^-}, h_{i^+}) \leq h_i$, Step 3 performs $O(b)$ work at each level between $\min(h_{i^-}, h_{i^+})$ and h_i; Step 4 performs $O(b)$ work at each level from h_i through $H(S)$. Again applying the Depth Lemma yields the result. □

3.3 Joining Two Skiplists

Consider two biased skip lists, S_L and S_R, of total weights W_L and W_R respectively. The item with the largest key in S_L is denoted $L_{\max}$, and the item with the smallest key in S_R is denoted $R_{\min}$. If $L_{\max} < R_{\min}$, we can *join* S_L and S_R to form a single biased skip list.

Procedure Join(S_L, S_R**)**

1. Trace through the profiles of $L_{\max}$ and $R_{\min}$ to splice the pointers leaving S_L together with the pointers going into S_R.

2. Restore **(I1)**, if necessary. For j in turn from $\max(h_{L_{\max}}, h_{R_{\min}})$ through $\max(H(S_L), H(S_R))$, if **(I1)** is violated at level j, then among the $k \in (b, 2b+1]$ consecutive items of height j, promote the $\lfloor \frac{k}{2} \rfloor$'th node (in order) to height $j+1$.

Lemma 3.5 *Invariants* ***(I1)*** *and* ***(I2)*** *are true after* **Join**(S_L, S_R).

Proof. Assuming the invariants were true before the join, splicing the pointers cannot violate **(I2)**, because nodes never lose predecessors or successors.

(I1) can be violated by joining two sequences of nodes at a given level; $\max(h_{L_{\max}}, h_{R_{\min}})$ is the lowest height at which such a violation can occur. Promoting the node in the middle cannot violate **(I2)**, because $a \leq \lfloor \frac{b}{2} \rfloor$. The promotion can add another node to the next higher level, but the splicing procedure left no more than $2b$ nodes there, by the assumption that **(I1)** was true before the join. Thus, no more than $2b+1$ nodes occur at any level prior to a promotion, and so the promotion strategy restores **(I1)**. □

Theorem 3.6 *Joining* (a,b)*-biased skip lists* S_L *and* S_R *takes* $O\left(1 + b\log_a \frac{W_L}{w_{L_{\max}}} + b\log_a \frac{W_R}{w_{R_{\min}}}\right)$ *time.*

Proof. Lemma 3.5 proves correctness of **Join**(S_L, S_R). Step 1 performs $O(b)$ work at each level between $\min(h_{L_{\max}}, h_{R_{\min}})$ and $\min(H(S_L), H(S_R))$. Step 2 potentially performs $O(b)$ work at each level between $\max(h_{L_{\max}}, h_{R_{\min}})$ and $\max(H(S_L), H(S_R))$. Putting this together and applying the Depth Lemma yields the result. □

3.4 Splitting A Skiplist

Given a biased skip list S of total weight W and a key $i \notin S$, we can *split* S into two biased skip lists S_L, containing keys in S less than i, and S_R, containing keys in S greater than i. (We can formulate this equivalently when $i \in S$.)

Procedure Split(S, i)

1. Perform **Insert**(S, i), where the weight of i is $a^{H(S)+1}$.

2. Disconnect the pointers between i and its predecessors to form S_L; disconnect the pointers between i and its successors to form S_R.

Theorem 3.7 *Splitting an* (a,b)*-biased skip list on key* i *takes* $O\left(1 + b\log_a \frac{W}{\min(w_{i^-}, w_{i^+})}\right)$ *time.*

Proof. Lemma 3.1 proves that **(I1)** and **(I2)** are true after Step 1. Because i is taller than all of its predecessors and successors, disconnecting the pointers between them and i in Step 2 does not violate either invariant. Thus, Procedure **Split**(S, i) is correct.

Step 1 takes $O\left(1 + b\log_a \frac{W}{\min(w_{i^-}, w_{i^+})}\right)$ time, by Theorem 3.2 together with the observation that $w_i = \Theta(W)$, and yields a biased skip list of height $H(S)+1$. Step 2 takes $O(2(H(S)+1) - h_{i^-} - h_{i^+})$ time. Applying the Depth Lemma finishes the proof. □

3.5 Finger Searching

We can search for a key j in a biased skip list S starting at any node i (not just the left sentinel) to which we are given an initial pointer (or *finger*). Assume without loss of generality that $j > i$. The following procedure is symmetric for the case $j < i$.

