Linear-Space Data Structures for Range Mode Query in Arrays*

Timothy M. Chan[1], Stephane Durocher[2], Kasper Green Larsen[3], Jason Morrison[2], and Bryan T. Wilkinson[1]

1 **University of Waterloo, Waterloo, Canada,** tmchan@uwaterloo.ca **and** b3wilkin@uwaterloo.ca
2 **University of Manitoba, Winnipeg, Canada,** durocher@cs.umanitoba.ca **and** jason_morrison@umanitoba.ca
3 **Aarhus University, Aarhus, Denmark,** larsen@cs.au.dk

Abstract

A mode of a multiset S is an element $a \in S$ of maximum multiplicity; that is, a occurs at least as frequently as any other element in S. Given an array $A[1:n]$ of n elements, we consider a basic problem: constructing a static data structure that efficiently answers range mode queries on A. Each query consists of an input pair of indices (i, j) for which a mode of $A[i:j]$ must be returned. The best previous data structure with linear space, by Krizanc, Morin, and Smid (ISAAC 2003), requires $O(\sqrt{n} \log \log n)$ query time. We improve their result and present an $O(n)$-space data structure that supports range mode queries in $O(\sqrt{n/\log n})$ worst-case time. Furthermore, we present strong evidence that a query time significantly below $\sqrt{n}$ cannot be achieved by purely *combinatorial* techniques; we show that boolean matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. Additionally, we give linear-space data structures for orthogonal range mode in higher dimensions (queries in near $O(n^{1-1/2d})$ time) and for halfspace range mode in higher dimensions (queries in $O(n^{1-1/d^2})$ time).

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases mode, range query, data structure, linear space, array

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.290

1 Introduction

The *frequency* of an element x in a multiset S, denoted $\text{freq}_S(x)$, is the number of occurrences (i.e., the multiplicity) of x in S. A *mode* of S is an element $a \in S$ such that for all $x \in S$, $\text{freq}_S(x) \leq \text{freq}_S(a)$. A multiset S may have multiple distinct modes; the frequency of the modes of S, denoted by m, is unique.

Along with the mean and median, the mode is a fundamental statistic in data analysis. Given a sequence of n elements ordered in a list A, a range query seeks to compute the corresponding statistic on the multiset determined by a subinterval of the list: $A[i:j]$. The objective is to preprocess A to construct a data structure that supports efficient response to one or more subsequent range queries, where the corresponding input parameters (i, j) are provided at query time. Such a data structure is useful as it allows us to report statistics over any window of a given sequence of data.

* Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and in part by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 290–301

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We assume the standard RAM model of computation with word size $w = \Omega(\log n)$. Although the complete set of possible queries can be precomputed and stored using $\Theta(n^2)$ space, practical data structures require less storage while still enabling efficient response time. For all i, if $i = j$, then a range query must report $A[i]$. Consequently, any range query data structure for a list of n items requires $\Omega(n)$ storage space in the worst case [2]. This leads to a natural question: how quickly can an $O(n)$-space data structure answer range queries? A range mean query is equivalent to a normalized range sum query (partial sum query), for which a precomputed prefix-sum array provides a linear-space static data structure with constant query time [16]. Range median queries have been analyzed extensively in recent years and are closely related to range counting, where efficient data structures are now known (with linear space and logarithmic or slightly sublogarithmic query time) [2, 3, 5, 10, 11, 14, 16, 20, 21]. In contrast, range mode queries appear more challenging than range mean and median. As expressed recently by Brodal et al. [3, page 2]: "The problem of finding the most frequent element within a given array range is still rather open."

The best previous linear-space data structure for range mode query was by Krizanc et al. [15, 16], who obtained a query time of $O(\sqrt{n}\log\log n)$.[1] No better approaches have been discovered in the intervening eight years, which leads one to suspect that a $\sqrt{n}$-type bound might be the best one could hope for.

Indeed, we present strong evidence that purely combinatorial approaches cannot avoid the $\sqrt{n}$ effect in the preprocessing or query costs, up to polylogarithmic factors. (Krizanc et al.'s method has near $n^{3/2}$ preprocessing time.) More specifically, we show in Section 7 that boolean matrix multiplication (matrix multiplication on $\{0,1\}$-matrices with addition and multiplication replaced by OR and AND, respectively) of two $\sqrt{n}\times\sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. This reduction implies that any data structure for range mode must have either $\Omega(n^{\omega/2})$ preprocessing time or $\Omega(n^{\omega/2-1})$ query time in the worst case, where ω denotes the matrix multiplication exponent. Since the current best matrix multiplication algorithm has exponent 2.3727 [23], we cannot obtain preprocessing time better than $n^{1.18635}$ and query time better than $n^{0.18635}$ simultaneously with current knowledge. Moreover, since the current best *combinatorial* algorithm for boolean matrix multiplication (which avoids algebraic techniques as in Strassen's) has running time only a polylogarithmic factor better than cubic [1], we cannot obtain preprocessing time better than $n^{3/2}$ and query time better than $\sqrt{n}$ simultaneously by purely combinatorial techniques with current knowledge, except for polylogarithmic-factor speedups.

