Cell Probe Lower Bounds and Approximations for Range Mode

Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen

MADALGO[⋆], Department of Computer Science, Aarhus University, Denmark.
{mgreve,jallan,larsen,jakobt}@madalgo.au.dk

Abstract. The mode of a multiset of labels, is a label that occurs at least as often as any other label. The input to the range mode problem is an array A of size n. A range query $[i, j]$ must return the mode of the subarray $A[i], A[i+1], \ldots, A[j]$. We prove that any data structure that uses S memory cells of w bits needs $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a range mode query. Secondly, we consider the related range k-frequency problem. The input to this problem is an array A of size n, and a query $[i, j]$ must return whether there exists a label that occurs precisely k times in the subarray $A[i], A[i+1], \ldots, A[j]$. We show that for any constant $k > 1$, this problem is equivalent to 2D orthogonal rectangle stabbing, and that for $k = 1$ this is no harder than four-sided 3D orthogonal range emptiness. Finally, we consider approximate range mode queries. A c-approximate range mode query must return a label that occurs at least $1/c$ times that of the mode. We describe a linear space data structure that supports 3-approximate range mode queries in constant time, and a data structure that uses $O(\frac{n}{\varepsilon})$ space and supports $(1+\varepsilon)$-approximation queries in $O(\log \frac{1}{\varepsilon})$ time.

1 Introduction

In this paper we consider the range mode problem, the range k-frequency problem, and the c-approximate range mode problem. The frequency of a label l in a multiset S of labels, is the number of occurrences of l in S. The mode of S is the most frequent label in S. In case of ties, any of the most frequent labels in S can be designated the mode.

For all the problems we consider the input is an array A of length n containing labels. For simplicity we assume that each label is an integer between one and n. In the range mode problem, we must preprocess A into a data structure that given indices i and j, $1 \leq i \leq j \leq n$, returns the mode, $M_{i,j}$, in the subarray $A[i, j] = A[i], A[i+1], \ldots, A[j]$. We let $F_{i,j}$ denote the frequency of $M_{i,j}$ in $A[i, j]$. In the c-approximate range mode problem, a query is given indices i and j, $1 \leq i \leq j \leq n$, and returns a label that has a frequency of at least $F_{i,j}/c$.

[⋆] Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

In the range k-frequency problem, a query is given indices i and j, $1 \leq i \leq j \leq n$, and returns whether there is a label occurring exactly k times in $A[i, j]$.

For the upper bounds we consider the unit cost RAM with word size $w = \Theta(\log n)$. For lower bounds we consider the cell probe model of Yao [1]. In this model of computation a random access memory is divided into cells of w bits. The complexity of an algorithm is the number of memory cells the algorithm accesses. All other computations are free.

Previous Results. The first data structure supporting range mode queries in constant time was developed in [2], and this data structure uses $O(n^2 \log\log n / \log n)$ space. This was subsequently improved to $O(n^2/\log n)$ space in [3] and finally to $O(n^2 \log\log n/\log^2 n)$ in [4]. For non-constant query time, the first data structure developed uses $O(n^{2-2\varepsilon})$ space and answers queries in $O(n^{\varepsilon} \log n)$ time, where $0 < \varepsilon \leq \frac{1}{2}$ is a query-space tradeoff constant [2]. The query time was later improved to $O(n^{\varepsilon})$ without changing the space bound [3].

Given the rather large bounds for the range mode problem, the approximate variant of the problem was considered in [5]. With constant query time, they solve 2-approximate range mode with $O(n \log n)$ space, 3-approximate range mode with $O(n \log\log n)$ space, and 4-approximate range mode with linear space. For $(1+\varepsilon)$-approximate range mode, they describe a data structure that uses $O(\frac{n}{\varepsilon})$ space and answers queries in $O(\log\log_{(1+\varepsilon)} n) = O(\log\log n + \log \frac{1}{\varepsilon})$ time. This data structure gives a linear space solution with $O(\log\log n)$ query time for c-approximate range mode when c is constant. There are no known non-trivial lower bounds for the any of the problems we consider.

Our Results. In this paper we show the first lower bounds for range mode data structures and range k-frequency data structures and provide new upper bounds for the c-approximate range mode problem and the range k-frequency problem.

In Section 2 we prove our lower bound for range mode data structures. Specifically, we prove that any data structure that uses S cells and supports range mode queries must have a query time of $\Omega(\frac{\log n}{\log(Sw/n)})$. This means that any data structure that uses $O(n \log^{O(1)} n)$ space needs $\Omega(\log n/\log\log n)$ time to answer a range mode query. Similarly, any data structure that supports range mode queries in constant time needs $n^{1+\Omega(1)}$ space.

We suspect that the actual lower bound for near-linear space data structures for the range mode problem is significantly larger. However a fundamental obstacle in the cell probe model is to prove lower bounds for static data structures that are higher than the number of bits needed to describe the query. The highest known lower bounds are achieved by the techniques in [6, 7] that uses reductions from problems in communication complexity. We use this technique to obtain our lower bound and our bound matches the highest lower bound achieved with this technique.

