On Randomized Online Labeling with Polynomially Many Labels*

Jan Bulánek**[1,2], Michal Koucký***[2], and Michael Saks†[3]

[1] Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague
[2] Institute of Mathematics, Academy of Sciences CR, Prague
[3] Department of Mathematics, Rutgers University

Abstract. We prove an optimal lower bound on the complexity of *randomized* algorithms for the online labeling problem with polynomially many labels. All previous work on this problem (both upper and lower bounds) only applied to deterministic algorithms, so this is the first paper addressing the (im)possibility of faster randomized algorithms. Our lower bound $\Omega(n \log(n))$ matches the complexity of a known deterministic algorithm for this setting of parameters so it is asymptotically optimal.
In the online labeling problem with parameters n and m we are presented with a sequence of n *items* from a totally ordered universe U and must assign each arriving item a label from the label set $\{1, 2, \ldots, m\}$ so that the order of labels (strictly) respects the ordering on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

1 Introduction

In the online labeling problem with parameters n, m, r, we are presented with a sequence of n *items* from a totally ordered universe U of size r and must assign each arriving item a label from the label set $\{1, 2, \ldots, m\}$ so that the order of labels (strictly) respects the ordering on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which

* Full version of the paper is available from the second author's homepage.
** Partially supported by student project GAUK 344711 and RVO: 67985840.
*** Supported in part by grant IAA100190902 of GA AV ČR, Center of Excellence CE-ITI (P202/12/G061 of GA ČR) and RVO: 67985840.
† The work of this author was done while on sabbatical at Princeton University and was also supported in part by NSF under grant CCF-0832787 and CCF-1218711.

the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

The problem, which was introduced by Itai, Konheim and Rodeh [13], is natural and intuitively appealing, and has had applications to the design of data structures (see for example the discussion in [10], and the more recent work on cache-oblivious data structures [4, 8, 5]). A connection between this problem and distributed resource allocation was recently shown by Emek and Korman [12].

The parameter m, the size of the *label space* must be at least the number of items n or else no valid labeling is possible. There are two natural ranges of parameters which have received the most attention. In the case of *linearly many labels* we have $m = cn$ for some $c > 1$, and in the case of *polynomially many labels* we have $m = \theta(n^C)$ for some constant $C > 1$. The size r of the universe U is also a parameter which is not discussed explicitly in most of the literature on the paper. If $r \leq m$, since then we can simply fix an order preserving bijection from U to $\{1, \ldots, m\}$ in advance. In this paper we assume $U = \{1, \ldots, 2^n\}$.

Itai et al. [13] gave an algorithm for the case of linearly many labels having worst case total cost $O(n \log(n)^2)$. Improvements and simplifications were given by Willard [15] and Bender et al. [3]. In the special case that $m = n$, algorithms with cost $O(\log(n)^3)$ per item are known [16, 6]. It is also well known that the algorithm of Itai et al. can be adapted to give total cost $O(n \log(n))$ in the case of polynomially many labels. All of these algorithms are deterministic.

In a previous paper [9], we proved a $\Omega(n \log(n)^2)$ lower bound in the case of linearly many labels, and $\Omega(n \log(n)^3)$ lower bound for the case $m = n$. In subsequent work with Babka and Čunát [2] we proved a lower bound $\Omega(n \log(n)/(\log\log(m) - \log\log(n)))$ when $n^{1+\varepsilon} \leq m \leq 2^{n^\varepsilon}$, In particular, this gives a $\Omega(n \log(n))$ bound for the case of m being polynomial in n. These lower bounds match the known upper bounds to within a constant factor. Both of these papers built heavily on previous partial results of Dietz, Seiferas and Zhang ([16, 11, 10]). These lower bounds apply only to *deterministic* algorithms, leaving open the possibility of better randomized algorithms.

In this paper we use a model in which the cost of a randomized labeling algorithm is the worst case over all input sequences of a given length n of the expected number of moves made by the algorithm. This corresponds to running the algorithm against an *oblivious adversary* (see [7]) who selects the input sequence having full knowledge of the algorithm, but not of the random bits flipped in the execution of the algorithm.

There are many online problems where randomized algorithms perform provably better than deterministic ones. For example, the best deterministic algorithm for the paging problem with k pages has competitive ratio k but there are randomized algorithms having competitive ratio $\Theta(\log(k))$ [7].

Our Results. In this paper we establish the first lower bound for *randomized* online labeling algorithms by showing that in the case of polynomially many labels any randomized online labeling algorithm will have expected cost $\Omega(n \log(n))$ (for the worst case input). This matches the known deterministic upper bounds up to constant factors, and thus randomization provides no more

than a constant factor advantage over determinism. Our bound also implies an $\Omega(n \log(n))$ lower bound on the message complexity of randomized protocols for Distributed Controller Problem [12, 1].