In view of the above hardness result, it is therefore worthwhile to pursue more modest improvements for the range mode problem. Notably, can the extra $\log\log$ factor in Krizanc et al.'s bound be eliminated?

In Section 3, we give a data structure that accomplishes just that: with $O(n)$ space, we can answer range mode queries in $O(\sqrt{n})$ time. The data structure is based on—and in some ways simplifies—Krizanc et al.'s, since we use only rudimentary structures (mostly arrays), without van Emde Boas trees or repeated binary searches.

In fact, we go beyond eliminating a mere $\log\log$ factor: in Section 6, we present an $O(n)$-space data structure that answers range mode queries in $o(\sqrt{n})$ time. The precise worst-case time bound is $O(\sqrt{n/w}) \subseteq O(\sqrt{n/\log n})$. As one might guess, bit packing tricks are used to achieve the speedup, but in addition we need a nontrivial combination of ideas, including

[1] The original data structure described by Krizanc et al. [16] supports queries in $O(\sqrt{n}\log n)$ time. As they remarked, this time can be reduced to $O(\sqrt{n}\log\log n)$ by using van Emde Boas trees for predecessor search [22].

partitioning elements into two sets (one with small maximum frequency and another with a small number of distinct elements), each handled by a different method, and an interesting application of rank/select data structures (from the world of succinct data structures).

In Section 8, we consider a natural higher-dimensional generalization of the problem: given a set of coloured points in $\mathbb{R}^d$, support queries for the most frequently occurring colour in some query range. We obtain the first nontrivial results for this geometric problem. For example, for orthogonal ranges, we give a near-linear space data structure that supports queries in near $O(n^{1-1/2d})$ time. For halfspace ranges, we give a linear-space data structure that supports queries in $O(n^{1-1/d^2})$ time. This latter result is obtained using an interesting application of geometric *cuttings* [7], in addition to standard range searching data structures.

Throughout the paper, let m denote the maximum frequency (i.e., the mode of the overall array), and let Δ denote the number of distinct elements ($m, \Delta \leq n$).

2 Related Work

Computing a Mode. The mode of a multiset S of n items can be found in $O(n \log n)$ time by sorting S and scanning the sorted list to identify the longest sequence of identical elements. By reduction from element uniqueness, a matching $\Omega(n \log n)$ lower bound in the comparison model follows. Better bounds on the worst-case time can be obtained by parameterizing in terms of m or Δ. A worst-case time of $O(n \log \Delta)$ is easily achieved by inserting the n elements into a balanced search tree in which each node stores a key and its frequency. Munro and Spira [19] described an $O(n \log(n/m))$-time algorithm and a matching comparison-based lower bound. On the word RAM model, the mode can be computed in linear expected time by hashing.

Range Mode Query. As mentioned, a data structure of Krizanc et al. [16] requires linear space and provides $O(\sqrt{n} \log \log n)$ query time. Krizanc et al. also considered larger-space structures. They described data structures that provide constant-time queries using $O(n^2 \log \log n / \log n)$ space and $O(n^\epsilon \log n)$-time queries using $O(n^{2-2\epsilon})$ space, for any fixed $\epsilon \in (0, 1/2]$. Petersen and Grabowski [21] improved the first bound to constant time and $O(n^2 \log \log n / \log^2 n)$ space and Petersen [20] improved the second bound to $O(n^\epsilon)$-time queries using $O(n^{2-2\epsilon})$ space, for any fixed $\epsilon \in [0, 1/2)$. Although its space requirement is almost linear in n as ϵ approaches $1/2$, the data structure of Petersen [20] requires $\omega(n)$ space (the number of levels in a hierarchical set of tables and hash functions approaches ∞ as $\epsilon \to 1/2$). Our new approach can also lead to improved space-time tradeoffs (see the statement of Theorem 7 with the parameter $s = n^{1-\epsilon}$): we can obtain $O(n^\epsilon)$ query time with $O(n^{2-2\epsilon} / \log n)$ space for any fixed $\epsilon \in [0, 1/2]$. This improves Petersen's result (though for $\epsilon = 0$, Petersen and Grabowski's result remains slightly better). Finally, Greve et al. [12] prove a lower bound of $\Omega(\log n / \log(s \cdot w/n))$ query time for any data structure that uses s memory cells of w bits in the cell probe model.