Actually our construction proves the same lower bound for queries on the form, is there an element with frequency at least (or precisely) k in $A[i, j]$, where k is given at query time. In the scenario where k is fixed for all queries it is trivial to give a linear space data structure with constant query time for determining whether there is an element with frequency at least k. In Section 3 we consider the case of determining whether there is an element with frequency exactly k, which we denote the range k-frequency problem. To the best of our knowledge, we are the first to consider this problem. We show that 2D rectangle stabbing reduces to range k-frequency for any constant $k > 1$. This reduction proves that any data structure that uses S space, needs $\Omega(\log n/\log(Sw/n))$ time for a query [7, 8], for any constant $k > 1$. Secondly, we reduce range k-frequency to 2D rectangle stabbing. This reduction works for any k. This immediately gives a data structure for range k-frequency that uses linear space, and answers queries in optimal $O(\log n/\log\log n)$ time [9] (we note that 2D rectangle stabbing reduces to 2D range counting). In the restricted case where $k = 1$, this problem corresponds to determining whether there is a unique label in a subarray. The reduction from 2D rectangle stabbing only applies for $k > 1$. We show, somewhat surprisingly, that determining whether there is a label occurring exactly twice (or $k > 1$ times) in a subarray, is exponentially harder than determining if there is a label occurring exactly once. Specifically, we reduce range 1-frequency to four-sided 3D orthogonal range emptiness, which can be solved with $O(\log^2\log n)$ query time and $O(n\log n)$ space by a slight modification of the data structure presented in [10].

In Section 4 we present a simple data structure for the 3-approximate range mode problem. The data structure uses linear space and answers queries in constant time. This improves the best previous 3-approximate range mode data structures by a factor $O(\log\log n)$ either in space or query time. With linear space and constant query time, the best previous approximation factor was 4. In Section 5 we use our 3-approximate range mode data structure, to develop a data structure for $(1+\varepsilon)$-approximate range mode. This data structure uses $O(\frac{n}{\varepsilon})$ space and answers queries in $O(\log\frac{1}{\varepsilon})$ time. This removes the dependency on n in the query time compared to the previously best data structure, while matching the space bound. Thus, we have a linear space data structure with constant query time for the c-approximate range mode problem for any constant $c > 1$. We note that we get the same bound if we build on the 4-approximate range mode data structure from [5].

2 Cell Probe Lower Bound for Range Mode

In this section we show a query lower bound of $\Omega(\log n/\log(Sw/n))$ for any range mode data structure that uses S space for an input array of size n. The lower bound is proved for the slightly different problem of determining the frequency of the mode. Since the frequency of an element in any range can be determined in $O(\log\log n)$ time by a linear space data structure the lower bound for range mode follows. This data structure stores a linear space static rank data structure [11] for each label ℓ in the input, containing the positions in A storing ℓ. The frequency of a label in $A[i, j]$ is the rank difference between $i - 1$ and j.

Communication Complexity and Lower Bounds. In communication complexity we have two players Alice and Bob. Alice receives as input a bit string x and

Bob a bit string y. Given some predefined function, f, the goal for Alice and Bob is to compute $f(x, y)$ while communicating as few bits as possible.

Lower bounds on the communication complexity of various functions have been turned into lower bounds for static data structure problems in the cell probe model. The idea is as follows [12]: Assume we are given a static data structure problem and consider the function $f(q, D)$ that is defined as the answer to a query q on an input set D for this problem. If we have a data structure for the problem that uses S memory cells and supports queries in time t we get a communication protocol for f where Alice sends $t \log S$ bits and Bob sends tw bits. In this protocol Alice receives q and Bob receives D. Bob constructs the data structure on D and Alice simulates the query algorithm. In each step Alice sends $\log S$ bits specifying the memory cell of the data structure she needs and Bob replies with the w bits of this cell. Finally, Alice outputs $f(q, D)$. Thus, a communication lower bound for f gives a lower bound tradeoff between S and t.

This construction can only be used to distinguish between polynomial and superpolynomial space data structures. Since range mode queries are trivially solvable in constant time with $O(n^2)$ space, we need a different technique to obtain lower bounds for near-linear space data structures. Pătraşcu and Thorup [6, 7] have developed a technique for distinguishing between near linear and polynomial space by considering reductions from communication complexity problems to k parallel data structure queries. The main insight is that Alice can simulate all k queries in parallel and only send $\log \binom{S}{k} = O(k \log \frac{S}{k})$ bits to define the k cells she needs. For the right values of k this is significantly less than $k \log S$ bits which Alice needs if she performs the queries sequentially.

Lopsided Set Disjointness (LSD). In LSD Alice and Bob receive subsets S and T of a universe U. The goal for Alice and Bob is to compute whether $S \cap T \neq \emptyset$. LSD is parameterized with the size $|S| = N$ of Alice's set and the fraction between the size of the universe and N, which is denoted B, e.g. $|U| = NB$. Notice that the size of Bob's set is arbitrary and could be as large as NB. We use $[X]$ to denote the set $\{1, 2, \ldots, X\}$. There are other versions of LSD where the input to Alice has more structure. For our purpose we need Blocked-LSD. For this problem the universe is considered as the cartesian product of $[N]$ and $[B]$, e.g. $U = [N] \times [B]$ and Alice receives a set S such that $\forall j \in [N]$ there exists a unique $b_j \in [B]$ such that $(j, b_j) \in S$, e.g. S is of the form $\{(1, b_1), (2, b_2), \ldots, (N, b_N) \mid b_i \in [B]\}$. The following lower bound applies for this problem [7].

Theorem 1. *Fix $\delta > 0$. In a bounded-error protocol for Blocked-LSD, either Alice sends $\Omega(N \log B)$ bits or Bob sends $\Omega(NB^{1-\delta})$ bits.*

Blocked-LSD reduces to N/k parallel range mode queries. Given n, we describe a reduction from Blocked-LSD with a universe of size n ($n = NB$) to N/k parallel range mode queries on an input array A of size $\Theta(n)$. The size of A may not be exactly n but this will not affect our result. The parameters k and B are fixed later in the construction. From a high level perspective we construct an array of permutations of $[kB]$. A query consists of a suffix of one permutation, a number of complete permutations, and a prefix of another permutation. They are chosen such that the suffix determines a subset of Bob's set and the prefix a subset of Alice's set. These two subsets intersect if and only if the frequency of the mode is equal to two plus the number of complete permutations spanned by the query.