Unlike many other lower bounds for non-uniform computation models, our proof does not use Yao's principle. Yao's principle says (roughly) that to prove a lower bound on the expected cost of an arbitrary randomized algorithm it suffices to fix a distribution over inputs, and prove a lower bound on the expected cost of a deterministic algorithm against the chosen distribution. Rather than use Yao's principle, our proof takes an arbitrary randomized algorithm and selects a (deterministic) sequence that is hard for that algorithm.

The construction and analysis of the hard sequence follow the same overall strategy of the previous lower bound for deterministic algorithms in the case of polynomially many labels [10, 2] which involves relating online labeling to a family of one-player games called *bucketing games* (introduced in [10]) which involve the sequential placement of items into an ordered sequence of bins subject to certain rules and costs. We define a map (an *adversary*) which associates to a labeling algorithm $\mathbf{A}$ a hard sequence of items. We then show that the behavior of the algorithm on this hard sequence can be associated to a strategy for playing a particular bucketing game, such that the cost incurred by the algorithm on the hard sequence is bounded below by the cost of the associated bucketing game strategy. Finally we prove a lower bound on the cost of any strategy for the bucketing game, which therefore gives a lower bound on the cost of the algorithm on the hard input sequence.

In extending this argument from the case of deterministic algorithms to the randomized case, each part of the proof requires significant changes. The adversary which associates an algorithm to a hard sequence requires various careful modifications. The argument that relates the cost of $\mathbf{A}$ on the hard sequence to the cost of an associated bucketing strategy does not work for the original version of the bucketing game, and we can only establish the connection to a new variant of the bucketing game called tail-bucketing. Finally the lower bound proof on the cost of any strategy for tail-bucketing is quite different from the previous lower bound for the original version of bucketing.

Mapping a randomized algorithm to a hard input sequence. We now give an overview of the adversary which maps an algorithm to a hard input sequence $y_1, \ldots, y_n$. The adversary is deterministic. Its behavior will be determined by the expected behavior of the randomized algorithm. Even though we are choosing the sequence obliviously, without seeing the actual responses of the algorithm, we view the selection of the sequence in an online manner. We design the sequence item by item. Having selected the first $t-1$ items, we use the known randomized algorithm to determine a probability distribution over the sequence of labelings determined by the algorithm after each step. We then use this probability distribution to determine the next item, which we select so as to ensure that the expected cost incurred by the algorithm is large.

The adversary will maintain a hierarchy consisting of a nested sequence subsets of the set of items inserted so far. The hierarchy at step t is used to determine

the item to be inserted at step p. This hierarchy is denoted

$$S_t(1) \supset T_t(2) \supset S_t(2) \supset T_t(3) \supset \cdots \supset T_t(d) \supset S_t(d).$$

The set $S_t(1)$ consists of all items inserted through step $t-1$. Each of the other sets is an interval relative to the items inserted so far, i.e., it consists of all inserted items in a given interval. The final subset $S_t(d)$ has between 2 and 6 elements. (Note that this differs from the previous work for deterministic algorithms where the hierarchy was a nested sequence of intervals of label values rather than items; this modification seems necessary to handle randomized algorithms). The next item to be inserted is selected to be an item that is between two items in the final set $S_t(d)$.

The hierarchy at step t is constructed as follows. The hierarchy for $t = 1$ has $d_1 = 1$ and $S_1(1) = \{0, 2^n\}$. The hierarchy at step $t > 1$ is constructed based on the hierarchy at the previous step $t-1$ and the expected behavior of the algorithm on $y_1, \ldots, y_{t-1}$.

We build the sets for the hierarchy at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous hierarchy, with the addition of y_{t-1}) or *rebuilt*. To specify which intervals are preserved, we specify a *critical level for step* t, q_t which is at most the depth d_{t-1} of the previous hierarchy. We'll explain the choice of q_t below. At step t, the intervals $T_t(i)$ and $S_t(i)$ for $i \leq q_t$ are *preserved*, which means that it is obtained from the corresponding interval at step $t-1$ by simply adding y_{t-1}. The intervals $T_t(i)$ and $S_t(i)$ for $i \geq q_t$ are *rebuilt*. The rule for rebuilding the hierarchy for $i > q_t$ is defined by induction on i as follows: Given $S_t(i-1)$, $T_t(i)$ is defined to be either the first or second half of $S_t(i-1)$, depending on which of these sets is more likely to have a smaller range of labels (based on the distribution over labels determined by the given algorithm). More precisely, we look at the median item of $S_t(i)$ and check whether (based on the randomized labeling) it is more likely that its label is closer to the label of the minimum or to the maximum element of $S_t(i)$. If the median is more likely to have label close to the minimum we pick the first half as $T_t(i)$ otherwise the second half. Having chosen $T_t(i)$, we take $S_t(i)$ to be the middle third of items in $T_t(i)$. This process terminates when $|S_t(i)| < 7$ and the depth d_t of the hierarchy is set to this final i. The adversary selects the next requested item y_t to be between two items in $S_t(d)$.