Other Query Problems. Bose et al. [2] considered approximate range mode queries, in which the objective is to return an element whose frequency is at least αm. They gave a data structure that requires $O(n/(1-\alpha))$ space and answers approximate range mode queries in $O(\log \log_{1/\alpha} n)$ time for any fixed $\alpha \in (0, 1)$, as well as data structures that provide constant-time queries for $\alpha \in \{1/2, 1/3, 1/4\}$, using space $O(n \log n)$, $O(n \log \log n)$, and $O(n)$, respectively. Greve et al. [12] gave data structures that support approximate range mode queries in $O(1)$ time using $O(n)$ space for $\alpha = 1/3$, and $O(\log(\alpha/(1-\alpha)))$ time using $O(n\alpha/(1-\alpha))$ space for any fixed $\alpha \in [1/2, 1)$.

Durocher et al. [9] described an $O(n)$-space data structure that supports constant-time

range majority queries; this data structure is then extended to range α-majority queries, a generalization of range majority.

3 First Method: $O(\sqrt{n})$ Query Time and $O(n)$ Space

We begin by presenting a linear-space data structure with $O(\sqrt{n})$ query time, improving Krizanc et al.'s result [16] by a $\log\log$ factor. We build on the data structure of Krizanc et al. and introduce a different technique that avoids the need for predecessor search. We will actually establish the following time-space tradeoff—the linear-space result follows by setting the parameter $s = \lceil\sqrt{n}\rceil$.

▶ **Theorem 1.** *Given an array $A[1:n]$ and any fixed value $s \in [1,n]$, there exists a data structure requiring $O(n+s^2)$ space that supports range mode queries on A in $O(n/s)$ time.*

The following observation will be useful:

▶ **Lemma 2** (Krizanc et al. [16])**.** *Let A and B be any multisets. If c is a mode of $A \cup B$ and $c \notin A$, then c is a mode of B.*

Data Structure Precomputation. Given input array $A[1:n]$, let D denote the set of distinct elements stored in A and assume some arbitrary ordering on the elements. We first apply rank space reduction: construct an array $B[1:n]$ such that for each i, $B[i]$ stores the rank of $A[i]$ in D. Here, $B[i] \in \{1,\ldots,\Delta\}$. For any a, i, and j, $B[a]$ is a mode of $B[i:j]$ if and only if $A[a]$ is a mode of $A[i:j]$. For simplicity, we describe our data structures in terms of array B; a table look-up provides a direct bijective mapping from $\{1,\ldots,\Delta\}$ to D. Set D, array B, and the value Δ are independent of any query range and can be computed in $O(n\log\Delta)$ time during preprocessing.

For each $a \in \{1,\ldots,\Delta\}$, let $Q_a = \{b \mid B[b] = a\}$. That is, Q_a is the set of indices b such that $B[b] = a$. For any a, a range counting query for element a in $B[i:j]$ can be answered by searching for the predecessors of i and j, respectively, in the set Q_a; the difference of their indices is the frequency of a in $B[i:j]$ [16]. Such a range counting query can be implemented using an efficient predecessor data structure in $\Theta(\log\log n)$ time in the worst case (e.g., [22]).

The following related decision problem, however, can be answered in *constant* time by a linear-space data structure: does $B[i:j]$ contain at least q instances of element $B[i]$? This question can be answered by a "select" query that returns the index of the qth instance of $B[i]$ in $B[i:n]$. For each $a \in \{1,\ldots,\Delta\}$, store the set Q_a as an ordered array (also denoted Q_a for simplicity). Define a rank array $B'[1:n]$ such that for all b, $B'[b]$ denotes the rank (i.e., the index) of b in $Q_{B[b]}$. Given any q, i, and j, to determine whether $B[i:j]$ contains at least q instances of $B[i]$ it suffices to check whether $Q_{B[i]}[B'[i]+q-1] \le j$. Since array $Q_{B[i]}$ stores the sequence of indices of instances of element $B[i]$ in B, looking ahead $q-1$ positions in $Q_{B[i]}$ returns the index of the qth occurrence of element $B[i]$ in $B[i:n]$; if this index is at most j, then the frequency of $B[i]$ in $B[i:j]$ is at least q. If the index $B'[i]+q-1$ exceeds the size of the array $Q_{B[i]}$, then the query returns a negative answer. This gives the following lemma:

▶ **Lemma 3.** *Given an array $A[1:n]$, there exists a data structure requiring $O(n)$ space that can determine in constant time for any $0 \le i \le j \le n$ and any q whether $A[i:j]$ contains at least q instances of element $A[i]$.*

Following Krizanc et al. [16], given any $s \in [1,n]$ we partition array B into s blocks of size $t = \lceil n/s\rceil$. That is, for each $i \in \{0,\ldots,s-2\}$, the ith block spans $B[i\cdot t+1:(i+1)t]$

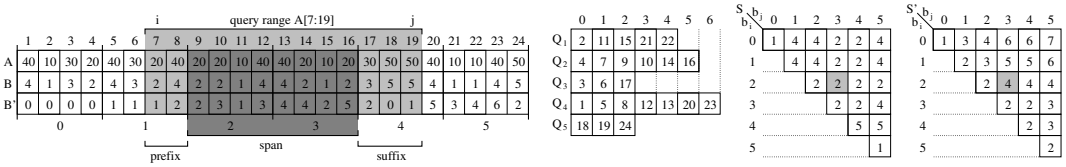

Figure 1 The array has size $n = 24$ (of which $\Delta = 5$ are distinct), partitioned into $s = 6$ blocks of size $t = 4$. The query range is $A[i : j] = A[7 : 19]$, for which the unique mode is 20, occurring with frequency 5. The corresponding mode of $B[i : j]$ is 2. The query range is partitioned into the prefix $B[7 : 8]$, the span $B[9 : 16]$, and the suffix $B[17 : 19]$. The span covers blocks $b_i = 2$ to $b_j = 3$, for which the corresponding mode is $S[2, 3] = 2$, occurring with frequency $S'[2, 3] = 4$.

and the last block spans $B[(s-1)t+1 : n]$. We precompute tables $S[0 : s-1, 0 : s-1]$ and $S'[0 : s-1, 0 : s-1]$, each of size $\Theta(s^2)$, such that for any $0 \le b_i \le b_j < s$, $S[b_i, b_j]$ stores a mode of $B[b_i t + 1 : (b_j + 1)t]$ and $S'[b_i, b_j]$ stores the corresponding frequency.

The arrays $Q_1, \ldots, Q_\Delta$ can be constructed in $O(n)$ total time in a single scan of array B. The arrays S and S' (which we call the *mode table*) can be constructed in $O(n \cdot s)$ time by scanning array B s times, computing one row of each array S and S' per scan. Thus, the total precomputation time required to initialize the data structure is $O(n \cdot s)$.

Query Algorithm. Given a query range $B[i : j]$, let $b_i = \lceil (i-1)/t \rceil$ and $b_j = \lfloor j/t \rfloor - 1$ denote the respective indices of the first and last blocks completely contained within $B[i : j]$. We refer to $B[b_i t + 1 : (b_j + 1)t]$ as the *span* of the query range, to $B[i : \min\{b_i t, j\}]$ as its *prefix*, and to $B[\max\{(b_j + 1)t + 1, i\} : j]$ as its *suffix*. One or more of the prefix, span, and suffix may be empty; in particular, if $b_i > b_j$, then the span is empty. See Figure 1.

The value $c = S[b_i, b_j]$ is a mode of the span with frequency $f_c = S'[b_i, b_j]$. If the span is empty, then let $f_c = 0$. By Lemma 2, either c is a mode of $B[i : j]$ or some element of the prefix or suffix is a mode of $B[i : j]$. Thus, to find a mode of $B[i : j]$, we verify for every element in the prefix and suffix whether its frequency in $B[i : j]$ exceeds f_c and, if so, we identify this element as a *candidate* mode and count its additional occurrences in $B[i : j]$.

We now describe how to compute the frequency of all candidate elements in the prefix, storing the value and frequency of the current best candidate in c and f_c; an analogous procedure is applied to the suffix. Sequentially scan the items in the prefix starting at the leftmost index, i, and let x denote the index of the current item. If $Q_{B[x]}[B'[x] - 1] \ge i$, then an instance of element $B[x]$ appears in $B[i : x-1]$, and its frequency has been counted already; in this case, simply skip $B[x]$ and increment x. Otherwise, check whether the frequency of $B[x]$ in $B[i : j]$ (which is equivalent to the frequency of $B[x]$ in $B[x : j]$) is at least f_c by Lemma 3 (i.e., by testing whether $Q_{B[x]}[B'[x] + f_c - 1] \le j$). If not, we again skip $B[x]$. Otherwise, $B[x]$ is a candidate, and the exact frequency of $B[x]$ in $B[i : j]$ can be counted by a linear scan[2] of $Q_{B[x]}$, starting at index $B'[x] + f_c - 1$ and terminating upon reaching either an index y such that $Q_{B[x]}[y] > j$ or the end of array $Q_{B[x]}$ (i.e., $y = |Q_{B[x]}| + 1$). That is, $Q_{B[x]}[y]$ denotes the index of the first instance of element $B[x]$ that lies beyond the query range $B[i : j]$ (or no such element exists). Consequently, the frequency of $B[x]$ in $B[i : j]$ is $f_x = y - B'[x]$. Update the current best candidate: $c \leftarrow B[x]$ and $f_c \leftarrow f_x$.