Bob stores a range mode data structure and Alice simulates the query algorithm. First we describe the array A that Bob constructs when he receives his input. Let $T \subseteq [N] \times [B]$ be this set. The array Bob constructs consists of two parts which are described separately. We let $\cdot$ denote concatenation of lists. We also use this operator on sets and in this case we treat the set as a list by placing the elements in lexicographic order. Bob partitions $[N]$ into N/k consecutive chunks of k elements, e.g. the i'th chunk is $\{(i-1)k+1, \ldots, ik\}$ for $i = 1, \ldots, N/k$. With the i'th chunk Bob associates the subset L_i of T with first coordinate in that chunk, e.g. $L_i = T \cap (\{(i-1)k+t \mid t = 1, \ldots, k\} \times [B])$. Each L_i is mapped to a permutation of $[kB]$.

We define the mapping $f : (x, y) \to (x - 1 \bmod k)B + y$ and let the permutation be $([kB] \setminus f(L_i)) \cdot f(L_i)$, e.g. we map the elements in L_i into $[kB]$ and prepend the elements of $[kB]$ not mapped to by any element in L_i such that we get a full permutation of $[kB]$. The first part of A is the concatenation of the permutations defined for each chunk L_i ordered by i, e.g. $([kB] \setminus f(L_1)) \cdot f(L_1) \cdots ([kB] \setminus f(L_{N/k})) \cdot f(L_{N/k})$. The second part of A consists of B^k permutations of $[kB]$. There is one permutation for each way of picking a set of the form $\{(1, b_1), \ldots, (k, b_k) \mid b_i \in [B]\}$. Let $R_1, \ldots, R_{B^k}$ denote the B^k sets on this form ordered lexicographically. The second part of the array becomes $f(R_1) \cdot ([kB] \setminus f(R_1)) \cdots f(R_{B^k}) \cdot ([kB] \setminus f(R_{B^k}))$.

We now show how Alice and Bob can determine whether $S \cap T \neq \emptyset$ from this array. Bob constructs a range mode data structure for A and sends $|L_i|$ for $i = 1, \ldots, N/k$ to Alice. Alice then simulates the query algorithm on the range mode data structure for N/k queries in parallel. The i'th query determines whether the k elements $Q_i = \{((i-1)k+1, b_{(i-1)k+1}), \ldots, (ik, b_{ik})\}$ from S have an empty intersection with T (actually L_i) as follows.

Alice determines the end index of $f(Q_i)$ in the second part of A. We note that $f(Q_i)$ always exists in the second part of A by construction and Alice can determine the position without any communication with Bob. Alice also determines the start index of $f(L_i)$ in the first part of A from the sizes she initially received from Bob. The i'th query computes the frequency R_i of the mode between these two indices. Let p be the number of permutations of $[kB]$ stored between the end of $f(L_i)$ and the beginning of $f(Q_i)$ in A, then $F_i - p = 2$ if and only if $Q_i \cap T \neq \emptyset$, and $F_i - p = 1$ otherwise. Since each permutation of $[kB]$ contributes one to F_i, $F_i - p$ is equal to two if and only if at least one of the elements from Q_i is in L_i meaning that $S \cap T \neq \emptyset$. We conclude that Blocked-LSD reduces to N/k range mode queries in an array of size $NB + B^k kB$.

To obtain a lower bound for range mode data structures we consider the parameters k and B and follow the approach from [7]. Let S be the size of Bob's range mode data structure and let t be the query time. In our protocol for Blocked-LSD Alice sends $t \log \binom{S}{N/k} = O(t \frac{N}{k} \log \frac{Sk}{N})$ bits and Bob sends

$twN/k + N/k \log(kB)$ bits. By Theorem 1, either Alice sends $\Omega(N \log B)$ bits or Bob sends $\Omega(NB^{1-\delta})$. Fix $\delta = \frac{1}{2}$. Since $N/k \log(kB) = o(N\sqrt{B})$ we obtain that either $t\frac{N}{k} \log(\frac{Sk}{N}) = \Omega(N \log B)$ or $twN/k = \Omega(N\sqrt{B})$. We constrain B such that $B \geq w^2$ and $\log B \geq \frac{1}{2} \log(\frac{Sk}{N}) \Rightarrow B \geq \frac{Sk}{n}$ and obtain $t = \Omega(k)$. Since $|A| = NB + B^k kB$ and we require $|A| = \Theta(n)$, we set $k = \Theta(\log_B n)$. To maximize k we choose $B = \max\{w^2, \frac{Sk}{n}\}$. We obtain that $t = \Omega(k) = \Omega(\log N / \log \frac{Swk}{n}) = \Omega(\log n / \log \frac{Sw}{n})$ since $w > k$.

Summarizing, we get the following theorem.

Theorem 2. *Any data structure that uses S space needs $\Omega\left(\frac{\log n}{\log(\frac{Sw}{n})}\right)$ time for a range mode query in an array of size n.*

It follows from the construction that we get the same lower bound for data structures that support queries that are given i, j and k, and returns whether there exists an element with frequency exactly k in $A[i, j]$ or support queries that are given i, j and k and returns whether there is an element with frequency at least k in $A[i, j]$.