Procedure FingerSearch(S, i, j)

1. Initialize $u \leftarrow i$, $h \leftarrow r_i$.

2. (Up phase.) If $R_h^u \geq j$, then go to Step 3. Otherwise, if $h < h_u$, set $h \leftarrow h + 1$; else set $u \leftarrow R_h^u$; iterate at Step 2.

3. (Down phase.) Search from u, starting at height h, as in the normal skip-list search procedure outlined in Section 2.

The up phase moves up and to the right in the skip list until we detect a node $u < j$ with a level-h successor (for some h) $R_h^u > j$. That the procedure finds j if $j \in S$ or the pair j^-, j^+ if $j \notin S$ follows from the correctness of the vanilla search procedure and that we enter the down phase at the specified node u and height h.

Define $V(i, j) = \sum_{i \leq u \leq j} w_u$.

Lemma 3.8 *For any node u and $h \in [r_u, h_u]$, $V(L_h^u, u) \geq a^h$ and $V(u, R_h^u) \geq a^h$.*

Proof. We prove $V(L_h^u, u) \geq a^h$; the other direction is symmetric. If $h = r_u$, then $V(L_h^u, u) \geq w_u \geq a^h$ by definition. Otherwise, $h > r_u$, and by **(I2)**, there are at least a elements of height $h-1$ between u and L_h^u. By induction, $V(L_h^u, u) \geq a a^{h-1} = a^h$. □

Theorem 3.9 *Accessing an item j in an (a, b)-biased skip list, given a pointer to an item i, takes $O\left(1 + b \log_a \frac{V(i,j)}{\min(w_i, w_j)}\right)$ time if $j \in X$ and $O\left(1 + b \log_a \frac{V(i,j^+)}{\min(w_i, w_{j^-}, w_{j^+})}\right)$ time if $j \notin X$.*

Proof. We can assume constant-time access to level r_i of any node i without affecting previous time bounds. Consider the node u and height h at which we enter the down phase. Intuitively, we show that sufficient weight supports either the link into which u is originally entered during the up phase or the link out of which u is exited during the down phase.

Define $j' = j$ if $j \in X$ and $j' = j^+$ if $j \notin X$. The total search time is 1 plus $O(b)$ per each of $h - \min(r_i, h_{j'}) \leq h - \min(r_i, r_{j'})$ levels. We need only show that $V(i, j') = a^{\Omega(h)}$, which, together with the definition of rank, proves the theorem.

In the case $u > i$, consider the first height $h' \leq h$ at which u is entered during the up phase. If $h' = h$, then $V(i, u) \geq V(L_h^u, u)$, which by Lemma 3.8 is at least a^h. Otherwise, $h' < h$, and hence $V(u, j') \geq V(u, R_{h-1}^u)$, which by Lemma 3.8 is at least a^{h-1}. Since $V(i, j') \geq V(i, u)$ and $V(i, j') \geq V(u, j')$, either subcase yields $V(i, j') = a^{\Omega(h)}$.

In the remaining case, $u = i$. If $h = r_u$, then $V(i, j') \geq w_i \geq a^h$. If $h > r_u$, then $V(i, j') \geq V(i, R_{h-1}^i)$, which by Lemma 3.8 is at least a^{h-1}. Again $V(i, j') = a^{\Omega(h)}$. □

Note that we start the finger search at height r_i, not h_i, which enables the proof to work in case the search starts the down phase immediately.

3.6 Changing the Weight of an Item

Finally, we can change the weight of an item i from w_i to w'_i without fully deleting and reinserting i. Denote the new rank of i by r'_i.

Procedure Reweight(S, i, w'_i)

1. Search S to find node i.

2. If $r'_i = r_i$, then stop.

3. If $r'_i > r_i$, then do nothing if $h_i \geq r'_i$. Otherwise, promote i to height r'_i; apply Step 3 from Procedure **Insert**(S, i) but starting at height $h_i + 1$; and apply Step 4 from Procedure **Insert**(S, i) starting at height r'_i.