This construction of the hierarchy is similar to that used in [2] in the deterministic case. An important difference comes in the definition of the critical level q_t. In the deterministic case the critical level is the smallest index i such that neither endpoint of $T_{t-1}(i)$ was moved by the algorithm when inserting y_{t-1}. In the randomized case we need a probabilistic version of this: the critical level is the smallest index i such that the probability that either endpoint of $T(i)$ was moved since the last time it was rebuilt is less than $1/4$.

One of the crucial requirements in designing the adversary is that the hierarchy never grows too deep. Note that when we rebuild $T_t(i)$ it's size is at most $|S_t(i-1)|/2$ and when we rebuild $S_t(i)$ its size is at most $|T_t(i)|/3$. This suggests

that as we proceed through the hierarchy each set is at most 1/2 the size of the previous and so the depth is at most $\log(n)$. This is not true since during a sequence of steps in which a set in the hierarchy is not rebuilt its size grows by 1 at each step and so the condition that the set is at most half the size of its predecessor may not be preserved. Nevertheless we can show that the depth never grows to more than $4\log(m+1)$ levels.

A lower bound on the expected cost of the algorithm on the hard sequence. In [9] it is noted that we can assume without loss of generality that the algorithm is *lazy*, in the sense that for each step the set of relabeled items is a sub-interval of the inserted items that contains the most recently inserted item. (Intuitively, a non-lazy algorithm can be modified so that any relabeling that violates laziness is deferred until later). This observation extends to randomized algorithms.

In the deterministic case, this assumption and the definition of the critical level q_{t+1} can be used to show that when the algorithm responds to the item y_t it moved at least a constant fraction of the items belonging to $S_t(q_{t+1}+1)$ must have moved at step $t-1$ and so the total cost of the algorithm is at least $\Omega(\sum_t |S_t(q_{t+1}+1)|)$. In the randomized case we get a related bound that the expected total number of items is $\Omega(\sum_t |S_t(q_{t+1}) - S_t(q_{t+1}+1)|)$.

Bucketing games. The next step in the analysis is to define bucketing games, and to show that the lower bound on the cost of the algorithm given in the previous paragraph is an upper bound on the cost of an appropriate bucketing game.

The prefix bucketing game with n items and k buckets is a one player game. The game starts with k empty buckets indexed $1,\ldots,k$. At each step the player places an item in some bucket p. All the items from buckets $1,\ldots,p-1$ are then moved into bucket p as well, and the cost is the number of items in buckets $1,\ldots,p$ before the merge, which is the number of items in bucket p after the merge. The goal is to select the sequence of indices so as to minimize the total cost. The total cost is the sum of the costs of each step. The goal is to select the sequence of indexes p so that we would minimize the total cost. In [2] (following [10]) it is shown that any deterministic labeling algorithm could be associated to a bucketing strategy such that the cost of the labeling algorithm against our adversary is at least a constant times the cost of the bucketing strategy. This result is deduced using the lower bound of $\Omega(\sum_t |S_t(q_{t+1}+1)|$ for the cost of the algorithm mentioned earlier. It was also shown in [10] (see also [2]) that the minimal cost of any bucketing strategy (for more than $2\log(n)$ buckets) is $\Omega(n\log(n)/(\log(k)-\log\log(n))$. These results together gave the lower bound on deterministic labeling.

We use the same basic idea for the randomized case, but require several significant changes to the game. The first difficulty is that the lower bound $\Omega(\sum_t |S_t(q_{t+1}) - S_t(q_{t+1}+1)|)$ on the cost of a randomized algorithm in terms of a sum given earlier is not the same as the lower bound we had for deterministic algorithms. This lower bound is no longer bounded below by the cost of prefix bucketing. To relate this lower bound to bucketing, we must replace the cost

function in bucketing by a smaller cost function, which is the number of items in the bucket p *before* the merge, not after. In general, this cost function is less expensive (often much less expensive) than the original cost function and we call it the *cheap* cost function. The argument relating the cost of a randomized algorithm to a bucketing strategy requires that the number of buckets be at least $4\log(m)$ buckets. If we could prove a lower bound on the cost of bucketing under the cheap function similar to the bound mentioned above for the original function this would be enough to deduce the desired lower bound on randomized labeling. However with this cheap cost function this lower bound fails: if the number of buckets is at least $1 + \log(n)$, there is a bucketing strategy that costs 0 with the cheap cost function! So this will not give any lower bound on cost of a randomized labeling algorithm

We overcome this problem by observing that we may make a further modification of the rules for bucketing and still preserve the connection between the cost of a randomized algorithm against our adversary and the cheap cost of a bucketing. This modification is called *tail bucketing*. In a tail bucketing, after merging all the items into the bucket p, we redistribute these items back among buckets $1, \ldots, p$, so that bucket p keeps $1 - \beta$ fraction of the items and passes the rest to the bucket $p - 1$, bucket $p - 1$ does the same, and the process continues down until bucket 1 which keeps the remaining items. It turns our that our adversary can be related to tail bucketing for $\beta = 1/6$. We can prove that the minimal *cheap* cost of tail bucketing is $\Omega(n \log(n))$ when $k = O(\log n)$. This lower bound is asymptotically optimal and yields a similar bound for randomized online labeling.