After all elements in the prefix and suffix have been processed, a mode of $B[i : j]$ and its frequency are stored in c and f_c, respectively.

[2] Although the time required to complete a linear scan could be reduced by instead using a binary search or a more efficient predecessor data structure, the worst-case time remains unchanged; for simplicity, a linear scan suffices.

Analysis. Excluding the linear scans of $Q_{B[x]}$, the query cost is clearly bounded by $O(t)$. For each candidate $B[x]$ encountered during the processing of the prefix, the cost of the linear scan of $Q_{B[x]}$ is $O(f_x - f_c)$. Since f_c is at least the frequency of the mode of the span, at least $f_x - f_c$ instances of $B[x]$ must occur in the prefix or suffix. We can thus charge the cost of the scan to these instances. Since each element $B[x]$ is considered a candidate at most once (during its first appearance) in the prefix, we conclude that the total cost of all the linear scans is proportional to the total number of elements in the prefix, i.e., $O(t)$. An analogous argument holds for the cost of processing the suffix. Therefore, a range mode query requires $O(t) = O(n/s)$ total time. The data structure requires $O(n)$ space to store the arrays A, B, and B', $O(n)$ total space to store the arrays $Q_1, \ldots, Q_\Delta$, and $O(s^2)$ space to store the tables S and S'. This proves Theorem 1.

4 Second Method: $O(\sqrt{n/w})$ Query Time and $O(n)$ Space When $m \leq \sqrt{nw}$

Our second method is a refinement of the first method (from Section 3), in which we store the mode table (S and S') more compactly by an encoding scheme that enables efficient retrieval of the relevant information, using techniques from succinct data structures, specifically, for *rank/select* operations. We show how to reduce a query to four rank/select operations. These new ideas allow us to improve the space bound in Theorem 1 by a factor of w, which enables us to use a slightly larger number of blocks, s, which in turn leads to an improved query time. However, there is one important caveat: our space-saving technique only works when the maximum frequency is small, namely, when $m \leq s$. Specifically, we will prove the following theorem in this section: choosing $s = \lceil \sqrt{nw} \rceil$ gives $O(n)$ space and $O(\sqrt{n/w})$ query time for $m \leq \sqrt{nw}$.

▶ **Theorem 4.** *Given an array $A[1:n]$ and any fixed $s \in [1, n]$ such that $m \leq s$ (where m is the frequency of the overall mode), there exists a data structure requiring $O(n + s^2/w)$ space that supports range mode queries on A in $O(n/s)$ time.*

Modified Data Structure. Recall that for a span from block b_i to block b_j, the mode table stores a mode of the span and its frequency in $S[b_i, b_j]$ and $S'[b_i, b_j]$, respectively. As we will show, a mode of the span can be computed efficiently if its frequency is known; consequently, we omit table S. Also, instead of storing the frequency of the mode explicitly, we store column-to-column frequency deltas (i.e., differences of adjacent frequency values); observe that frequency values are monotone increasing across each row. We encode the frequency deltas for a single row as a bit string, where a zero bit represents an increment in the frequency of the mode (i.e., each frequency delta is encoded in unary) and a one bit represents a former cell boundary. In any row, the number of ones is at most the number of blocks, s, and the number of zeroes is at most $m \leq s$. Precompute a data structure that uses a linear number of bits to support $O(1)$-time binary *rank* and *select* operations on each row (e.g., see [18]):[3] given a binary string, for each $a \in \{0, 1\}$, $\text{rank}_a(i)$ returns the number of times a occurs in the first i positions of the string, and $\text{select}_a(i)$ returns the position of the ith occurrence of a in the string. Thus, each row of the table uses $O(s)$ bits of space. The table has s rows and requires $O(s^2)$ bits of space in total. We pack these bits into words, resulting in an $O(s^2/w)$-space data structure.

[3] Succinct data structures can ensure that space usage is very close to the length of the bit string up to lower-order terms, but this fact is not needed in our application.

