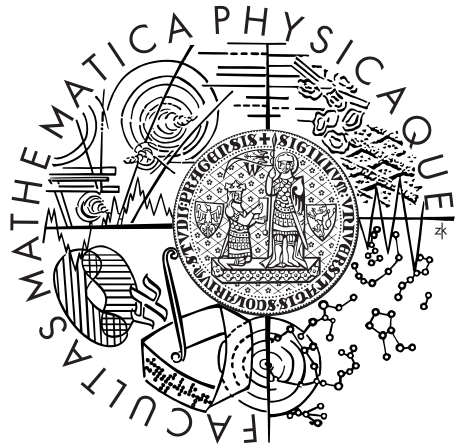
Charles University in Prague
Faculty of Mathematics and Physics

Abstract of Doctoral Thesis



Mgr. Jan Bulánek

The Online Labeling Problem

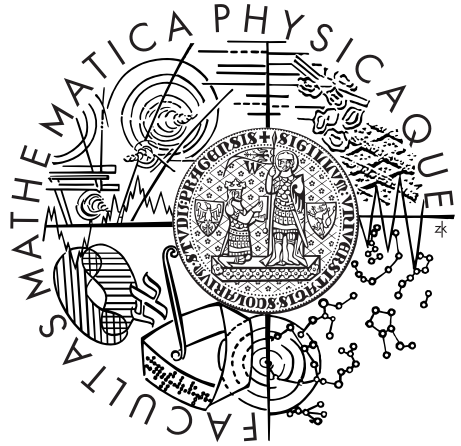
Department of Theoretical Computer Science
and Mathematical Logic,
and
Institute of Mathematics of the Academy of Sciences
of the Czech Republic

Supervisor: doc. Mgr. Michal Koucký, Ph.D.

2014

Univerzita Karlova v Praze
Matematicko-fyzikálni fakulta

Abstract doktorské práce



Mgr. Jan Bulánek

Problém Online Labelingu

Katedra teoretické informatiky a matematické logiky a
Matematický ústav Akademie věd České republiky

Vedoucí práce: doc. Mgr. Michal Koucký, Ph.D.

2014

Doktorská práce byla vypracována v rámci doktorského studia, které uchazeč absolvoval na teoretické informatiky a matematické logiky Matematicko-fyzikální fakulty Univerzity Karlovy v Praze v letech 2010–2014.

Uchazeč:	Mgr. Jan Bulánek Katedra teoretické informatiky a matematické logiky (MFF UK) a Matematický ústav Akademie věd České republiky jan.bulanek@gmail.com
Obor studia:	4I-1 – Teoretická informatika
Předseda RDSO:	prof. RNDr. Václav Koubek, DrSc. Katedra teoretické informatiky a matematické logiky (MFF UK) koubek@ktiml.mff.cuni.cz
Školitel:	doc. Mgr. Michal Koucký, Ph.D. Informatický ústav Univerzity Karlovy (MFF UK) koucky@iuuk.mff.cuni.cz
Oponenti:	Prof. Gerth Stølting Brodal Aarhus University, Denmark Department of Computer Science gerth@cs.au.dk
	Prof. John Iacono Polytechnic School of Engineering New York University, New York Department of Computer Science and Engineering iacono@nyu.edu

Autoreferát byl rozeslán dne 18. srpna 2014.

Obhajoba se koná dne 18. září 2014 v 10:00 hodin před komisí pro obhajoby disertačních prací v oboru 4I-1 – Teoretická informatika na Matematicko-fyzikální fakultě Univerzity Karlovy, Malostranské nám. 25, Praha 1.

S disertací je možné se seznámit na studijním oddělení MFF UK, Ke Karlovu 3, Praha 2

Contents

1 Introduction

The construction of effective algorithms and data structures is the central focus of computer science since its very beginning. As the amount of data we need to process increases this is an important topic even though the computational power of our devices is increasing. Therefore, we need to design the best algorithms or data structures possible. To do so we need to be able to recognize what is the best possible solution. Thus, we need to prove lower bounds on the best possible cost of algorithms and data structures.

In this thesis we do both. We design algorithms and prove lower bounds on their cost. For a particular problem called the *online labeling problem* we not only present a new algorithm which is superior to known algorithms for certain input sizes, but we also prove matching general lower bounds.

The online labeling problem is tightly connected to sorting. Sorting is no doubt one of the fundamental problems of computer science. We know optimal time complexity of sorting. Sorted arrays are easy to use, provide asymptotically optimal search time and utilize caches well during scan query.

A natural question is, whether inserts of additional items in a sorted array can be implemented efficiently. There are immediate applications of such a structure in algorithm and data structure design, e.g., dynamic sorted arrays, priority queues, dictionaries etc. First, we formulate the problem.

You are given an array of size m and a stream of n integers coming in arbitrary order where $n \leq m$. Your task is to maintain all already received items in the array in sorted order. The inserted items do not have to be stored consecutively in the array. Since the final order of the items is not known until we see all the items, moves of already inserted items are allowed but should be minimized. The obvious solution moves $O(n)$ items after each insert with the total complexity $O(n^2)$. This, however, does not benefit from the fact that the size of the array may be significantly larger than n.

This is one of several possible formulations of online labeling problem which is also known as the *file maintenance problem*. In the general formulation, items from a universe U of size r are assigned *labels* from a given linearly ordered universe of size m. The ordering of their labels must respect the ordering of items. Relabeling of items (which is equivalent to moving items in the array) is allowed, but should be minimized.

The first non-trivial algorithm for the online labeling problem was given at the beginning of the eighties by Itai, Konheim and Rodeh [17]. The amortized time complexity was $\Omega(\log^2(n))$ per inserted item, while the size of the array is only cn for any constant $c > 1$. This surprising result was achieved by a clever utilization

of empty space in the arrays. Since then, several other algorithms were designed, but it remained open whether whether any of them is asymptotically optimal.

In this thesis we resolve this question by proving lower bounds matching the complexity of these algorithms. Indeed, we prove optimal lower bounds for almost all possible array sizes, in the case of linear size arrays we prove such lower bounds even under the assumption of limited universe ($r \in O(n)$ while previous results assume $r \geq 2^n$), and we also prove the first lower bound for randomized algorithms for the online labeling problem.

Thus, we essentially completely determine the optimal complexity of deterministic algorithms for the online labeling problem. There are two possible further directions how to extend our results: Are there asymptotically better randomized algorithms for the online labeling problem? We know that in the case of polynomial size arrays the answer is negative. In the other direction one can ask whether it is possible to obtain better algorithms when the size of the universe from which the items are selected is comparable to the size of the array. We know that in the case of linear size arrays the answer is negative as well but what about larger arrays?