The lower bound proof for the cheap cost of tail bucketing has some interesting twists. The proof consists of several reductions between different versions of bucketing. The reductions show that we can lower bound the cheap cost of tail bucketing with $C \log(n)$ buckets (for any C) by the cheap cost of ordinary prefix bucketing with $k = \frac{1}{4}\log n$ buckets. Even though the cheap cost of ordinary bucketing dropped to 0 once $k = \log(n) + 1$, we are able to show that for $k = \frac{1}{4}\log(n)$ there is a $\theta(n \log(n))$ bound for ordinary bucketing with the cheap cost.

Due to space limitations large portion of the proofs are omitted and they will appear in the full version. Unless otherwise specified, logarithms in this paper are to base 2.

2 The Online Labeling Problem

We first define the deterministic version of online labeling. We have parameters $n \leq m < r$, and are given a sequence of n numbers from the set $U = [1, r]$ and must assign to each of them a label in the range $[1, m]$. (Here, and throughout the paper, interval notation is used for consecutive sets of integers). A *deterministic* online labeling algorithm A with parameters (n, m, r) is an algorithm that on input sequence $(y_1, y_2, \ldots, y_t)$ with $t \leq n$ of distinct elements from U outputs a *labeling* $f_A : \{y_1, y_2, \ldots, y_t\} \to [m]$ that respects the natural ordering

of $y_1, \ldots, y_t$, that is for any $x, y \in \{y_1, y_2, \ldots, y_t\}$, $f_A(x) < f_A(y)$ if and only if $x < y$. We refer to $y_1, y_2, \ldots, y_t$ as *items*.

Fix an algorithm A. Any item sequence $y_1, \ldots, y_n$ determines a sequence $f_{A,0}, f_{A,1}, \ldots, f_{A,n}$ of labelings where f_t is the labeling of $(y_1, \ldots, y_t)$ determined by A. When the algorithm A is fixed we omit the subscript A. We say that A *relabels* $y \in \{y_1, y_2, \ldots, y_t\}$ *at time* t if $f_{t-1}(y) \neq f_t(y)$. In particular, y_t is relabeled at time t. $Rel_t = Rel_{A,t}$ denotes the set of items relabeled at step t. The cost of A on $y_1, y_2, \ldots, y_n$ is $\chi_A(y_1, \ldots, y_n) = \sum_{t=1}^{n} |Rel_t|$.

A *randomized* online labeling algorithm $\mathbf{A}$ is a probability distribution on deterministic online labeling algorithms. Given a item sequence $y_1, \ldots, y_n$, the algorithm $\mathbf{A}$ determines a probability distribution over sequences of labelings $f_0, \ldots, f_n$. The set Rel_t is a random variable whose value is a subset of $y_1, \ldots, y_t$. The cost of $\mathbf{A}$ on $y_1, y_2, \ldots, y_n \in U$ is the expected cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n) = \mathbf{E}[\chi_A(y_1, \ldots, y_n)]$. The maximum cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n)$ over all sequences $y_1, \ldots, y_n$ is denoted $\chi_{\mathbf{A}}(n)$. We write $\chi_m(n)$ for the smallest cost $\chi_{\mathbf{A}}(n)$ that can be achieved by any algorithm $\mathbf{A}$ with range m.

We state our main theorem.

Theorem 1. *For any constant C_0, there are positive constants C_1 and C_2 so that the following holds. Let $\mathbf{A}$ be a randomized algorithm with parameters (n, m, r), where $n \geq C_1$, $r \geq 2^n$ and $m \leq n^{C_0}$. Then $\chi_{\mathbf{A}}(n) \geq C_2 n \log(n)$.*

To prove the theorem we will need some additional definitions. Let $S \subseteq Y \subseteq U$. We write $\min(S)$ and $\max(S)$ for the least and greatest elements, respectively. We say that S is a Y*-interval* if $S = Y \cap [\min(S), \max(S)]$. We write $\mathrm{med}(S)$ for the *median* of S which we take to be the $\lceil |S|/2 \rceil$-th largest element of S. We define **left-half**$(S) = \{y \in S | y \leq \mathrm{med}(S)\}$ and **right-half**$(S) = \{y \in S | y \geq \mathrm{med}(S)\}$ (note that $\mathrm{med}(S)$ is contained in both). Also define **left-third**(S) to be the smallest $\lfloor |S|/3 \rfloor$ elements, **right-third**(S) to be the largest $\lfloor |S|/3 \rfloor$ elements and **middle-third**$(S) = S -$ **left-third**$(S) -$ **right-third**(S).