Modified Query Algorithm. Assuming we know a mode of the span and its frequency, we can process the prefix and suffix ranges in $O(t)$ time as before. Our attention turns now to determining a mode of the span and its frequency. We first obtain the frequency of the mode of the span in $O(1)$ time using rank and select queries on the bit string of the b_ith row:

$$pos_{b_j} \leftarrow \text{select}_1(b_j - b_i + 1), \qquad \text{and} \qquad freq \leftarrow \text{rank}_0(pos_{b_j}).$$

Having found the frequency of the mode, identifying a mode itself is still a tricky problem. We proceed in two steps. We first determine the block in which the last occurrence of a mode lies, in $O(1)$ time, as follows:

$$pos_{\text{last}} \leftarrow \text{select}_0(freq), \qquad \text{and} \qquad b_{\text{last}} \leftarrow \text{rank}_1(pos_{\text{last}}) + b_i.$$

Next we find a mode of the span by iteratively examining each element in block b_{last}, using a technique analogous to that for processing a suffix from Section 3. By Lemma 3 (reversed with $j \leq i$), we can check whether each element $B[x]$ in b_{last} has frequency *freq* in $B[b_i t + 1 : x]$, in $O(1)$ time per element. If the mode occurs multiple times in block b_{last}, its last occurrence will be successfully identified. Processing block b_{last} requires $O(t)$ total time. We conclude that the total query time is $O(t) = O(n/s)$ time. This proves Theorem 4.

5 Third Method: $O(\Delta)$ Query Time and $O(n)$ Space

In this section, we take a quick detour and consider a third method that has query time sensitive to Δ, the number of distinct elements; this "detour" turns out to be essential in assembling our final solution. We show the following:

▶ **Theorem 5.** *Given an array $A[1 : n]$, there exists a data structure requiring $O(n)$ space that supports range mode queries on A in $O(\Delta)$ time, where Δ denotes the number of distinct elements in A.*

The proof is simple: to answer a range mode query, the approach is to compute the frequency (in the query range) for each of the Δ possible elements explicitly, and then just compute the maximum in $O(\Delta)$ time.

Data Structure Precomputation. As before, we work with the array B by rank space reduction. This time, we divide B into blocks of size $t = \Delta$. For each $i \in \{1, \ldots, \lfloor n/\Delta \rfloor\}$, and for every $x \in \{1, \ldots, \Delta\}$, store the frequency $C_i[x]$ of x in the range $B[1 : i\Delta]$. The total size of all these *frequency tables* is $O((n/\Delta) \cdot \Delta) = O(n)$. The preprocessing time required is $O(n)$ (or $O(n \log \Delta)$ time if Δ or B must be computed).

Query Algorithm. Given a query range $B[i : j]$, as mentioned, it suffices to compute the frequency of x in $B[i : j]$ for every $x \in \{1, \ldots, \Delta\}$.

Let $b_j = \lfloor j/\Delta \rfloor - 1$. We can compute the frequency $C(x)$ of x in the suffix $B[b_j\Delta + 1 : j]$ for every $x \in \{1, \ldots, \Delta\}$ by a linear scan, in $O(\Delta)$ time since the suffix has size at most Δ. Then the frequency of x in $B[1, j]$ is given by $C_{b_j}[x] + C(x)$. The frequency of x in $B[1, i]$ can be computed similarly. The frequency of x in $B[i, j]$ is just the difference of these two numbers. The total query time is clearly $O(\Delta)$. This proves Theorem 5.

6 Final Method: $O(\sqrt{n/w})$ Query Time and $O(n)$ Space

We are finally ready to present our improved linear-space data structure with $O(\sqrt{n/w})$ query time. Our final idea is simple: if the elements all have small frequencies, the second

method (Section 4) already works well; otherwise, the number of distinct elements with large frequencies is small, and so the third method (Section 5) can be applied instead.

More precisely, let s be any fixed value in $[1, n]$. Partition the elements of A into those with *low* frequencies, i.e., at most s, and those with *high* frequencies, i.e., greater than s. A mode of the low-frequency elements has frequency at most s. Thus we can apply Theorem 4 to build an $O(n + s^2/w)$-space range mode query data structure on the low-frequency elements to support $O(n/s)$ query time. On the other hand, there are at most n/s distinct high-frequency elements. Thus we can apply Theorem 5 to build an $O(n)$-space range mode query data structure on the high-frequency elements to support $O(n/s)$ query time. The following simple decomposition lemma allows us to combine the two structures:

▶ **Lemma 6.** *Given an array $A[1 : n]$ and any ordered partition of A into two arrays $B_1[1 : n']$ and $B_2[1 : n - n']$ such that no element in B_1 occurs in B_2 nor vice versa, if there exist respective $s_1(n)$- and $s_2(n)$-space data structures that support range mode queries on B_1 and B_2 in $t_1(n)$ and $t_2(n)$ time, then there exists an $O(n + s_1(n) + s_2(n))$-space data structure that supports range mode query on A in $O(t_1(n) + t_2(n))$ time.*

Proof. For each $a \in \{1, 2\}$ and $i \in \{1, \ldots, n\}$, precompute $I_a[i]$, the index in the B_a array of the first element in A to the right of $A[i]$ that lies in B_a; and precompute $J_a[i]$, the index in the B_a array of the first element in A to the left of $A[i]$ that lies in B_a. Given a range query $A[i : j]$, we can compute the mode in the range $B_1[I_1[i], J_1[j]]$ and the mode in the range $B_2[I_2[i], J_2[j]]$ and determine which has larger frequency; this is a mode of $A[i : j]$. ◀

We have thus completed the proof of our main theorem:

▶ **Theorem 7.** *Given an array $A[1 : n]$ and any fixed $s \in [1, n]$, there exists a data structure requiring $O(n + s^2/w)$ space that supports range mode queries on A in $O(n/s)$ time. In particular, by setting $s = \lceil \sqrt{nw} \rceil$, there exists a data structure requiring $O(n)$ space that supports range mode queries on A in $O(\sqrt{n/w})$ time.*

7 Boolean Matrix Multiplication and Range Mode

In this section, we show that boolean matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. Greve et al. [12] observe the following:

▶ Observation 8 (Greve et al. [12]). Let S be a multiset whose elements belong to a universe U. Adding one of each element in U to S increases the frequency of the mode of S by one.

▶ Observation 9 (Greve et al. [12]). Let S_1 and S_2 be two sets (not multisets) and let S be the multiset union of S_1 and S_2. The frequency of the mode of S is one if $S_1 \cap S_2 = \emptyset$ and it is two if $S_1 \cap S_2 \neq \emptyset$.

Now let A and B be two $\sqrt{n} \times \sqrt{n}$ boolean matrices for which we are to compute the product $C = A \cdot B$. The entry $c_{i,j}$ in C must be 1 precisely if there exists at least one index k, where $1 \leq k \leq \sqrt{n}$, such that $a_{i,k} = b_{k,j} = 1$. Our goal is to determine whether this is the case using one range mode query for each entry $c_{i,j}$. Our first step in achieving this is to transform each row of A and each column of B into a set. For the ith row of A, we construct the set A_i containing all those indices k for which $a_{i,k} = 1$, i.e., $A_i = \{k \mid a_{i,k} = 1\}$. Similarly we let $B_j = \{k \mid b_{k,j} = 1\}$. Clearly $c_{i,j} = 1$ if and only if $A_i \cap B_j \neq \emptyset$. By Observation 9, this can be tested if we can determine the frequency of the mode in the multiset union of A_i and B_j. Our last step is thus to embed all the sets A_i and B_j into an array, such that we can use range mode queries to perform these intersection tests for every pair i, j. Our

constructed array M has two parts, a left part L and a right part R. The array M is then simply the concatenation of L and R. The array L represents all the sets A_i. It consists of $\sqrt{n}$ blocks of $\sqrt{n}$ entries. The ith block (entries $L[(i-1)\sqrt{n}+1 : i\sqrt{n}]$) represents the set A_i, and it consists of the elements $\{1,\ldots,\sqrt{n}\} \setminus A_i$ in some arbitrary order, followed by the elements of A_i in some arbitrary order. The array R similarly represents the sets B_j and it also consists of $\sqrt{n}$ blocks of $\sqrt{n}$ entries. The jth block represents the set B_j and it consists of the elements in B_j in some arbitrary order, followed by the elements $\{1,\ldots,\sqrt{n}\} \setminus B_j$ in some arbitrary order.

Now assume that $|A_i|$ and $|B_j|$ are known for each set A_i and B_j. We can now determine whether $A_i \cap B_j \neq \emptyset$ (i.e., whether $c_{i,j} = 1$) from the result of the range mode query on $M[\text{start}(i) : \text{end}(j)]$, where

$$\text{start}(i) = (i-1)\sqrt{n} + 1 + \sqrt{n} - |A_i| \qquad \text{and} \qquad \text{end}(j) = n + (j-1)\sqrt{n} + |B_j|.$$