1.1 Applications

The very first algorithm for the online labeling problem was given by Itai, Konheim and Rodeh [17] who used it as a tool for an implementation of a priority queue. This is a natural application as one can see the online labeling problem as a dynamic maintenance of sorted array. Unfortunately, the best algorithm for the online labeling problem in arrays of linear size is worse than binary trees by a logarithmic factor which makes approach in [17] inferior to them.

However, in recent years there has been renewed interest in the online labeling problem problem because of its applications in the design of cache-oblivious algorithms, e.g., design of cache-oblivious B-trees [4, 9], cache-oblivious dynamic dictionaries [5], and packed arrays [6]. In these results, algorithms for the online labeling problem are used to implement buffers that utilize caches well.

Recently, Emek and Korman [15] established a connection between the online labeling and the distributed controller problem introduced in [1]. In distributed controller problem, nodes in an asynchronous distributed network receive requests from outside the network for units of a limited resource, and issue usage permits in response to the requests. The number of permits issued may not exceed the total resource supply, and the protocol must also ensure that no request is declined until the number of permits committed exceeds a $1 - \varepsilon$ fraction of the supply. Protocols with message complexity $O(n \log^2(n))$ on n-node networks are known

(e.g. [1, 19]). Emek and Korman showed that the online labeling problem can be reduced to distributed controller problem. They noted that, using their reduction, a matching $\Omega(n \log^2(n))$ lower bound would follow from an $\Omega(n \log^2(n))$ lower bound on the online labeling problem. We provide such lower bound thus implying the optimality of known upper bounds.

1.2 Formal Definition of Online Labeling Problem

First we define the deterministic version of online labeling problem. We have parameters $n \leq m < r$, and given a sequence of n numbers from the set $U = \{1, \ldots, r\}$ and must assign to each of them a label in the range $\{1, \ldots, m\}$. This is the only meaningful setting of parameters, as for $n > m$ we have insufficient number of labels for all inserted items and for $r \leq m$ the problem is trivial.

A *deterministic* online labeling algorithm $\mathcal{A}$ 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* $\mathbf{f}_{\mathcal{A}} : \{y_1, y_2, \ldots, y_t\} \to [m]$ that respects the natural ordering of $y_1, \ldots, y_t$, that is for any $x, y \in \{y_1, y_2, \ldots, y_t\}$, $\mathbf{f}_{\mathcal{A}}(x) < \mathbf{f}_{\mathcal{A}}(y)$ if and only if $x < y$. We refer to $y_1, y_2, \ldots, y_t$ as *items*. We write Y_t for the set $\{y_1, y_2, \ldots, y_t\}$.

Fix an algorithm $\mathcal{A}$. Any item sequence $y_1, \ldots, y_n$ determines a sequence $\mathbf{f}_{\mathcal{A},0}, \mathbf{f}_{\mathcal{A},1}, \ldots, \mathbf{f}_{\mathcal{A},n}$ of labelings (or allocations in case of file maintenance problem formulation) where $\mathbf{f}_{\mathcal{A},t}$ is the labeling of $(y_1, \ldots, y_t)$ determined by $\mathcal{A}$ immediately after y_t was presented. When the algorithm $\mathcal{A}$ is fixed we omit the subscript $\mathcal{A}$. We say that $\mathcal{A}$ *relabels* $y \in Y_t$ *at step* t if $\mathbf{f}_{t-1}(y) \neq \mathbf{f}_t(y)$. In particular, y_t is relabeled at step t. $\mathbf{Rel}_{\mathcal{A},t}$ denotes the set of items relabeled at step t. The cost of $\mathcal{A}$ on $y_1, y_2, \ldots, y_n$ is $\chi_{\mathcal{A}}((y_1, \ldots, y_n), m, r) = \sum_{t=1}^{n} |\mathbf{Rel}_t|$. If $r \geq 2^n - 1$ parameter r is omitted as it does not influence the cost of the algorithm.

A *randomized* online labeling algorithm $\mathcal{A}_r$ is a probability distribution on deterministic online labeling algorithms. Given an item sequence $y_1, \ldots, y_n$, the algorithm $\mathcal{A}_r$ determines a probability distribution over sequences of labelings $\mathbf{f}_0, \ldots, \mathbf{f}_n$. The set $\mathbf{Rel}_{\mathcal{A}_r,t}$ is a random variable whose value is a subset of $y_1, \ldots, y_t$. The cost of $\mathcal{A}_r$ on $y_1, y_2, \ldots, y_n \in U$ is the expected cost $\chi_{\mathcal{A}_r}((y_1, \ldots, y_n), m) = \mathbf{E}[\chi_{\mathcal{A}}((y_1, \ldots, y_n), m)]$.

The maximum cost $\chi_{\mathcal{A}_r}((y_1, \ldots, y_n), m)$ over all sequences $y_1, \ldots, y_n$ is denoted $\chi_{\mathcal{A}_r}(n, m)$. We write $\chi(n, m)$ for the smallest cost $\chi_{\mathcal{A}_r}(n, m)$ that can be achieved by any algorithm $\mathcal{A}_r$ with range m.

2 Online Labeling Problem Upper Bounds

We present two algorithms. The first one is optimal for arrays of small size, and the second one is optimal for arrays of large size. Together they cover almost all array sizes except for arrays of size $m \in [n^{\omega(1)}, n^{\log n})$ where no non-trivial upper bound is known.

A trivial algorithm can use the same approach as the Insertion sort [18]. First we determine a position of the new item. Then we move all items inserted so far which are behind its position by one cell for the asymptotic cost $O(n)$. Thus we obtain one empty cell into which the new item can be inserted.

Notice, that such approach does not take any advantage of arrays of size larger than n. To make better (or any) use of the additional space, our algorithms follow two basic concepts. First we maintain gaps (free space) between already inserted items and second we try to keep these gaps spread as evenly as possible.

Both our algorithms work in rounds. At the very beginning of the round the algorithm is provided with the next item y to be inserted. Then the algorithm finds out where y should be inserted. Then it finds the *suitable* part of the array (*segment*), which contains this point and finally it rearranges the items in this segment (including y) as evenly as possible.

2.1 Previous work

The very first non-trivial algorithm for solving the online labeling problem was given by Itai, Konheim and Rodeh [17] who developed this algorithm to solve the priority queue problem. They achieved $O(n \log^2(n))$ time complexity for arrays of size $O(n)$ in the amortized setting. This algorithm was simplified by Itai and Katriel [16]. However the analysis of the algorithm became more complicated. Another improvement was done by Willard [20], who achieved the same time complexity but in the worst case settings (i.e. each insert has time complexity at most $\log^2(n)$). His result was then simplified by Bender et al. [3]. In the special case of $m = n$, an algorithm with cost $O(n \log^3(n))$ in amortized settings was first developed by Zhang [21] and then it was simplified by Bird and Sadnicky [7]. This result is surprising since when you insert last few items the array contains almost no gaps.