3 Range k-frequency

In this section, we consider the *range k-frequency* problem and its connection to classic geometric data structure problems. We show that the range k-frequency problem is equivalent to 2D rectangle stabbing for any fixed constant $k > 1$, and that for $k = 1$ the problem reduces to four-sided 3D orthogonal range emptiness.

In the 2D rectangle stabbing problem the input is n axis-parallel rectangles. A query is given a point, (x, y), and must return whether this point is contained[1] in at least one of the n rectangles in the input. A query lower bound of $\Omega(\log n / \log(Sw/n))$ for data structures using S space is proved in [7], and a linear space static data structure with optimal $O(\log n / \log \log n)$ query time can be found in [9].

In four-sided 3D orthogonal range emptiness, we are given a set P of n points in 3D, and must preprocess P into a data structure, such that given an open-ended four-sided rectangle $R = (-\infty, x] \times [y_1, y_2] \times [z, \infty)$, the data structure returns whether R contains a point $p \in P$. Currently, the best solution for this problem uses $O(n \log n)$ space and supports queries in $O(\log^2 \log n)$ time [10].

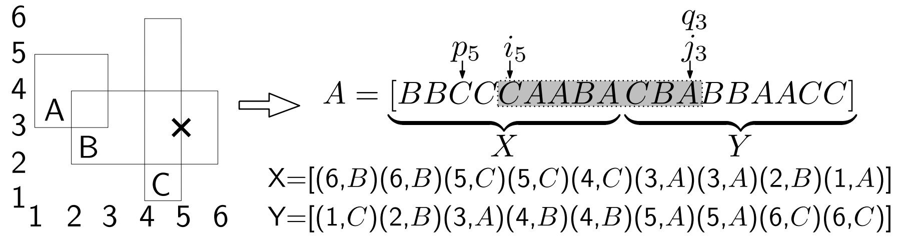
For simplicity, we assume that each coordinate is a unique integer between one and $2n$ (rank space).

Theorem 3. *Let k be a constant greater than one. The 2D rectangle stabbing problem reduces to the range k-frequency problem.*

Proof. We show the reduction for $k = 2$ and then generalize this construction to any constant value $k > 2$.

Let $R_1, \ldots, R_n$ be the input to the rectangle stabbing problem. We construct a range 2-frequency instance with n distinct labels each of which is duplicated

[1] points on the border of a rectangle are contained in the rectangle



6
5
4
3
2
1
1 2 3 4 5 6
A
B
C
×
p_5
i_5
q_3
j_3
$A = [BBCCCAABACBABBAACC]$
X
Y
X=[(6,B)(6,B)(5,C)(5,C)(4,C)(3,A)(3,A)(2,B)(1,A)]
Y=[(1,C)(2,B)(3,A)(4,B)(4,B)(5,A)(5,A)(6,C)(6,C)]

Fig. 1. Reduction from 2D rectangle stabbing to range 2-frequency. The $\times$ marks a stabbing query, $(5, 3)$. This query is mapped to the range 2-frequency query $[i_5, |X|+j_3]$ in A, which is highlighted. Notice that $i_5 = p_5 + 2$ since $A[p_5] = A[p_5 + 1]$.

exactly 6 times. Let R_ℓ be the rectangle $[x_{\ell_0}, x_{\ell_1}] \times [y_{\ell_0}, y_{\ell_1}]$. For each rectangle, R_ℓ, we add the pairs (x_{ℓ_0}, ℓ), (x_{ℓ_1}, ℓ) and (x_{ℓ_1}, ℓ) to a list X. Similarly, we add the pairs (y_{ℓ_0}, ℓ), (y_{ℓ_1}, ℓ), and (y_{ℓ_1}, ℓ) to a list Y. We sort X in descending order and Y in ascending order by their first coordinates. Since we assumed all coordinates are unique, the only ties are amongst pairs originating from the same rectangle, here we break the ties arbitrarily. The concatenation of X and Y is the range 2-frequency instance and we denote it A, i.e. the second component of each pair are the actual entries in A, and the first component of each pair is ignored.

We translate a 2D rectangle stabbing query, (x, y), into a query for the range 2-frequency instance as follows. Let p_x be the smallest index where the first coordinate of $X[p_x]$ is x, and let q_y be the largest index where the first coordinate of $Y[p_y]$ is y. If $A[p_x] = A[p_x + 1]$, two consecutive entries in A are defined by the right endpoint of the same rectangle, we set $i_x = p_x + 2$ (we move i_x to the right of the two entries), otherwise we set $i_x = p_x$. Similarly for the y coordinates, if $A[|X| + q_y] = A[|X| + q_y - 1]$ we set $j_y = q_y - 2$ (move j_y left of the two entries), otherwise we set $j_y = q_y$. Finally we translate (x, y) to the range 2-frequency query $[i_x, |X|+j_y]$ on A, see Figure 1. Notice that in the range 2-frequency queries that can be considered in the reduction, the frequency of a label is either one, two, three, four or six. The frequency of label ℓ in $A[i_x, |X|]$ is one if $x_{\ell_0} \le x \le x_{\ell_1}$, three if $x > x_{\ell_1}$ and zero otherwise. Similar, the frequency of ℓ in $A[|X| + 1, |X| + j_y]$ is one if $y_{\ell_0} \le y \le y_{\ell_1}$, three if $y > y_{\ell_1}$ and zero otherwise. We conclude that the point (x, y) stabs rectangle R_ℓ if and only if the label ℓ has frequency two in $A[i_x, |X| + j_y]$.

Since $x, y \in \{1, \ldots, 2n\}$, we can store a table with the translations from x to i_x and y to j_y. Thus, we can translate 2D rectangle stabbing queries to range 2-frequency queries in constant time.