4. If $r'_i < r_i$, then demote i to height r'_i; apply Step 3 from Procedure **Delete**(S, i) but starting at height r'_i; and apply Step 4 from Procedure **Delete**(S, i) starting at the least $j \in PL(i)$ greater than r'_i.

Lemma 3.10 *Invariants **(I1)** and **(I2)** are true after* **Reweight**(S, i, w'_i).

Proof. Assume the invariants were true before the reweight. If $r'_i = r_i$, then neither invariant can be violated. If $r'_i > r_i$, then if $h_i \geq r'_i$, again neither invariant can be violated. Otherwise, if $r'_i > h_i \geq r_i$, i must attain height at least r'_i, in which case **(I2)** can be violated between levels $h_i + 1$ and r'_i, and **(I1)** can be violated at level r'_i. Applying the demotion and promotion steps from the insert procedure fixes the violations as shown in the proof of Lemma 3.1.

Finally, if $r'_i < r_i$, **(I2)** can be violated between levels $r'_i + 1$ and r_i. Demoting i to height r'_i and then applying the promotion and deletion procedures from the delete procedure fixes the violations as shown in the proof of Lemma 3.3. □

Theorem 3.11 *Changing the weight of any node i in an (a, b)-biased skip list from w_i to w'_i takes $O\left(1 + b \log_a \frac{W + w'_i}{\min(w_i, w'_i)}\right)$ time.*

Proof. Lemma 3.10 proves correctness of **Reweight**(S, i, w'_i). Step 1 takes $O\left(1 + b \log_a \frac{W}{w_i}\right)$ time by the Access Lemma. (We assume $i \in X$.) Height promotions and demotions in Steps 3 and 4 perform $O(b)$ work per level and occur no lower than heights $h_i + 1$ and r'_i respectively. Applying the Depth Lemma completes the proof. □

4 Randomized Updates

We can randomize the biased skip list structure presented in Section 3 to yield expected optimal access times without the need for promotions or demotions. Mehlhorn and Näher [17] suggested the following approach but claimed only that the expected maximal height of a node is $\log W + O(1)$. We will show that the expected depth of a node i is $E[d_i] = O\left(\log \frac{W}{w_i}\right)$.

A *randomized biased skip list* S is parameterized by a positive constant $0 < p < 1$. Here we define the *rank* of an item i as $r_i = \lfloor \log_{\frac{1}{p}} w_i \rfloor$. When inserting i into S, we assign its height to be $h_i = r_i + e_i$ with probability $p^{e_i}(1 - p)$ for e_i a non-negative integer, which we call the *excess height* of i. Algorithmically, we start node i at height r_i and continually increment the height by one as long as a biased coin flip returns heads (with probability p).

Reweight is the only operation that changes the height of a node. The new height is chosen as for insertion but based on the new weight, and the tower is adjusted appropriately. The remaining operations (insert, delete, join, split, and (finger) search) perform no rebalancing.

Lemma 4.1 (Randomized Height Lemma) *The expected height of any item i in a randomized, biased skip list is* $\log_{\frac{1}{p}} w_i + O(1)$.

Proof.

$$\begin{aligned} E[h_i] &= r_i + E[e_i] \\ &= r_i + \sum_{j=0}^{\infty} j p^j (1-p) \\ &= r_i + \frac{p}{1-p} \\ &= \lfloor \log_{\frac{1}{p}} w_i \rfloor + O(1). \end{aligned}$$

□

The proof of the Depth Lemma for the randomized structure follows that for the deterministic structure.

Recall the definitions $N_i = |\{x : r_x = i\}|$; $N'_i = |\{x : r_x \leq i \wedge h_x \geq i\}|$; and $W_i = \sum_{r_x \leq i} w_x$.

Lemma 4.2 $E[N'_i] = \sum_{j=0}^{i} p^{i-j} N_j$.