It is also interesting that none of these works considers the case of limited universe. Obviously, if $r \leq m$ the online labeling problem is trivial. A natural question is whether for r say less than $10m$ the problem is significantly easier. However, Theorem 3.4 shows that in the case of linear array size you cannot benefit from the small universe.

2.2 Online Labeling Problem in Small Arrays

We present an algorithm $\mathcal{A}^{small}$ for the online labeling problem which achieves an asymptotically optimal time complexity $O(n \cdot \frac{\log^2(m)}{\log(m)-\log(n)})$ for arrays of size $m \in [\frac{4}{3}n, n^c]$ where c is a constant greater than one and its modification, algorithm $\mathcal{A}^{tiny}$, which achieves an asymptotically optimal time complexity $O(n \cdot \log^2(m) \log\left(\frac{m}{m-n}\right))$ for arrays of size $m \in [n, \frac{4}{3}n]$. This matches (up to a constant factor) the known lower bounds.

The algorithm $\mathcal{A}^{small}$ is a modification of the algorithm by Itai, Konheim and Rodeh [17]. However, unlike the original algorithm, $\mathcal{A}^{small}$ is optimal even for arrays larger than linear size. The existence of such modification was known as a folklore, however, to our knowledge it was not published before. Furthermore, we provide a more detailed analysis which handles rounding issues during the rearrange of items.

The algorithm $\mathcal{A}^{tiny}$ is a reformulation of the result by Zhang [21]. We iteratively use the $\mathcal{A}^{small}$ on the groups of items instead of items itself.

2.2.1 Algorithm Outline

Let us recall that each online labeling algorithm works in rounds. During each round the algorithm is given a next item y to be inserted, then it moves some of the already inserted items and finally it inserts y. Thus we only have to decide which items will be moved during each round. From the high level perspective we make this decision as follows. After we obtain y we determine where it should be inserted. Then we find the smallest segment of the array which contains this point and whose density of items in it is smaller than a certain threshold defined for segments of various sizes. Finally we rearrange items in this segment (including y) as evenly as possible.

The resulting algorithms have the following time complexities.

Theorem 2.1 (Main Theorem)**.** *Let n, m be integers such that $n < m$. Then there exist constants C_0 and C_1 and algorithms $\mathcal{A}^{small}$ and $\mathcal{A}^{tiny}$ such that*

1. $\chi_{\mathcal{A}^{small}}(n, m) \leq C_0 \cdot n \cdot \frac{\log^2(m)}{\log(m)-\log(n)}$ *if* $n < \frac{3}{4}m$, *and*
2. $\chi_{\mathcal{A}^{tiny}}(n, m) \leq C_1 \cdot n \cdot \log^2(m) \log\left(\frac{m}{m-n}\right)$ *otherwise.*

2.3 Online Labeling Problem in Large Arrays

Second algorithm we present achieves asymptotically optimal time complexity $O(n \cdot \frac{\log n}{\log\log m - \log\log n})$ for the label space $m \in [n^{\log n}, 2^n]$ where c is a constant greater than

one. This matches our lower bound, however, leaving a small gap for arrays of size $m \in [n^{\omega(1)}, n^{\log n})$ where no nontrivial upper bound is known.

Prior to our joint work with Michal Koucký and Michael Saks [10], to the best of our knowledge, no algorithm was published for this range of array sizes.

2.3.1 Algorithm Outline

First notice that we can store $\log(m)$ items for the cost of 1 per item into an array of size m. In such a case there is no need to relabel any item. This idea can be iterated further to insert $\frac{1}{4} \cdot \frac{\log^2(m)}{\log\log(m)}$ items for the cost of 2 per item. We proceed in k rounds. In each round we insert $\frac{1}{2} \cdot \log(m)$ items without any relabels. After the round we redistribute items inserted in this round as evenly as possible. After k rounds, every two items are at distance at least $\frac{2^k \cdot m}{\log^k(m)}$. So we can repeat the process $\frac{1}{2} \cdot \frac{\log(m)}{\log\log(m)}$ times while keeping the minimum distance at least $\sqrt{m}$. Our algorithm generalizes of this idea so we obtain an algorithm A^k with the following properties.

Theorem 2.2. *Let $m > 2^{16}$ and k be integers such that $k \leq 1/2\sqrt{\log m / \log\log m}$. Assume $n \leq \log^{k/3}(m)$. Then $\chi_{A^k}(n, m) \leq (2k-1)n$, i.e., there is an algorithm A^k that inserts n items into an array of size m with amortized cost of $2k-1$ per item.*

For n large enough, $m \geq n^{\log n}$ and $k = \frac{3\log n}{\log\log m}$, the assumptions of Theorem 2.2 are satisfied, so we get an algorithm which inserts n items into an array of size m with complexity $O(\frac{\log n}{\log\log m})$. Notice that for $m \geq n^{\log n}$, this is asymptotically the same as $O(\frac{\log n}{\log\log m - \log\log n})$.

3 Online Labeling Problem Lower Bounds

In the thesis, we present all lower bounds for the online labeling problem known to date. These results prove tight lower bounds for deterministic online labeling algorithms for all array sizes. In addition we provide the first (tight) lower bound for linear size arrays for the case when the universe U is small (recall that for the case $|U| < m$ the online labeling problem is trivial). This result implies that even for small universe size algorithms using arrays of linear size cannot perform asymptotically better than in the case of exponential size universe. Finally we present the first nontrivial lower bound for randomized algorithms, which is tight for arrays of polynomial size. Refer to Table 1 for the overview of bounds.

Array size (m)	Asymptotic bound	Note
$n < m \leq 2n$	$\Theta\big(n \log^2(n) \log(\frac{n}{m-n})\big)$	
$m = cn$, constant $c > 1$	$\Theta\big(n \log^2(n)\big)$	1)
$m = n^{1+o(1)}$	$\Theta\Big(\frac{n \log^2(n)}{1+\log m - \log n}\Big)$	
$m = n^C$, constant $C > 1$	$\Theta(n \log n)$	2)
$m \in [n^{1+\omega(1)}, n^{\log(n)})$	$\Omega\Big(\frac{n \log n}{1+\log\log m - \log\log n}\Big)$	3)
$m = n^{\Omega(\log(n))}$	$\Theta\Big(\frac{n \log n}{1+\log\log m - \log\log n}\Big)$	

1) Valid even for the case when the universe U is small.
2) Valid even for randomized algorithms.
3) We do not know a matching upper bound.