For $k > 2$ we place $k - 2$ copies of each label between X and Y and translate the queries accordingly. □

The following theorem provides a matching upper bound.

Theorem 4. *The range k-frequency problem reduces to 2D rectangle stabbing.*

Proof. Let A be the input to the range k-frequency problem. We translate the ranges of A where there is a label with frequency k into $O(n)$ rectangles as follows. Fix a label $x \in A$, and let $s_x \geq k$ denote the number of occurrences of x in A. If $s_x < k$ then x is irrelevant and we discard it. Otherwise, let $i_1 < i_2 < \ldots < i_s$ be the position of x in A, and let $i_0 = 0$ and $i_{s+1} = n + 1$. Consider the ranges of A where x has frequency k. These are the subarrays, $A[a, b]$, where there exists an integer ℓ such that $i_\ell < a \leq i_{\ell+1}$ and $i_{\ell+k} \leq b < i_{\ell+k+1}$ for $0 \leq \ell \leq s_x - k$. This defines $s_x - k + 1$ two dimensional rectangles, $[i_\ell + 1, i_{\ell+1}] \times [i_{\ell+k}, i_{\ell+k+1} - 1]$ for $\ell = 0, \ldots, s_x - k$, such that x has frequency k in $A[i, j]$ if and only if the point (i, j) stabs one of the $s_x - k + 1$ rectangles defined by x. By translating the ranges of A where a label has frequency k into the corresponding rectangles for all distinct labels in A, we get a 2D rectangle stabbing instance with $O(n)$ rectangles. □

This means that we get a data structure for the range k-frequency problem that uses $O(n)$ space and supports queries in $O(\log n / \log\log n)$ time.

Theorem 5. *For $k = 1$, the range k-frequency problem reduces to four-sided orthogonal range emptiness queries in 3D.*

Proof. For each distinct label $x \in A$, we map the ranges of A where x has frequency one (it is unique in the range) to a 3D point. Let $i_1 < i_2 < \ldots < i_s$ be the positions of x in A, and let $i_0 = 0$ and $i_{s+1} = n+1$. The label x has frequency one in $A[a, b]$ if there exist an integer ℓ such that $i_{\ell-1} < a \leq i_\ell \leq b < i_{\ell+1}$. We define s points, $P_x = \{(i_{\ell-1} + 1, i_\ell, i_{\ell+1} - 1) \mid 1 \leq \ell \leq s\}$. The label x has frequency one in the range $A[a, b]$ if and only if the four-sided orthogonal range query $[-\infty, a] \times [a, b] \times [b, \infty]$ contains a point from P_x (we say that x is inside range $[x_1, x_2]$ if $x_1 \leq x \leq x_2$). Therefore, we let $P = \bigcup_{x \in A} P_x$ and get a four-sided 3D orthogonal range emptiness instance with $O(n)$ points. □

Thus, we get a data structure for the range 1-frequency problem that uses $O(n \log n)$ space and supports queries in $O(\log^2 \log n)$ time and we conclude that for data structures using $O(n \log^{O(1)} n)$ space, the range k-frequency problem is exponentially harder for $k > 1$ than for $k = 1$.

4 3-Approximate Range Mode

In this section, we construct a data structure that given a range $[i, j]$ computes a 3-approximation of $F_{i,j}$.

We use the following observation also observed in [5]. If we can cover $A[i, j]$ with three disjoint subintervals $A[i, x], A[x + 1, y]$ and $A[y + 1, j]$ then we have $\frac{1}{3} F_{i,j} \leq \max\{F_{i,x}, F_{x+1,y}, F_{y+1,j}\} \leq F_{i,j}$.

First, we describe a data structure that uses $O(n \log\log n)$ space, and then we show how to reduce the space to $O(n)$. The data structure consists of a tree T of polynomial fanout where the i'th leaf stores $A[i]$, for $i = 1, \ldots, n$. For a node v let T_v denote the subtree rooted at v and let $|T_v|$ denote the number of leaves in T_v. The fanout of node v is $f_v = \lceil \sqrt{|T_v|} \rceil$. The height of T is $\Theta(\log \log n)$. Along with T, we store a lowest common ancestor (LCA) data structure, which given indices i and j, finds the LCA of the leaves corresponding to i and j in T in constant time [13].

For every node $v \in T$, let $R_v = A[a, b]$ denote the consecutive range of entries stored in the leaves of T_v. The children $c_1, \ldots, c_{f_v}$ of v partition R_v into f_v disjoint subranges $R_{c_1} = A[a_{c_1}, b_{c_1}]$, $\ldots$, $R_{c_{f_v}} = A[a_{c_{f_v}}, b_{c_{f_v}}]$ each of size $O(\sqrt{|T_v|})$. For every pair of children c_r and c_s where $r < s - 1$, we store $F_{a_{c_{r+1}}, b_{c_{s-1}}}$. Furthermore, for every child range R_{c_i} we store $F_{a_{c_i}, k}$ and $F_{k, b_{c_i}}$ for every prefix and suffix range of R_{c_i} respectively. To compute a 3-approximation of $F_{i,j}$, we find the LCA of i and j. This is the node v in T for which i and j lie in different child subtrees, say T_{c_x} and T_{c_y} with ranges $R_{c_x} = [a_{c_x}, b_{c_x}]$ and $R_{c_y} = [a_{c_y}, b_{c_y}]$. We then lookup the frequency $F_{a_{c_{x+1}}, b_{c_{y-1}}}$ stored for the pair of children c_x and c_y, as well as the suffix frequency $F_{i, b_{c_x}}$ stored for the range $A[i, b_{c_x}]$ and the prefix frequency $F_{a_{c_y}, j}$ stored for $A[a_{c_y}, j]$, and return the max of these.