Given a labeling f of Y and a Y-interval S, we say that the Y-interval S is *left-leaning* with respect to f if $\mathrm{med}(S)$ has a label that is closer to the label of $\min(S)$ than it is to the label of $\max(S)$, i.e. $(f(\mathrm{med}(S)) - f(\min(S))) \leq (f(\max(S)) - f(\mathrm{med}(S))$. It is *right-leaning* otherwise.

A deterministic labeling algorithm is *lazy* if at each step t, the set of relabeled items is a Y_t-interval (which necessarily contains y_t), and a randomized algorithm is lazy if it is a distribution over lazy deterministic algorithms. In [9], it was shown that there is an optimal deterministic algorithm that is lazy, and the same proof works to show that there is an optimal lazy randomized algorithm. (Intuitively this is the case because if the relabeled items at step t do not form a Y_t-interval and W is the largest Y_t-interval of relabeled items containing y_t then we can defer relabeling the items in $Y_t \setminus W$ until later.)

2.1 The adversary

We now specify an adversary **Adversary(**$\mathbf{A}, n, m$**)** which given an online labeling algorithm $\mathbf{A}$, a length n, and label space size m, constructs a item sequence

$y_1, y_2, \ldots, y_n$ from the universe $U = \{1, \ldots, 2^n - 1\}$. Our adversary and notation borrow from past work in the deterministic case ([10, 9]).

We think of the adversary as selecting $y_1, \ldots, y_n$ online, but after each step the adversary only knows a probability distribution over the configurations of the algorithm.

To avoid dealing with special cases in the description of the adversary, we augment the set of items by items 0 and 2^n which are given (permanent) labels 0 and $m+1$ respectively. We write Y_t for the set $\{y_1, \ldots, y_t\} \cup \{0, 2^n\}$ of labeled items after step t. At the beginning of step t, having chosen the set Y_{t-1}, the adversary will select a Y_{t-1}-interval, denoted $S_t(*)$, of size at least 2 and select y_t to be $\min(S_t(*)) + 2^{n-t}$. An easy induction on t shows that the items chosen through step t are multiples of 2^{n-t}, and it follows that y_t is strictly between the smallest and second smallest elements of $S_t(*)$. Therefore all of the chosen items are distinct.

To choose $S_t(*)$, the adversary constructs a nested sequence of sets:

$$Y_{t-1} = S_t(1) \supset T_t(2) \supset S_t(2) \supset T_t(3) \supset \cdots \supset T_t(d_t) \supset S_t(d_t)$$

called the *hierarchy at step* t, and chooses $S_t(*) = S_t(d_t)$.

Note that the subscript for the hierarchy is one larger than the subscript of the containing set Y_{t-1} because the hierarchy is built in step t in order to determine y_t. Each set $S_t(i)$ and $T_t(i)$ is a Y_{t-1}-interval of size at least 2. The depth d_t of the hierarchy may vary with t. The sets $S_t(i)$ and $T_t(i)$ are said to be at *level* i in the hierarchy.

The hierarchy for $t = 1$ has $d_1 = 1$ and $S_1(1) = \{0, 2^n\}$. The hierarchy at step $t > 1$ is constructed based on the hierarchy at the previous step $t - 1$ and the expected behavior of the algorithm on $y_1, \ldots, y_{t-1}$ as reflected by the joint probability distribution over the sequence of functions $f_1, \ldots, f_{t-1}$.

The pseudo-code for the adversary is given in Figure 2.1, and follows the informal description of the adversary sketched in the introduction.

Here is a more detailed explanation of how the critical level q_t is selected. When constructing each set $S_t(i)$ of the hierarchy for $i \geq 2$, the adversary defines a parameter $\mathbf{birth}_t(i)$ which is set equal to t if $S_t(i)$ is rebuilt, and is otherwise set to $\mathbf{birth}_{t-1}(i)$. It is easy to see (by induction on t), that $\mathbf{birth}_t(i)$ is equal to the largest time $u \leq t$ such that $S_u(i)$ was rebuilt. It follows that for each $u \in [\mathbf{birth}_t(i), t]$, $\min(T_u(i)) = \min(T_t(i))$ and $\max(T_u(i)) = \max(T_t(i))$.

Say that item y has *stable label* during interval $[a, b]$ if for each step u in $[a, b]$, $f_u(y)$ has the same value, and has unstable label on $[a, b]$ otherwise. We define the event $\mathbf{stable}_t(i)$ to be the event (depending on $\mathbf{A}$) that $\min(T_t(i))$ and $\max(T_t(i))$ have stable labels during interval $[\mathbf{birth}_t(i-1), t]$.