Table 1: Overview of results of Part II

Prior to our results, only little was known about lower bounds for the online labeling problem. There were only two results: the tight lower bound for polynomial size arrays (which of course applies also for smaller arrays, but provides non-tight bound) by Dietz et al. [13] and another result by Dietz et al. [12], that proves a tight lower bound for a *restricted* class of online labeling algorithms (so called *smooth* algorithms) in case of linear size arrays. Both of these results appear also in Ph.D. thesis by Zhang [21], which prior to our work was the most comprehensive source for the problem.

3.1 Overview of Results

First, we present the tight lower bound for deterministic algorithms using arrays of size from n^c ($c > 1$) to 2^n. Our proof extends and simplifies the result of Dietz at al. [13] (also in [21]) which is valid only for arrays of polynomial size

Second, we describe the first lower bound for randomized algorithms. In particular, this result is tight for polynomial size arrays. Since this bound is the same (up to a constant factor) as the one presented in Upper Bounds Part we can infer that at least for polynomial size arrays randomized algorithms are not asymptotically better in expectation than the deterministic ones.

The remaining results focus on arrays of almost linear size.

We present the first tight lower bound for the superlinear (but "subpolynomial") arrays. This result also proves the tight lower bounds for linear arrays which makes it seemingly superior to the next result. However, in this result we do not prove the tight bounds for array of size close to n and we do not consider the case of small item universe.

Finally, we prove the tight lower bound for linear arrays even in the case of arrays of size of n (recall that for such arrays we have an upper bound $O(n \log^3(n)))$ for arbitrary deterministic algorithms. The previous results by Dietz et al. [12, 21] consider only the restricted class of *smooth* algorithms. This restriction makes the construction of an adversary against the algorithms easy (as opposed to the case of arbitrary algorithms), still the rest of their analysis is non-trivial and contains useful ideas which we build on.

In addition this is the first result which considers the case of limited universe of items we insert. This is important question as we already mention that for the universe U of size $r < m$ the online labeling problem is trivial and it is not clear whether a small universe of size comparable to m cannot help one to develop better algorithms. However, we show that for linear size arrays the lower bound is asymptotically the same as for non-limited universe.

3.2 Online Labeling with Large Label Space

For the large arrays we prove an $\Omega\left(n \cdot \frac{\log(n)}{\log\log(m) - \log\log(n)}\right)$ lower bound on the number of moves for inserting n items. This lower bound holds for m between n and 2^n. Note that for polynomially many labels this bound simplifies to $\Omega(n \log n)$. This is tight except for $m \in [n^{\omega(1)}, n^{\log n})$, where we do not know a matching upper bound.

For the case of polynomially many labels, Dietz at al. [13] (also in [21]) proved a matching lower bound for the $O(n \log n)$ upper bound. Their result consists of two parts; a lower bound for a problem they call *prefix bucketing* and a reduction from prefix bucketing to online labeling. We provide a simpler, tighter and more general lower bound for prefix bucketing, which allows us to extend the lower bounds for online labeling to the case when the label space size is as large as 2^n.

Our initial study of the proof in [13] of the reduction from prefix bucketing to online labeling led us to think that there is a significant gap in their proof. We modified their proof and obtained a correct version of the reduction. In [2] (on which this result is based) we claimed that we were correcting an apparently significant gap in the previous proof; we made this claim after checking with one of the authors who agreed with it. Having done a more careful comparison of our final proof to theirs, we see that while the proof in [13] has some misleading

statements and missing details, it is essentially correct. We present the details of our modification in order to clarify the ambiguities that were present in [13].

The result for the large array sizes is summarized in the following theorem.

Theorem 3.1. *There are positive constants C_0, and C_1 so that the following holds. Let $\mathcal{A}$ be a deterministic algorithm with parameters (n, m, r), such that $C_0 \leq n \leq m \leq 2^n$ and $r \geq 2^n - 1$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_1 \cdot \frac{n \log n}{3 + \log\log m - \log\log n}$.*

Notice that this theorem implies a nontrivial lower bound even for the linearly sized array ($\Omega(n \log n)$), however, it is not tight.

3.2.1 Reducing Prefix Bucketing to Online Labeling

Dietz et al. [13] sketched a reduction from prefix bucketing to online labeling. In their reduction they describe an adversary for the labeling problem. They show that given any algorithm for online labeling, the behavior of the algorithm against the adversary can be used to construct a strategy for prefix bucketing. We extend and simplify their analysis.

The goal in constructing an adversary is to force any online algorithm to perform many relabelings during insertion of n items.

As a guide to picking each successive item, the adversary maintains a sequence (*chain*) of nested subintervals of already inserted items. The chain serves a dual purpose: the chain after step t is used by the adversary to select the item inserted at step $t + 1$, and the sequence of chains over time provides a way to lower bound the total cost incurred by the algorithm. Each successive interval in the chain has span at most half the previous interval, and its density is within a constant factor of the density of the previous interval. The chain ends with an interval containing between 2 and 7 items. The next item to be inserted is then chosen so that it is adjacent to the smallest item in the lowest interval of the chain.

Initially, and during the first 7 inserts, the chain consists of just the single interval containing at most 7 items. After each subsequent insert, the algorithm $\mathcal{A}$ specifies the label of the next item and (possibly) relabels some items. The adversary then updates its chain. For the chain immediately prior to the insert, the adversary specifies the *critical interval* to be the smallest interval of the chain such that its smallest and greatest items were not relabeled. Its index in the chain is then called critical level and is denoted by q_t. The new chain is then obtained as follows. We say that intervals with index at most q_t are *preserved* for step t which means that they are carried over from the previous chain, with the addition of y_t. Otherwise the intervals are *rebuilt*. Beginning from the critical interval the chain is extended as follows. Having chosen the interval I from the chain, define its *left buffer* to be the smallest 1/8 items of I, and its *right buffer* to be the greatest 1/8

items of I. Let I' be the interval obtained from I by deleting the left and right buffers. The successor interval of I in the chain is a subinterval of I' with the minimum span that contains exactly half (rounded down) of the items of I'. The chain ends when we reach an interval with at most seven items.

In [13], the authors show that there is always a *dense point*, which is a point with the property that every subsegment of the array containing it has density at least half the overall density of the label space. They use this as the basis for building the chain. We build a chain with similar properties using a simpler argument.

It remains to prove that the algorithm will make lot of relabels on the sequence of items produced by the adversary. Following Dietz et al. [13], we do this by relating online labeling to the prefix bucketing game. (Our definition of the game differs slightly from that in [13].)

A *prefix bucketing of n items into k buckets* (numbered 1 to k) is a one player game consisting of n steps. At the beginning of the game all the buckets are empty. In each step a new item arrives and the player selects an index $p \in \{1, \ldots, k\}$. The new item as well as all items in buckets $1, \ldots, p-1$ are moved into bucket p at a cost equal to the total number of items in bucket p after the move. The run of the game is therefore completely specified by the sequence $p_1, \ldots, p_n$, where p_t is the bin into which the player placed the new item at step t. The goal is to minimize the total cost of n steps of the game.