Each node $v \in T$ uses $O(|T_v|)$ space for the frequencies stored for each of the $O(|T_v|)$ pairs of children, and for all the prefix and suffix range frequencies. Since each node v uses $O(|T_v|)$ space and the LCA data structure uses $O(n)$ space, our data structure uses $O(n \log \log n)$ space. A query makes one LCA query and computes the max of three numbers which takes constant time.

We just need one observation to bring the space down to $O(n)$. Consider a node $v \in T$. The largest possible frequency that can be stored for any pair of children of v, or for any prefix or suffix range of a child of v is $|T_v|$, and each such frequency can be represented by $b = 1 + \lfloor \log |T_v| \rfloor$ bits. We divide the frequencies stored in v into chunks of size $\lfloor \frac{\log n}{b} \rfloor$ and pack each of them in one word. This reduces the total space usage of the nodes on level i to $O(n/2^i)$. We conclude that the data structure uses $O(n)$ space and supports queries in constant time.

Theorem 6. *There exists a data structure for the 3-approximate range mode problem that uses $O(n)$ space and supports queries in constant time.*

5 $(1 + \varepsilon)$-Approximate Range Mode

In this section, we describe a data structure using $O(\frac{n}{\varepsilon})$ space that given a range $[i, j]$, computes a $(1 + \varepsilon)$-approximation of $F_{i,j}$ in $O(\log \frac{1}{\varepsilon})$ time. Our data structure consists of two parts. The first part solves all queries $[i, j]$ where $F_{i,j} \leq \lceil \frac{1}{\varepsilon} \rceil$ (small frequencies), and the latter solves the remaining. The first data structure also decides whether $F_{i,j} \leq \lceil \frac{1}{\varepsilon} \rceil$. We use that $\frac{1}{\log(1+\varepsilon)} = O(\frac{1}{\varepsilon})$ for any $0 < \varepsilon \leq 1$.

Small Frequencies. For $i = 1, \ldots, n$ we store a table, Q_i, of length $\lceil \frac{1}{\varepsilon} \rceil$, where the value in $Q_i[k]$ is the largest integer $j \geq i$ such that $F_{i,j} = k$. To answer a query $[i, j]$ we do a successor search for j in Q_i. If j does not have a successor in Q_i then $F_{i,j} > \lceil \frac{1}{\varepsilon} \rceil$, and we query the second data structure. Otherwise, let s be the index of the successor of j in Q_i, then $F_{i,j} = s$. The data structure uses $O(\frac{n}{\varepsilon})$ space and supports queries in $O(\log \frac{1}{\varepsilon})$ time.

Large Frequencies. For every index $1 \leq i \leq n$, define a list T_i of length $t = \lceil \log_{1+\varepsilon}(\varepsilon n) \rceil$, with the following invariant: For all j, if $T_i[k-1] < j \leq T_i[k]$ then $\lceil \frac{1}{\varepsilon}(1+\varepsilon)^k \rceil$ is a $(1+\varepsilon)$-approximation of $F_{i,j}$. The following assignment of values to the lists T_i satisfies this invariant:

Let $m(i,k)$ be the largest integer $j \geq i$ such that $F_{i,j} \leq \lceil \frac{1}{\varepsilon}(1+\varepsilon)^{k+1} \rceil - 1$. For T_1 we set $T_1[k] = m(1,k)$ for all $k = 1, \ldots, t$. For the remaining T_i we set

$$T_i[k] = \begin{cases} T_{i-1}[k] \text{ if } F_{i,T_{i-1}[k]} \geq \lceil \frac{1}{\varepsilon}(1+\varepsilon)^k \rceil + 1 \\ m(i,k) \text{ otherwise} \end{cases}$$

The n lists are sorted by construction. For T_1, it is true since $m(i,k)$ is increasing in k. For T_i, it follows that $F_{i,T_i[k]} \leq \lceil \frac{1}{\varepsilon}(1+\varepsilon)^{k+1} \rceil - 1 < F_{i,T_i[k+1]}$, and thus $T_i[k] < T_i[k+1]$ for any k.

Let s be the index of the successor of j in T_i. We know that $F_{i,T_i[s]} \leq \lceil \frac{1}{\varepsilon}(1+\varepsilon)^{s+1} \rceil - 1$, $F_{i,T_i[s-1]} \geq \lceil \frac{1}{\varepsilon}(1+\varepsilon)^{s-1} \rceil + 1$ and $T_i[s-1] < j \leq T_i[s]$. It follows that

$$\lceil \tfrac{1}{\varepsilon}(1+\varepsilon)^{s-1} \rceil + 1 \leq F_{i,j} \leq \lceil \tfrac{1}{\varepsilon}(1+\varepsilon)^{s+1} \rceil - 1 \tag{1}$$

and that $\lceil \frac{1}{\varepsilon}(1+\varepsilon)^s \rceil$ is a $(1+\varepsilon)$-approximation of $F_{i,j}$.