We are finally ready to define q_t. If there is at least one level $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_{t-1}(i)] \leq 3/4$, let $i_{\min}$ be the least such level, and choose $q_t = i_{\min} - 1$. Otherwise set $q_t = d_{t-1}$.

The following lemma immediately implies Theorem 1.

Lemma 1. *Let $n \leq m$ be integers. Let $\mathbf{A}$ be a lazy randomized online labeling algorithm with the range m. Let $y_1, y_2, \ldots, y_n$ be the output of* $\mathbf{Adversary}(\mathbf{A}, n, m)$.

Then the cost satisfies:

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \geq \frac{5}{96}\left(\frac{1}{6}\right)^{2^8 c \log(2c)} (n+1)\log(n+1) - \frac{n}{4},$$

where $c = \log(m+1)/\log(n+1)$.

The proof of this lemma has two main steps. The first step is to bound the cost $\chi_{\mathbf{A}}(y_1, \ldots, y_n)$ from below by the minimum cost of a variant of the prefix-bucketing game. The prefix-bucketing game was introduced and studied before to get lower bounds for deterministic online labeling. The variant we consider is called *tail-bucketing*. The second step is to give a lower bound on the cost of tail-bucketing.

Adversary($\mathbf{A}, n, m$)
$t = 1$: $S_1(1) \longleftarrow \{0, 2^n\}$ and $\mathbf{birth}_1(1) = 1$, $y_1 = 2^{n-1}$.
For $t = 2, \ldots, n$ do

- $S_t(1) \longleftarrow S_{t-1}(1) \cup \{y_{t-1}\}$;
- (Choose critical level) Consider the sequence of (dependent) random functions $f_1, \ldots, f_{t-1}$ produced by $\mathbf{A}$ in response to $y_1, \ldots, y_{t-1}$. If there is an index $i \geq 2$ for which $\mathbf{Pr}[\mathbf{stable}_{t-1}(i)] \leq 3/4$, let $i_{\min}$ be the least such index and let $q_t = i_{\min} - 1$. Otherwise set $q_t = d_{t-1}$.
- $i \longleftarrow 1$.
- (Preserve intervals up to the critical level) While $i < q_t$ do:
  - $i \longleftarrow i + 1$.
  - $T_t(i) \longleftarrow T_{t-1}(i) \cup \{y_{t-1}\}$
  - $S_t(i) \longleftarrow S_{t-1}(i) \cup \{y_{t-1}\}$
  - $\mathbf{birth}_t(i) \longleftarrow \mathbf{birth}_{t-1}(i)$
- (Build intervals after the critical level) While $|S_t(i)| \geq 7$ do:
  - $i \longleftarrow i + 1$
  - If $S_t(i-1)$ is left-leaning with respect to f_{t-1} with probability at least $1/2$ then $T_t(i) \longleftarrow$ **left-half**$(S_t(i-1))$ otherwise $T_t(i) \longleftarrow$ **right-half**$(S_t(i-1))$
  - $S_t(i) \longleftarrow$ **middle-third**$(T_t(i))$.
  - $\mathbf{birth}_t(i) \longleftarrow t$. [Record that $S_t(i)$ and $T_t(i)$ were rebuilt]
- $d_t \longleftarrow i$.
- $y_t \longleftarrow \min(S_t(d_t)) + 2^{n-t}$.

Output: $y_1, y_2, \ldots, y_n$.

Fig. 1. Pseudocode for the adversary

To prove the first step we will need two properties of **Adversary($\mathbf{A}, n, m$)**. In what follows, we fix $\mathbf{A}, n, m$. **Adversary($\mathbf{A}, n, m$)** determines $y_1, \ldots, y_n$ and the critical levels $q_1, \ldots, q_n$. Note that the definition of q_j only depends on the expected behavior of the algorithm through the construction of f_{j-1}. For purposes of analysis, we also define the critical level q_{n+1} based on the expected behavior of $f_1, \ldots, f_n$ in exactly the same way.

Lemma 2. *For any $t \in [1, n]$, $d_t \leq 4 \log(m + 1)$.*

Lemma 3. *The cost of* **A** *on $y_1, \ldots, y_n$ satisfies:*

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \geq \frac{1}{40} \sum_t |S_t(q_{t+1}) \setminus S_t(1 + q_{t+1})|,$$

where the sum ranges over time steps $t \in [1, n]$ for which $q_{t+1} < d_t$.