The lower bound on online labeling is obtained by the following correspondence. Consider the run of an arbitrary online labeling algorithm $\mathcal{A}$ against the given adversary. At each step t, the adversary determines a particular level p_t of the chain to be the critical level (which is always at most $k = \lceil \log(m+1) \rceil$). Consider the sequence $p_1, \ldots, p_n$ as a sequence of placements defining a prefix bucketing of n items into $k = \lceil \log(m+1) \rceil$ buckets. It turns out that the total cost of the prefix bucketing will be within a constant factor of the total number of relabelings performed by the online labeling algorithm. Hence, a lower bound on the cost of a prefix bucketing of n items into k buckets will imply a lower bound on the cost of the algorithm against our adversary.

The connection between the cost of $\mathcal{A}$ against the adversary, and the cost of the associated prefix bucketing is obtained as follows. Recall that we may assume that the algorithm is *lazy*. The cost of the bucketing merge step p_t at step t is at most the number of items in the critical interval, so to relate this to the cost incurred by the online labeling algorithm, it is enough to argue that at step t a constant fraction of the items in the critical interval were relabeled. This is done by arguing that for each successor (sub)interval of the critical interval, either all labels in its left buffer or all labels in its right buffer were reassigned, and the total

number of such items is a constant fraction of the items in the critical interval.

3.3 Randomized Online Labeling with Polynomially Many Labels

We prove an $\Omega(n \cdot \log(n))$ lower bound on the expected number of moves for inserting n items for *randomized* online labeling algorithms. This lower bound is valid for m between cn and n^c ($c > 1$) for any *randomized* online labeling algorithm. For $m = n^c$, this matches the known deterministic upper bounds up to constant factors, and thus randomization provides no more than a constant factor advantage over determinism.

Prior to our joint work with Michal Koucký and Michael Saks [11], all lower bound proofs considered only deterministic algorithms. There are however 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 $\Theta(\log(k))$ [8]. Thus it is a natural question whether there exists randomized algorithms which are (in expectation) better than deterministic.

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 [8]) who selects the input sequence having full knowledge of the algorithm, but not of the random bits flipped in the execution of the algorithm.

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 lower bound for deterministic algorithms in the case of polynomially many labels. This involves relating online labeling to a *bucketing games*. We define an *adversary* which associates to a labeling algorithm $\mathcal{A}$ 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 $\mathcal{A}$ 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.

The result for the randomized algorithms are summarized in the following theorem. Recall that randomized online labeling algorithm $\mathcal{A}$ is a probability distribution on deterministic online labeling algorithms. The cost $\chi_{\mathcal{A}}(n, m, r)$ is the expected cost of the algorithm sampled from this distribution.

Theorem 3.2. *For any constant C_0, there are positive constants C_1 and C_2 so that the following holds. Let $\mathcal{A}$ be a randomized algorithm with parameters (n, m, r), where $n \geq C_1$, $r \geq 2^n - 1$ and $m \leq n^{C_0}$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_2 n \log(n)$.*

3.3.1 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 labellings 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 an *interval chain* consisting of a nested sequence of intervals of items inserted so far. The chain serves a dual purpose: the chain after step t is used by the adversary to select the item inserted at step $t+1$, and the sequence of chains over time provides a way to lower bound the total (expected) cost incurred by the algorithm.

This chain is denoted[1]

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

[1]Recall, we use subscript to denote step and we use $(\cdot)$ notation to denote a particular coordinate of such a vector or sequence at that step.

The interval $I_t(1)$ equals $Y_t \cup \{\min_U, \max_U\}$, the final interval $I_t(d)$ has between 2 and 6 elements. The next item to be inserted is selected to be an item that is between two items in the final interval $I_t(d)$.

The chain at step t is constructed as follows. The chain for $t = 0$ has $\mathbf{depth}_0 = 1$ and $I_0(1) = \{\min_U, \max_U\}$. The chain at step $t \geq 1$ is constructed based on the chain at the previous step $t-1$ and the expected behavior of the algorithm on $y_1, \ldots, y_t$.

We build the intervals for the chain at step t in order of increasing level (i.e., decreasing size). Intervals are either *preserved* (carried over from the previous chain, with the addition of y_t) or *rebuilt*. To specify which intervals are preserved, we specify a *critical level for step* t, q_t which is at most the depth $\mathbf{depth}_{t-1}$ of the previous chain. We'll explain the choice of q_t below. At step t, the intervals $T_t(i)$ and $I_t(i)$ 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. The intervals $T_t(i)$ and $I_t(i)$ for $i \geq q_t$ are *rebuilt*. The rule for rebuilding the chain for $i > q_t$ is defined by induction on i as follows: Given $I_t(i-1)$, $T_t(i)$ is defined to be either the first or second half of $I_t(i-1)$, depending on which of these intervals 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 $I_t(i-1)$ 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 $I_t(i-1)$. 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 $I_t(i)$ to be the middle third of items in $T_t(i)$. This process terminates when $|I_t(i)| < 7$ and the depth $\mathbf{depth}_t$ of the chain is set to this final i. The adversary selects the next requested item y_{t+1} to be between two items in $I_t(\mathbf{depth}_t)$.

This construction of the chain is similar (except we do not have two types of segments) to that used 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 $I_t(i)$ was moved by the algorithm when inserting y_t. 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 $T_{t-1}(i)$ was moved since the last step when it was rebuilt is less than $1/4$.

One of the crucial requirements in designing the adversary is that the chain never grows too deep. Note that when we rebuild $T_t(i)$ it's size is at most $|I_t(i-1)|/2$ and when we rebuild $I_t(i)$ its size is at most $|T_t(i)|/3$. This suggests that as we proceed through the chain each interval is at most $1/2$ the size of the previous and so the depth is at most $\log(n)$. This reasoning is invalid because during a sequence of steps in which an interval in the chain is not rebuilt its size

grows by 1 at each step and so the condition that the interval 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.

In the deterministic case, the definition of the critical level q_t 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 $I_{t-1}(q_t)$ and so the total cost of the algorithm is at least $DLB = \Omega(\sum_t |I_{t-1}(q_t)|)$. In the randomized case we get a related bound that the expected total number of moves is $RLB = \Omega(\sum_t |I_{t-1}(q_t) \setminus I_{t-1}(q_t+1)|)$. So the cost incurred by the algorithm is related to the extent of changes in the chain.

3.3.2 Bucketing Game

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.