The second important property of the n lists, is that they only store $O(\frac{n}{\varepsilon})$ different indices, which allows for a space-efficient representation. If $T_{i-1}[k] \neq T_i[k]$ then the following $\lceil \frac{1}{\varepsilon}(1+\varepsilon)^{k+1} \rceil - 1 - \lceil \frac{1}{\varepsilon}(1+\varepsilon)^k \rceil - 1 \geq \lfloor (1+\varepsilon)^k \rfloor - 3$ entries, $T_{i+a}[k]$ for $a = 1, \ldots, \lfloor (1+\varepsilon)^k \rfloor - 3$, are not changed, hence we store the same index at least $\max\{1, \lfloor (1+\varepsilon)^k \rfloor - 2\}$ times. Therefore, the number of changes to the n lists, starting with T_1, is bounded by $\sum_{k=1}^{t} \frac{n}{\max\{1, \lfloor (1+\varepsilon)^k \rfloor - 2\}} = O(\frac{n}{\varepsilon})$. This was observed in [5], where similar lists are maintained in a partially persistent search tree [14].

We maintain these lists without persistence such that we can access any entry in any list T_i in constant time. Let $I = \{1, 1+t, \ldots, 1 + \lfloor (n-1)/t \rfloor t\}$. For every $\ell \in I$ we store T_ℓ explicitly as an array S_ℓ. Secondly, for $\ell \in I$ and $k = 1, \ldots, \lceil \log_{1+\varepsilon} t \rceil$ we define a bit vector $B_{\ell,k}$ of length t and a change list $C_{\ell,k}$, where

$$B_{\ell,k}[a] = \begin{cases} 0 \text{ if } T_{\ell+a-1}[k] = T_{\ell+a}[k] \\ 1 \text{ otherwise} \end{cases}$$

Given a bit vector L, define $\text{sel}(L,b)$ as the index of the b'th one in L. We set

$$C_{\ell,k}[a] = T_{\ell+\text{sel}(B_{\ell,k},a)}[k] \ .$$

Finally, for every $\ell \in I$ and for $k = 1 + \lceil \log_{1+\varepsilon} t \rceil, \ldots, t$ we store $D_\ell[k]$ which is the smallest integer $z > \ell$ such that $T_z[k] \neq T_\ell[k]$. We also store $E_\ell[k] = T_{D_\ell[k]}[k]$. We store each bit vector in a rank and select data structure [15] that uses $O(\frac{n}{w})$ space for a bit vector of length n, and supports *rank*(i) in constant time. A rank(i) query returns the number of ones in the first i bits of the input.

Each change list, $C_{l,k}$ and every D_ℓ and E_ℓ list is stored as an array. The bit vectors indicate at which indices the contents of the first $\lceil \log_{1+\varepsilon} t \rceil$ entries of $T_\ell, \ldots, T_{\ell+t-1}$ change, and the change lists store what the entries change to. The

D_ℓ and E_ℓ arrays do the same thing for the last $t - \lceil \log_{1+\varepsilon} t \rceil$ entries, exploiting that these entries change at most once in an interval of length t.

Observe that the arrays, $C_{\ell,k}, D_\ell[k]$ and $E_\ell[k]$, and the bit vectors, $B_{\ell,k}$ allow us to retrieve the contents of any entry, $T_i[k]$ for any i, k, in constant time as follows. Let $\ell = \lfloor i/t \rfloor t$. If $k > \lceil \log_{1+\varepsilon} t \rceil$ we check if $D_\ell[k] \le i$, and if so we return $E_\ell[k]$, otherwise we return $S_\ell[k]$. If $k \le \lceil \log_{1+\varepsilon} t \rceil$, we determine $r = \text{rank}(i - \ell)$ in $B_{\ell,k}$ using the rank and select data structure. We then return $C_{\ell,k}[r]$ unless $r = 0$ in which case we return $S_\ell[k]$.

We argue that this correctly returns $T_i[k]$. In the case where $k > \lceil \log_{1+\varepsilon} t \rceil$, comparing $D_\ell[k]$ to i indicates whether $T_i[k]$ is different from $T_\ell[k]$. Since $T_z[k]$ for $z = \ell, \ldots, i$ can only change once, $T_i[k] = E_\ell[k]$ in this case. Otherwise, $S_\ell[k] = T_\ell[k] = T_i[k]$. If $k \le \lceil \log_{1+\varepsilon} t \rceil$, the rank r of $i - \ell$ in $B_{\ell,k}$, is the number of changes that has occurred in the k'th entry from list T_ℓ to T_i. Since $C_{\ell,k}[r]$ stores the value of the k'th entry after the r'th change, $C_{\ell,k}[r] = T_i[k]$, unless $r = 0$ in which case $T_i[k] = S_\ell[k]$.

The space used by the data structure is $O(\frac{n}{\varepsilon})$. We store $3\lceil \frac{n}{t} \rceil$ arrays, S_ℓ, D_ℓ and E_ℓ for $\ell \in I$, each using t space, in total $O(n)$. The total size of the change lists, $C_{\ell,k}$, is bounded by the number of changes across the T_i lists, which is $O(\frac{n}{\varepsilon})$ by the arguments above. Finally, the rank and select data structures, $B_{\ell,k}$, each occupy $O(\frac{t}{w}) = O(\frac{t}{\log n})$ words, and we store a total of $\lceil \frac{n}{t} \rceil \lceil \log_{1+\varepsilon} t \rceil$ such structures, thus the total space used by these is bounded by $O\left(\frac{t}{\log n} \frac{n}{t} \log_{1+\varepsilon} t\right) = O\left(\frac{n}{\varepsilon} \frac{\log(n \log(\varepsilon n))}{\log n}\right) = O\left(\frac{n}{\varepsilon}\right)$. We use that if $\lceil \frac{1}{\varepsilon} \rceil \ge n$ then we only store the small frequency data structure. We conclude that our data structures uses $O\left(\frac{n}{\varepsilon}\right)$ space.