Proof. We prove the lemma by induction. The base case, $N'_0 = N_0$, is true by definition. Since the excess heights are i.i.d. random variables, we have, for $i > 0$, $E[N'_{i+1}] = N_{i+1} + pE[N'_i]$, which together with the induction hypothesis proves the lemma. □

Corollary 4.3 $E[N'_i] \leq p^i W_i$.

Proof. By definition,

$$\begin{aligned} W_i &\geq \sum_{j=0}^{i} \frac{1}{p^j} N_j \\ &= \frac{1}{p^i} \sum_{j=0}^{i} p^{i-j} N_j. \end{aligned}$$

Lemma 4.2 yields the result. □

Lemma 4.4 (Randomized Depth Lemma) *The expected depth of any node i in a randomized, biased skip list S is* $O\left(\log_{\frac{1}{p}} \frac{W}{w_i}\right)$.

Proof. The depth of i is $d_i = H(S) - h_i$. As before, define $R = \max_{x \in X} r_x$. By standard skip list analysis [19], we know that

$$\begin{aligned} E[H(S)] &= R + O(E[\log_{\frac{1}{p}} N'_R]) \\ &\leq R + cE[\log_{\frac{1}{p}} N'_R] \text{ for some constant } c \\ &\leq R + c \log_{\frac{1}{p}} E[N'_R] \text{ by Jensen's inequality} \\ &\leq R + c\left(\log_{\frac{1}{p}} W_R - R\right) \text{ by Corollary 4.3} \\ &= c \log_{\frac{1}{p}} W - (c-1)R. \end{aligned}$$

By the Randomized Height Lemma, therefore, $E[d_i] \leq c \log_{\frac{1}{p}} W - (c-1)R - \log_{\frac{1}{p}} w_i$. The lemma follows by observing that $R \geq \lfloor \log_{\frac{1}{p}} w_i \rfloor$. □

Corollary 4.5 (Randomized Access Lemma) *The expected access time for any key i in a randomized, biased skip list is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W}{w_i}\right)$ if $i \in X$ and $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W}{\min(w_{i^-}, w_{i^+})}\right)$ if $i \notin X$.*

Proof. As $n \to \infty$, the probability that a plateau starting at any given node is of size k is $p(1-p)^{k-1}$. The expected size of any plateau is thus $1/p$. Applying the Randomized Depth Lemma completes the proof. □

The operations discussed in Section 3 become simple to implement.

Insert(S, i). Locate i^- and i^+ and create a new node between them, as described above, to hold i. The expected time is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W + w_i}{\min(w_{i^-}, w_i, w_{i^+})}\right)$.

Delete(S, i). Locate and remove node i. The expected time is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W}{\min(w_{i^-}, w_i, w_{i^+})}\right)$. The Randomized Depth and Access Lemmas continue to hold, because S is as if i had never been inserted in the first place.

Join(S_L, S_R). Trace through the profiles of $L_{\max}$ and $R_{\min}$ to splice the pointers leaving S_L together with those going into S_R. The expected time is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W_L}{w_{L_{\max}}} + \frac{1}{p} \log_{\frac{1}{p}} \frac{W_R}{w_{R_{\min}}}\right)$.

Split(S, i). (Assume $i \notin X$. An equivalent formulation holds when $i \in X$.) Disconnect the pointers that join the left profile of i^- to the right profile of i^+. The expected time is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W}{\min(w_{i^-}, w_{i^+})}\right)$.

FingerSearch(S, i, j). Perform **FingerSearch(S, i, j)** as described in Section 3.5. It is straightforward to prove that Lemma 3.8 holds in the expected case. The expected time is thus $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{V(i,j)}{\min(w_i, w_j)}\right)$ if $j \in X$ and $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{V(i,j^+)}{\min(w_i, w_{j^-}, w_{j^+})}\right)$ if $j \notin X$.

Reweight(S, i, w'_i). Reconstruct the tower for node i as described above. The expected time is $O\left(1 + \frac{1}{p} \log_{\frac{1}{p}} \frac{W + w'_i}{\min(w_i, w'_i)}\right)$.