Recall, 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 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 the previous result 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 $\Omega(\sum_t |I_{t-1}(q_t)|)$ for the cost of the algorithm mentioned earlier. It was also shown 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 on the cost of the randomized algorithm stated earlier, RLB, is not the same as the lower bound DLB that was known for deterministic algorithms. While DLB was shown to be at least the minimal cost of the prefix bucketing, this is not true for RLB. To relate RLB 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! (For example a strategy which always picks p to be the smallest index of an empty bucket has cost zero; it emulates incrementing a binary counter.) So this will not give any lower bound on the 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.

Finally we prove an $\Omega\left(n \cdot \frac{\log^2(n)}{2+\log(m)-\log(n)}\right)$ lower bound on the number of moves for inserting n items. This lower bound is valid for m between cn ($c > 1$) and $n^{1+\varepsilon}$ ($\varepsilon < \frac{1}{16}$) for any deterministic online labeling algorithm, matching the known upper bound up to constant factors.

3.4 Online Labeling Problem with Small Label Space

This result is an extension of [10] obtained together with Martin Babka and Vladimír Čunát. It is based on the original simpler proof of [10]. In the next result we provide [10] which was the first lower bound for general algorithms in the linear space regime. Previously the only known general lower bound was by Dietz et al. [13] who proved an $\Omega(n \cdot \log(n))$ lower bound for polynomial size

arrays. Thus our bound is a significant improvement. For the case of $m = O(n)$, Dietz et al. [12, 21] proved an $\Omega(n \log^2(n))$ lower bound for the restricted case of so-called *smooth algorithms*, however, this did not give tight bound beyond the linear case.

We use ideas similar to next result, which provides lower bounds for the linear case and arrays of size close to n. It also gives lower bounds for the case of limited item universe. It would be possible to extend next result. However, the main ideas of the proof would disappear. Thus we present the lower bound proof for the superlinear arrays independently.

3.4.1 Hard Sequence Construction

In this section we sketch an adversary which given an algorithm $\mathcal{A}$, the number of inserted items n and the size of the array m (such that they satisfy some requirements we point out later) it produces an input sequence $y_1, y_2, \ldots, y_n$ that is costly for the algorithm $\mathcal{A}$.

As a guide to picking each successive item, the adversary maintains a sequence (*chain*) of nested intervals of already inserted items similar to previous results. There are however differences. First we start to build the chain after roughly $n/2$ items were inserted. The reason for this is that then we can fix the depth of the chain to be the same during the argument. The first $n/2$ items are therefore chosen almost arbitrarily, we only have to ensure that between any two inserted items, there remain enough unused items in the universe.

The other difference is that for this proof the density control is crucial. Ideally, we should build the chain so that the density of successive intervals of the chain is nondecreasing. This is easy to achieve if we can choose each successive subinterval arbitrarily. However we have to choose the subinterval so that it has significant buffers of items surrounding it. This allows us to charge the algorithm when it moves the boundaries of some interval of the chain (recall we may assume that algorithm is lazy). Unfortunately when the buffers are introduced it is infeasible to ensure increasing density. Thus we have to relax this condition so that the density does not decrease *too much*. Still the density decrease may be *too large*. Therefore we distinguish *good* and *bad* intervals of the chain (levels of the chain). We say that an interval is *good*, if it contains a large subinterval surrounded with large enough buffers and the density of the subinterval is close to the density of the whole interval. The interval is *bad* otherwise. We show that in the bad case we can find large subinterval whose density is significantly higher then the density of the whole interval. By very careful choice of "almost the same" and "significantly higher" we can build a chain which is suitable for a charging scheme which we describe later. In particular we will be able to limit the number of bad intervals

in the chain after inserting each item.

We will need to rebuild the chain after each insert. The basic idea is similar to previous results. We first determine so called *critical level* and then starting from this level we *rebuild* the chain while all intervals above the critical level are *preserved* (i.e. they are unchanged, except for the newly inserted item which is added to them). The choice of the critical level is however slightly tricky. Before a good interval is rebuilt we need to ensure that enough items were moved. This is easy to ascertain for intervals whose borders were crossed by the lazy algorithm (i.e. their leftmost or rightmost item was relabeled). However if the borders were not crossed, a good interval could technically become bad by redistributing the items within the interval.

Here we crucially use an additional property of good intervals. If the interval does not contain a large enough interval with much higher density, then all subintervals are almost as dense as the whole interval. Using this we can show, that unless a significant number of items which are in buffers of the interval are moved, a subinterval with almost the same density can be found.

3.4.2 Charging Scheme

Charging scheme defines a way we assign the cost of the algorithm among the inserted items. This scheme is used to lower bound the cost of the algorithm.

First we show that the cost of the algorithm at each step can be lower bound by the number of items in buffers of good intervals which were rebuilt in that step. Notice that we completely ignore the moves of items which occur in bad intervals of the chain, still the remaining cost will be large enough.

Now we focus on each interval that was rebuilt separately. We distribute the cost which equals to the size of buffers of each interval among the last half of items inserted to this interval since the interval was rebuilt the last time. We say that these items were charged at this step. The reason for not charging all of the items will be explained later. This is correct as from construction of the adversary it will be obvious that buffers of different intervals are disjoint. Notice that items may be charged on multiple levels in a single step.

Next we lower bound the total cost assigned to certain items. We choose those items which were charged enough times (i.e., at least on constant factor of all levels). First we estimate the charge assigned to the item at certain level to be (roughly) the fraction of the density of the interval on that level and the density of the interval on the next level. We consider the densities with respect to the step when the item was inserted. This approximates the size of the interval buffers divided by the number of items inserted to the level since the last rebuilt. This is

also the reason why we only consider the last half of the items inserted since last rebuilt, because for them we can estimate their charge well.

However the reason we insist on estimating charge using the densities at one particular step is that the way we build interval chain puts some limits on densities of intervals in it at each step (but not between the steps). In particular, the densities of intervals in which one particular item was charged cannot be growing that much. We use this to show that the number of newly inserted items among all the intervals in which the item was charged is limited. Thus the number of items among which the cost is distributed is relatively small and we can find sufficient lower bound on cost charged to the items.

Finally we show, that the number of items which are charged on sufficient number of levels is at least constant fraction of all items. This follows from the fact that the number of good levels at each step is a constant fraction of depth of the chain.

This result is summarized in the following theorem.

Theorem 3.3. *There are positive constants C_0, and C_1 so that the following holds. Let $\mathcal{A}$ be a deterministic algorithm with parameters (n, m, r), such that $C_0 \leq n \leq m \leq \frac{1}{4}n^{1+1/16}$ and $r \geq 2^n - 1$. Then $\chi_{\mathcal{A}}(n, m, r) \geq C_1 \cdot \frac{n \log^2(n)}{2+\log m - \log n}$.*

3.5 Online Labeling with Small Label Space and Universe

We prove an $\Omega(n \log^2(n))$ lower bound on the number of moves for inserting n items into an array of size $m = O(n)$ for any deterministic online labeling algorithm, matching the known upper bound up to constant factors. For the case of array of size $m \leq n + n^{1-\varepsilon}$ (where ε is a positive constant) we prove the asymptotically optimal lower bound $\Omega(n \log^3(n))$.