To answer a query $[i, j]$, we first compute a 3-approximation of $F_{i,j}$ in constant time using the data structure from Section 4. Thus, we find $f_{i,j}$ satisfying $f_{i,j} \le F_{i,j} \le 3 f_{i,j}$. Choose k such that $\lceil \frac{1}{\varepsilon}(1+\varepsilon)^k \rceil + 1 \le f_{i,j} \le \lceil \frac{1}{\varepsilon}(1+\varepsilon)^{k+1} \rceil - 1$ then the successor of j in T_i must be in one of the entries, $T_i[k], \ldots, T_i[k + O(\log_{1+\varepsilon} 3)]$. As stated earlier, the values of T_i are sorted in increasing order, and we find the successor of j using a binary search on an interval of length $O(\log_{1+\varepsilon} 3)$. Since each access to T_i takes constant time, we use $O(\log \log_{1+\varepsilon} 3) = O(\log \frac{1}{\varepsilon})$ time.

Theorem 7. *There exists a data structure for $(1+\varepsilon)$-approximate range mode that uses $O(\frac{n}{\varepsilon})$ space and supports queries in $O(\log \frac{1}{\varepsilon})$ time.*

The careful reader may have noticed that our data structure returns a frequency, and not a label that occurs approximately $F_{i,j}$ times. We can augment our data structure to return a label instead as follows.

We set $\varepsilon' = \sqrt{(1+\varepsilon)} - 1$, and construct our data structure from above. The *small frequency* data structure is augmented such that it stores the label $M_{i,Q_i[k]}$ along with $Q_i[k]$, and returns this in a query. The *large frequency* data structure is augmented such that for every update of $T_i[k]$ we store the label that caused the update. Formally, let $a > 0$ be the first index such that $T_{i+a}[k] \ne T_i[k]$. Next to $T_i[k]$ we store the label $L_i[k] = A[i + a - 1]$. In a query, $[i, j]$, let s be the index of the successor of j in T_i computed as above. If $s > 1$ we return the label $L_i[s-1]$, and if $s=1$ we return $M_{i,Q_i[\lceil 1/\varepsilon' \rceil]}$, which is stored in the small frequency data structure.

In the case where $s=1$ we know that $\lceil \frac{1}{\varepsilon'} \rceil \leq F_{i,j} \leq \lceil \frac{1}{\varepsilon'}(1+\varepsilon')^2 \rceil - 1 = \lceil \frac{1}{\varepsilon'}(1+\varepsilon) \rceil - 1$ and we know that the frequency of $M_{i,Q_i[\lceil 1/\varepsilon' \rceil]}$ in $A[i,j]$ is at least $\lceil \frac{1}{\varepsilon'} \rceil$. We conclude that the frequency of $M_{i,Q_i[\lceil 1/\varepsilon' \rceil]}$ in $A[i,j]$ is a $(1+\varepsilon)$-approximation of $F_{i,j}$.

If $s>1$ we know that $\lceil \frac{1}{\varepsilon'}(1+\varepsilon')^{s-1} \rceil + 1 \leq F_{i,j} \leq \lceil \frac{1}{\varepsilon'}(1+\varepsilon')^{s+1} \rceil - 1$ by equation (1), and that the frequency, f_L, of the label $L_i[s-1]$ in $A[i,j]$ is at least $\lceil \frac{1}{\varepsilon'}(1+\varepsilon')^{s-1} \rceil + 1$. This means that $F_{i,j} \leq \frac{1}{\varepsilon'}(1+\varepsilon')^{s+1} \leq (1+\varepsilon')^2 f_L = (1+\varepsilon) f_L$, and we conclude that f_L is a $(1+\varepsilon)$-approximation of $F_{i,j}$.

The space needed for this data structure is $O(\frac{n}{\varepsilon'}) = O(\frac{n(\sqrt{1+\varepsilon}+1)}{\varepsilon}) = O(\frac{n}{\varepsilon})$, and a query takes $O(\log \frac{1}{\varepsilon'}) = O(\log \frac{1}{\varepsilon} + \log(\sqrt{1+\varepsilon}+1)) = O(\log \frac{1}{\varepsilon})$ time.

References

1. Yao, A.C.C.: Should tables be sorted? J. ACM **28**(3) (1981) 615–628
2. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. Nord. J. Comput. **12**(1) (2005) 1–17
3. Petersen, H.: Improved bounds for range mode and range median queries. In: Proc. 34th Conference on Current Trends in Theory and Practice of Computer Science. (2008) 418–423
4. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. Inf. Process. Lett. **109**(4) (2008) 225–228
5. Bose, P., Kranakis, E., Morin, P., Tang, Y.: Approximate range mode and range median queries. In: Proc. 22nd Symposium on Theoretical Aspects of Computer Science. (2005) 377–388
6. Patrascu, M., Thorup, M.: Higher lower bounds for near-neighbor and further rich problems. In: Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science. (2006) 646–654
7. Pătraşcu, M.: (Data) STRUCTURES. In: Proc. 49th Annual IEEE Symposium on Foundations of Computer Science. (2008) 434–443
8. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: Proc. 39th ACM Symposium on Theory of Computing. (2007) 40–46
9. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Proc. 15th International Symposium on Algorithms and Computation. (2004) 558–568
10. Afshani, P.: On dominance reporting in 3D. In: Proc. of the 16th Annual European Symposium on Algorithms. (2008) 41–51
11. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space theta(n). Inf. Process. Lett. **17**(2) (1983) 81–84
12. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. J. Comput. Syst. Sci. **57**(1) (1998) 37–49
13. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2) (1984) 338–355
14. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. Journal of Computer and System Sciences **38**(1) (1989) 86–124
15. Jacobson, G.J.: Succinct static data structures. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1988)