5 Conclusion

Open is whether a deterministic biased skip list can be devised that has not only the worst-case times that we provide but also an amortized bound of $O(\log w_i)$ for *updating* node i; i.e., once the location of the update is discovered, inserting or deleting should take $O(\log w_i)$ amortized time.

The following counterexample demonstrates that our initial method of promotion and demotion does not yield this amortized bound. Consider a node i such that $h_i - r_i$ is large and, moreover, that separates two plateaus of size $b/2$ at each level j between $r_i + 1$ and h_i and two plateaus of size $b/2$ and $b/2 + 1$, rsp., at level r_i. Deleting i will cause a promotion starting at level r_i that will percolate to level h_i. Reinserting i with weight a^{r_i} will restore the structural condition before the deletion of i. This sequence of two operations can be repeated infinitely often; since $h_i - r_i$ is arbitrary, the cost of restoring the invariants cannot be amortized.

An initial attempt to address this bad case is to generalize the promotion operation to split a plateau of size exceeding b into several plateaus of size about b/η each, for some suitable constant η. Above, $\eta = 2$. The counterexample generalizes, however. Consider $\eta - 1$ nodes of some equal height, h, which exceed their (equal) rank, r, and which separate plateaus of size b/η at each height from r through h, except for one plateau, P, at height r, which has size $b/\eta + 1$. Deleting and reinserting one of the separating towers that delineates P restores the structure.

References

[1] N. Abramson. *Information Theory and Coding*. McGraw-Hill, New York, 1963.

[2] G. M. Adel'son-Vel'skii and Y. M. Landis. An algorithm for the organisation of information. *Doklady Akademii Nauk SSSR*, 146:263–6, 1962. English translation in *Soviet Math. Dokl.* **3**:1259-62, 1962.

[3] J. Bell and G. Gupta. Evaluation of self-adjusting binary search tree techniques. *Software Practice and Experience*, 23(4):369–382, 1993.

[4] S. W. Bent, D. D. Sleator, and R. E. Tarjan. Biased search trees. *SIAM Journal on Computing*, 14(3):545–68, 1985.

[5] Y.-J. Chiang and R. Tamassia. Dynamization of the trapezoid method for planar point location in monotone subdivisions. *International Journal of Computational Geometry and Applications*, 2(3):311–333, 1992.

[6] E. Cohen and H. Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proc. 2001 Symp. on Applications and the Internet (SAINT)*, pages 85–92. IEEE, 2001.

[7] F. Ergun, S. C. Sahinalp, J. Sharp, and R. K. Sinha. Biased dictionaries with fast inserts/deletes. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 483–91, 2001.

[8] F. Ergun, S. C. Sahinalp, J. Sharp, and R. K. Sinha. Biased skip lists for highly skewed access patterns. In *Proc. 3rd Wks. on Algorithm Engineering and Experiments*, volume 2153 of *Lecture Notes in Computer Science*, pages 216–29. Springer, 2001.

[9] J. Feigenbaum and R. E. Tarjan. Two new kinds of biased search trees. *Bell System Technical Journal*, 62(10):3139–58, 1983.

[10] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *Journal of Algorithms*, 23:51–73, 1997.

[11] S. D. Gribble and E. A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proc. 1st USENIX Symp. on Internet Technologies and Systems*, 1997.

[12] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symp. on Foundations of Computer Science*, pages 8–21, 1978.

[13] S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17:157–84, 1982.

[14] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[15] D. E. Knuth. *The Art of Computer Programming*, volume 3: *Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[16] C. Martínez and S. Roura. Optimal and nearly optimal static weighted skip lists. Technical Report LSI-95-34-R, Dept. Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 1995. http://www.lsi.upc.es/∼roura/techreps.html.

[17] K. Mehlhorn and S. Näher. Algorithm design and software libraries: Recent developments in the LEDA project. In *Proc. IFIP 12th World Computer Congress*, volume 1, pages 493–505. Elsevier, 1992.

[18] J. I. Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 367–75, 1992.

[19] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–76, June 1990.

[20] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–97, 1996.

[21] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–91, 1983.

[22] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–86, 1985.

[23] W. Stallings. *Operating Systems: Internals and Design Principles*. Prentice-Hall, 4th edition, 2001.