As noted before, if $r \leq m$ (where r is the size of input domain) then there is a trivial solution of cost n. A natural question (which has not, to our knowledge, been addressed previously in the literature) is whether it might be possible to improve the $O(n \log^2(n))$ upper bound if r is much larger than m but still restricted. Our lower bounds rule out such an improvement: we obtain an $\Omega(n \log^2(n))$ lower bound provided that r is at least a sufficiently large constant times m.

These lower bounds extend to slightly superlinear array size but they are inferior to previous result. On the other hand, in previous result we do not handle optimally arrays of size $m \leq n + n^{1-\varepsilon}$ (i.e. very small arrays) and we do not consider the case of small item universe.

This result is based on joint work with Michal Koucký and Michael Saks [10] which was the first lower bound for general algorithms in the small space regime. Previously Dietz et al. [12, 21] proved a $\Omega(n \log^2(n))$ lower bound for the restricted

case of so-called *smooth algorithms*. These lower bounds are especially interesting because the best known algorithms for the problem are smooth. Nevertheless, the restriction to smooth algorithms is significant. The lower bound for the small space regime of smooth algorithms is obtained by considering the trivial adversary that inserts items in decreasing order. This lower bound clearly relies heavily on the smoothness of the algorithm; a non-smooth algorithm can easily handle the given adversary with constant amortized time per item. It was not known whether a non-smooth algorithm could give a significant advantage over a smooth algorithm on general inputs. Our lower bound rules this out.

There is some confusion in the literature regarding the lower bounds of [12, 21]; the fact that they apply only to smooth algorithms is sometimes not mentioned ([7]), creating the impression that the general lower bound was already established.

3.5.1 Proof Techniques

We will describe our results in the language of the file maintenance problem, in which arriving items are placed in an array, rather than the online labeling problem. We give a lower bound on the cost of inserting n additional items into an array that is already partially full with $n_0 \geq n$ items. The $\Omega(n \log^2(n))$ lower bound for file maintenance problem for the case that $m = \Theta(n)$ is then an immediate consequence. By iterative application of this result we prove $\Omega(\log^3(n))$ lower bound for an array of size $m \leq n + n^{1-\varepsilon}$ This parallels the structure of the iterative algorithm from previous part that gives a matching upper bound.

The general idea builds heavily on the prior work [12, 21, 13] and is similar to ideas in previous result. However, we use a dual approach. The adversary will maintain a chain of nested segments of the array.

Recall, that the key challenge is to give an adversary procedure for building segment chains that ensures large buffers, but the density degrades very slowly. The previous result builds on the observation that if for a given segment every subsegment having large buffers has density significantly smaller than the given segment, then there must be a large subsegment (located near the boundary of the given segment) having substantially higher density than the given segment. This allows us to build a chain of $\Theta(\log(n))$ segments, such that (1) a constant fraction of the segments have large buffers with respect to their predecessors (good segments), (2) the segments that don't have large buffers have significantly higher density than their predecessor segments (bad segments), and (3) the degradation of density along the entire chain can be bounded by a constant factor. (To give a rough idea of the choice of parameters, when $m = \Theta(n)$, we allow decrease in density by a factor of at most $(1 - O(1/\log(n)))$ in a single step.)

The adversary doesn't work exactly like this. Unlike in previous result we

collapse consecutive bad segments into one and then we omit them from the chain. Thus, the chain of segments is built so that every segment in the chain (not just a constant fraction) has left and right buffers whose sizes are a constant fraction of the length of the segment. We do this by relaxing the requirement that the length of each segment in the chain is at least a constant fraction of the length of its predecessor. If we encounter a segment whose items are concentrated near the boundary then the next segment will be a small subsegment of high enough density whose distance from the boundary is large relative to its own size, even if this distance is small relative to the size of the predecessor segment. This raises a new problem: once we allow successive subsegments to shrink by more than a constant fraction we face the problem that the length of the segment chain d may not be $\Omega(\log(n))$. This is important because the lower bound that comes out of the analysis is proportional to d^2. So we need to ensure that the chains have length $\Omega(\log(n))$ even though we allow the length of segments to drop significantly. We show that we can construct a sequence of segments where each selected segment satisfies a strong uniformity property called *lower balance*: It has no subsegment of length at least $1/4$ of its length that has density significantly smaller than the given segment. (In the lower bound for smooth algorithms mentioned earlier, the ability to find such a sequence is essentially built into the smoothness restriction. Our construction allows us to dispense with this assumption.) To find the successor segment S' of a given segment S we first restrict to the middle third T of the segment and choose S' to be a subsegment of T. The restriction to a subsegment of T ensures that S' has large buffers relative to S. Furthermore, the uniformity property of S ensures that the density of T is close to that of S. We want to choose S' inside T having density at least that of T, having size not much smaller than T, and having the desired uniformity property. To identify S' we maximize a certain quality function of the form $\rho(I)|I|^\kappa$ (where $\rho(I)$ is the density of items stored in I and κ is a small positive parameter). This balances the requirement that S' have high density and large size. Furthermore, choosing S' in this way guarantees that S' has the needed uniformity property (since the presence of a subsegment that violates the uniformity property would imply that there is a subsegment of S' that has a higher quality). Maximizing the quality function implicitly captures the process of successively choosing subsegments of significantly higher density until one arrives at a subsegment for which no such selection is possible.

After identifying a segment chain with the required properties at each step, the item selected by the adversary to insert is one whose value is between two items stored in the final segment of the chain. Whenever the maintenance algorithm rearranges some portion of the array the adversary rebuilds the affected portion of the segment chain. To obtain our lower bound $\Omega(n \log^2(n))$ we use a careful accounting argument that encapsulates and extends the clever argument used in

[12, 21] to obtain an $\Omega(n \log^2(n))$ bound for smooth algorithms.

There is one additional complication that arises because we want our lower bounds to apply even in the case that the range r of items is relatively small. In a given step, after selecting the chain, the adversary is supposed to choose the next item to insert to be an item that is between two items currently stored in the final segment of the chain. However, if the set of items stored in the final segment are a consecutive subset of the set of possible values, the adversary is unable to choose an item to insert. Of course this is not a problem if the range of values is, for example, all rationals in a given interval but it is a potential problem if the range is a bounded subset of the integers. For example, we noted earlier that if the range of possible items is small enough, $r \leq m$, then there is a trivial algorithm that incurs only unit cost per item, so our lower bound proof must fail. How large does r have to be so that the adversary described above can avoid this problem? It is not hard to show that $r \geq 2^n$ is sufficient, but in fact when $m = O(n)$ we only need r to be a (sufficiently large) constant multiple of n. To carry out the argument for such small r, we modify the definition of density of segments by weighting more recent items with a smaller (but still non-negligible) weight than older items. When our adversary selects a segment chain, the decreased weight on recent items will tilt the adversary to prefer segments that are crowded mainly with older items over segments crowded mainly with newer items (unless the latter is significantly more crowded than the former). The reason we want to do this is that if the adversary continues to place items in a segment that mainly has newer items there is a risk (because of the limited range size r) that the adversary will end up with a segment where the items are consecutive integers, and not have another item to insert. By giving more recent items a smaller (but not too small) weight we can avoid this possibility.