3 Prefix bucketing and Tail bucketing

We will need several different variants of prefix bucketing game introduced by Dietz, Seiferas and Zhang [10]. We have k buckets numbered $1, \ldots, k$ in which items are placed. A *bucket configuration* is an arrangement of items in the buckets; formally it is a mapping $C : \{1, \ldots, k\}$ to the nonnegative integers, where $C(i)$ is the number of items in bucket i. It will sometimes be convenient to allow the range of the function C to be the nonnegative real numbers, which corresponds to allowing a bucket to contain a fraction of an item.

A *bucketing game* is a one player game in which the player is given a sequence of groups of items of sizes $n_1, \ldots, n_\ell$ and must sequentially place each group of items into a bucket. The case that the sequence $n_1 = \cdots = n_\ell = 1$ is called *simple bucketing*. The placement is done in a sequence of ℓ steps, and the player selects a sequence $p_1, \ldots, p_\ell \in [1, k]^\ell$, called an (ℓ, k)-*placement sequence* which specifies the bucket into which each group is placed. Bucketing games vary depending on two ingredients, the *rearrangement rule* and the *cost functions*.

When a group of m items is placed into bucket p, the items in the configuration are rearranged according to a specified rearrangement rule, which is not under the control of the player. Formally, a rearrangement rule is a function R that takes as input the current configuration C, the number m of new items being placed and the bucket p into which they are placed, and determines a new configuration $R(C, m, p)$ with the same total number of items.

The *prefix rearrangement rule* is as follows: all items currently in buckets below p are moved to bucket p. We say that items are *merged into bucket* p. Formally, the new configuration $C' = R(C, m, p)$ satisfies $C'(i) = 0$ for $i < p$, $C'(p) = C(1) + \cdots + C(p) + m$ and $C'(i) = C(i)$ for $i > p$. Most of the bucketing games we'll discuss use the prefix rearrangement function, but in Section 3.1 we'll need another rearrangement rule.

The *cost function* specifies a cost each time a placement is made. For the cost functions we consider the cost of placing a group depends on the current configuration C and the selected bucket p but not on the number m of items being placed. We consider four cost functions but for this extended abstract we need only the *cheap* cost function: In *cheap* bucketing, the cost is the number of items in bucket p before the placement:

$$\mathbf{cost_{cheap}}(C, p) = C(p).$$

Fix a rearrangement rule R and a cost function c. A placement sequence $p_1, \ldots, p_\ell$ and a load sequence $n_1, \ldots, n_\ell$ together determine a sequence of configurations $B = (B_0, B_1, \ldots, B_\ell)$, called a *bucketing* where B_0 is the empty configuration and for $i \in [1, \ell]$, $B_i = R(B_{i-1}, n_i, p_i)$. Each of these ℓ placements is charged a cost according to the cost rule c. We write $c[R](p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$ for the sum $\sum_{i=1}^{\ell} c(B_{i-1}, p_i)$, which is the sum of the costs of each of the ℓ rearrangements that are done during the bucketing. If R is the prefix rule, we call B a *prefix bucketing* and denote the cost simply by $c(p_1, \ldots, p_\ell | n_1, \ldots, n_\ell)$. In the case of simple bucketing, $n_1 = \ldots = n_\ell = 1$, we write simply $c[R](p_1, \ldots, p_\ell)$ or $c(p_1, \ldots, p_\ell)$ in the case of simple prefix bucketing.

3.1 Tail bucketing and online labeling

We will need an alternative rearrangement function, called the *tail rearrangement rule*. The bucketing game with this rule is called *tail bucketing*. The tail rearrangement rule $Tail_\beta$ with parameter β acts on configuration C, bucket p and group size m by first moving all items below bucket p to bucket p so that $w = C(1) + \cdots + C(p) + m$ items are in bucket p (as with the prefix rule), but then for j from p down to 1, β fraction of the items in bucket j are passed to bucket $j-1$, until we reach bucket 1. (Here we allow the number of items in a bucket to be non-integral.) Thus for $j \in [2, p]$ bucket j has $w(1-\beta)(\beta)^{p-j}$ items and bucket 1 has $w\beta^{p-1}$ items. We will consider tail bucketing with the cheap cost function.

We now relate the expected cost of randomized online labeling algorithm $\mathbf{A}$ on the sequence $y_1, y_2, \ldots, y_n$ produced by our adversary to the cost of a specific tail bucketing instance. For a lazy online labeling algorithm $\mathbf{A}$ and $t = 1, \ldots, n$, let $f_t, S_t(i), q_t, y_t$ be as defined by our adversary and the algorithm $\mathbf{A}$. Denote $Y = \{y_1, y_2, \ldots, y_n\}$. Set $k = \lfloor 4 \log(m+1) \rfloor$. Let $q_1, \ldots, q_n$ be the sequence of critical levels produced by the algorithm. As mentioned prior to stating Lemma 2, we define q_{n+1} for analysis purposes. For integer $i \in [k]$ define $\bar{i}$ to be $\bar{i} = (k+1) - i$. Define the placement sequence $p_1 = \bar{q}_2, \ldots, p_n = \bar{q}_{n+1}$, and consider the tail bucketing $B_0, \ldots, B_{n+1}$ determined by this placement sequence with parameter $\beta = 1/6$, and all group sizes 1 (so it is a simple bucketing). The following lemma relates the cost of online labeling to the tail bucketing.