The results are summarized in next theorems. The first theorem applies whenever $2n \leq m$, and gives interesting results provided that m is not too large. In the first part of the theorem, the range $\{1, \ldots, r\}$ of possible items has size exponential in N. In the second part, r is at most a constant times m. Despite this strong limitation, the lower bound is only slightly worse.

Theorem 3.4. *There is a (sufficiently large) constant C_2 so that the following holds. Let m, n be integers satisfying $C_2 \leq n$ and $2n \leq m$. Let $\delta = n/m$. Then*

1. *If $r \geq n2^{n-1}$ then $\chi(n, m, r) \geq n \log^2(n) \frac{\delta}{C_2(\log(1/\delta))^2}$.*

2. *If $r \geq C_2 m$ then $\chi(n, m, r) \geq n \log^2(n) \frac{\delta^2}{C_2(\log(1/\delta))^2}$.*

In the next theorem we consider array size satisfying $n < m < 2n$:

Theorem 3.5. *There are (sufficiently large) constants C_2, C_3 so that the following holds. Let m, n be integers satisfying $C_3 \leq n < m < 2n$ and let $\delta = n/m$. Assume $r \geq (\frac{1}{1-\delta})^{C_2} n$. Then:*

$$\chi(n, m, r) \geq \frac{1}{C_3} n \log^2(n) \log\left(\frac{1}{1-\delta}\right). \tag{1}$$

Bibliography

[1] Yehuda Afek, Baruch Awerbuch, Serge A. Plotkin, and Michael E. Saks. *Local management of a global resource in a communication network.* Journal of the ACM, 43(1):1–19, 1996.

[2] Martin Babka, Jan Bulánek, Vladimír Čunát, Michal Koucký, and Michael E. Saks. *On online labeling with polynomially many labels.* In Proceedings of the 20th Annual European Symposium on Algorithms, ESA '12, pages 121–132, 2012.

[3] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. *Two simplified algorithms for maintaining order in a list.* In Proceedings of the 10th Annual European Symposium on Algorithms, ESA '02, pages 152–164, 2002.

[4] Michael A. Bender, Erik D. Demaine, and Martin Farach-Colton. *Cache-oblivious B-trees.* SIAM Journal on Computing, 35:341–358, 2005.

[5] Michael A. Bender, Ziyang Duan, John Iacono, and Jing Wu. *A locality-preserving cache-oblivious dynamic dictionary.* Journal of Algorithms, 53(2):115 – 136, 2004.

[6] Michael A. Bender, and Haodong Hu. *An adaptive packed-memory array.* ACM Transactions on Database Systems 32(4), 2007.

[7] Richard S. Bird and Stefan Sadnicki. *Minimal on-line labelling.* Information Processing Letters, 101:41–45, 2007.

[8] Alan Borodin, and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[9] Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. *Cache oblivious search trees via binary trees of small height.* In Proceedings of the 13th annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02, pages 39–48, 2002.

[10] Jan Bulánek, Michal Koucký, and Michael E. Saks. *Tight lower bounds for the online labeling problem.* In Proceedings of the 44th Symposium of Theory of Computation, STOC '12, pages 1185–1198, 2012.

[11] Jan Bulánek, Michal Koucký, and Michael Saks. *On Randomized Online Labeling with Polynomially Many Labels.* In Proceedings of the 40th International Colloquium on Automata, Languages and Programming, ICALP '13, pages 291-302, 2013.

[12] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. *Lower bounds for smooth list labeling.* Manuscript, 2005. (Listed in the references of [13]).

[13] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. *A tight lower bound for online monotonic list labeling.* SIAM Journal on Discrete Mathematics, 18:626–637, 2005.

[14] Paul F. Dietz and Ju Zhang. *Lower bounds for monotonic list labeling.* In Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory, SWAT '90, pages 173–180, 1990.

[15] Yuval Emek and Amos Korman. *New bounds for the controller problem.* Distributed Computing, 24(3-4):177–186, 2011.

[16] Alon Itai, and Irit Katriel. *Canonical density control.* Information Processing Letters, 104:200–204, 2007.

[17] Alon Itai, Alan G. Konheim, and Michael Rodeh. *A sparse table implementation of priority queues.* In Proceedings of the 8th Colloquium on Automata, Languages and Programming, ICALP '81, pages 417–431, 1981.

[18] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*, Addison Wesley Longman Publishing Co., Inc., 1998.

[19] Amos Korman and Shay Kutten. *Controller and estimator for dynamic networks.* In Proceedings of the 26th annual ACM Symposium on Principles of Distributed Computing, PODC '26, pages 175–184, 2007.

[20] Dan E. Willard. *A density control algorithm for doing insertions and deletions in a sequentially ordered file in a good worst-case time.* Information and Computation, 97(2):150–204, 1992.

[21] Ju Zhang. *Density Control and On-Line Labeling Problems.* PhD thesis, University of Rochester, Rochester, NY, USA, 1993.

Author's Publications

- Martin Babka, Jan Bulánek, Vladimír Čunát, Michal Koucký, and Michael E. Saks. *On online labeling with polynomially many labels.* In Proceedings of the 20th Annual European Symposium on Algorithms, ESA '12, pages 121–132, 2012.

- Jan Bulánek, Michal Koucký, and Michael E. Saks. *Tight lower bounds for the online labeling problem.* In Proceedings of the 44th Symposium of Theory of Computation, STOC '12, pages 1185–1198, 2012, Revised version submitted to SIAM Journal on Computing (SICOMP).

- Jan Bulánek, Michal Koucký, and Michael Saks. *On Randomized Online Labeling with Polynomially Many Labels.* In Proceedings of the 40th International Colloquium on Automata, Languages and Programming, ICALP '13, pages 291-302, 2013.

- Zbyněk Falt, Jan Bulánek, and Jakub Yaghob. *On Parallel Sorting of Data Streams.* In Proceedings of the 16th Advances in Databases and Information Systems, ADBIS '13, pages 291-302, 2013.

- Jan Bulánek. *Unordered vs. Prefix Bucketing.* Submitted to SIAM Journal on Discrete Math (SIDMA).