Lemma 4. *Let $\{S_t(i) : 1 \le t \le n, 1 \le i \le d_t\}$ be the interval hierarchy computed by* **Adversary**$(\mathbf{A}, n, m)$ *and $B_{\mathbf{A}} = (B_0, \ldots, B_n)$ be the corresponding tail-bucketing. Then for any $t \in [1, n]$ and any $j \in [1, d_t]$:*

$$|S_t(j) \backslash S_t(j+1)| \ge B_{t-1}(\bar{j}) - 3.$$

Here, for the case $j = d_t$, we take $S_t(j+1)$ to be $\emptyset$.

Corollary 1. *The cost of randomized labeling algorithm $\mathbf{A}$ with label space $[1, m]$ on $y_1, \ldots, y_n$ satisfies:*

$$\chi_{\mathbf{A}}(y_1, y_2, \ldots, y_n) \ge \frac{1}{40}(\min \mathbf{cost}_{\mathbf{cheap}}[Tail_{1/6}](p_1, \ldots, p_n) - 10n),$$

where the minimum is over all placement sequences $(p_1, \ldots, p_n)$ *into* $\lfloor 4\log(m+1)\rfloor$ *buckets.*

Armed with Corollary 1, it now suffices to prove a lower bound on the cheap cost of simple tail bucketing when the number of items is n and the number of buckets is $\lfloor 4\log(m+1)\rfloor$. The lower bound is provided by the next statement:

Lemma 5. *Let* $p_1, \ldots, p_n$ *be an arbitrary placement sequence into* $\lfloor 4\log(m+1)\rfloor$ *buckets. Then*

$$\mathbf{cost_{cheap}}[Tail_{1/6}](p_1, \ldots, p_n) \geq \frac{5}{12}\left(\frac{1}{6}\right)^{2^8 c\log(2c)}(n+1)\log(n+1),$$

where $c = \log(m+1)/\log(n+1)$.

Now Lemma 1 follows from the above lemma and Corollary 1.

References

1. Afek, Y., Awerbuch, B., Plotkin, S., Saks, M.: Local management of a global resource in a communication network. *J. ACM*, 43(1), 1–19 (1996)
2. Babka, M., Bulánek, J., Čunát, V., Koucký, M., Saks, M.: On Online Labeling with Polynomially Many Labels. In *ESA*, 121–132 (2012)
3. Bender, M., Cole, R., Demaine, E., Farach-Colton, M., Zito, J.: Two simplified algorithms for maintaining order in a list. In *ESA*, 152–164 (2002)
4. Bender, M., Demaine, E., Farach-Colton, M.: Cache-oblivious B-trees. *SIAM J. Comput.*, 35(2), 341–358 (2005)
5. Bender, M., Duan, Z., Iacono, J., Wu, J.: A locality-preserving cache-oblivious dynamic dictionary. *J. Algorithms*, 53(2), 115–136 (2004)
6. Bird, R., Sadnicki, S.: Minimal on-line labelling. *Inf. Process. Lett.*, 101(1), 41–45 (2007)
7. http://www.stream.cz/sportovni-okamziky/811072-1936-cech-vitezi-nad-hitlerem Borodin, A., El-Yaniv, R.: *Online computation and competitive analysis*. Cambridge University Press, (1998)
8. Brodal, G., Fagerberg, R., Jacob, R.: Cache oblivious search trees via binary trees of small height. In *SODA*, 39–48, (2002)
9. Bulánek, J., Koucký, M., Saks, M.: Tight lower bounds for online labeling problem. In *STOC*, 1185–1198 (2012)
10. Dietz, P., Seiferas, J., Zhang, J.: A tight lower bound for online monotonic list labeling. *SIAM J. Discrete Math.*, 18(3), 626–637 (2004)
11. Dietz, P., Zhang, J.: Lower bounds for monotonic list labeling. In *SWAT*, 173–180 (1990)
12. Emek, Y., Korman, A.: New bounds for the controller problem. *Distributed Computing*, 24(3-4), 177–186 (2011)
13. Itai, A., Konheim, A., Rodeh, M.: A sparse table implementation of priority queues. In *ICALP*, 417–431 (1981)
14. Korman, A., Kutten, S.: Controller and estimator for dynamic networks. In *PODC*, 175–184 (2007)
15. Willard, D.: A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Inf. Comput.*, 97(2), 150–204 (1992)
16. Zhang, J.: Density Control and On-Line Labeling Problems. *PhD thesis*, University of Rochester (1993).