To see this, first observe that $\text{start}(i)$ is the first index in M of the elements in A_i, and that $\text{end}(j)$ is the last index in M of the elements in B_j. In addition to a suffix of the block representing A_i and a prefix of the block representing B_j, the subarray $M[\text{start}(i) : \text{end}(j)]$ contains $\sqrt{n} - i$ complete blocks from L and $j-1$ complete blocks from R. Since a complete block contains all the elements $\{1,\ldots,\sqrt{n}\}$, it follows from Observations 8 and 9 that $A_i \cap B_j \neq \emptyset$ (i.e., $c_{i,j} = 1$) if and only if the frequency of the mode in $M[\text{start}(i), \text{end}(j)]$ is $2 + \sqrt{n} - i + j - 1$. The answer to the query $(\text{start}(i), \text{end}(j))$ thus allows us to determine whether $c_{i,j} = 1$ or 0. The array M and the values $|A_i|$ and $|B_j|$ can clearly be computed in linear time when given matrices A and B, thus we have the following result:

▶ **Theorem 10.** *Let $p(n)$ be the preprocessing time of a range mode data structure and $q(n)$ its query time. Then boolean matrix multiplication on two $\sqrt{n} \times \sqrt{n}$ matrices can be solved in time $O(p(n) + n \cdot q(n) + n)$.*

8 Higher Dimensions

We now consider generalizations of the range mode problem to Euclidean spaces of constant dimension d. Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, we consider the problem of constructing an efficient data structure to support queries that return a most frequently occurring colour in $P \cap Q$ for a query range $Q \subseteq \mathbb{R}^d$. We consider orthogonal range queries in Section 8.1 and halfspace range queries in Section 8.2.

8.1 Orthogonal Ranges

We generalize the technique of Krizanc et al. [16] by dividing space into s^d grid cells such that there are $O(n/s)$ points between any two consecutive parallel grid hyperplanes. The generalization of a span of a query range Q is the largest rectangle inside Q whose sides lie along grid hyperplanes. There are s^{2d} distinct spans and for each we precompute and store the mode of the span. This component of our data structure thus requires $O(s^{2d})$ space. For each set of points of a given colour, we also build an orthogonal range counting data structure [8] with polylogarithmic space overhead that answers queries in polylogarithmic time (see [13] for the best known solution, using $O(n(\log n/\log\log n)^{d-2})$ space and with $O((\log n/\log\log n)^{d-1})$ query time). Across all colours, these data structures require $O(n \operatorname{polylog} n)$ space.

Given a query hyperrectangle Q we use binary search amongst the grid hyperplanes in order to determine the slabs in which Q's sides lie. We then determine the mode of Q's span

in $O(1)$ time from our precomputed table. For each of the $2d$ sides of Q we must additionally consider each of the $O(n/s)$ points in the slab in which the side lies. For each such point, we count the number of points of its colour in Q using the range counting data structure of its colour in polylogarithmic time to find the actual mode. So, the running time of a query is $O\left(2d \cdot (n/s) \cdot \operatorname{polylog} n\right) = O\left((n/s) \cdot \operatorname{polylog} n\right)$ time.

▶ **Theorem 11.** *Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, and any fixed $s \in \{1, \ldots, n\}$, there exists a data structure requiring $O(n \operatorname{polylog} n + s^{2d})$ space that supports orthogonal range mode queries in $O\left((n/s) \cdot \operatorname{polylog} n\right)$ time. In particular, by setting $s = \lceil n^{1/2d} \rceil$, there exists a data structure requiring $O(n \operatorname{polylog} n)$ space that supports range mode queries in $O(n^{1-1/2d} \operatorname{polylog} n)$ time.*

Alternatively, we can guarantee $O(n)$ space if we increase the query time by an n^ϵ factor, by switching to a linear-space data structure for orthogonal range counting with $O(n^\epsilon)$ query time (by using a range tree [8] with n^ϵ fan-out).

8.2 Halfspace Ranges

We now consider halfspace range queries. We work in *dual* space [8], where the input is transformed into n hyperplanes, each assigned a colour, and a query halfspace is transformed into a point. A query for a dual point q returns the most frequently occurring colour amongst the hyperplanes that lie below q. Let $s \in \{1, \ldots, n\}$ be a fixed parameter specified by the user. We use the key concept of cuttings [7] from computational geometry. Given a set of n hyperplanes in $\mathbb{R}^d$, a $(1/r)$-*cutting* is a partition of $\mathbb{R}^d$ into simplicial cells such that each cell intersects at most n/r hyperplanes. The following is known [7, 6]:

▶ **Lemma 12.** *For any set of n hyperplanes in $\mathbb{R}^d$, there exists a $(1/r)$-cutting with $O(r^d)$ cells. Furthermore, there is a data structure for point location in the $(1/r)$-cutting, also requiring $O(r^d)$ space and answering queries in $O(\log r)$ time.*

We set $r = (n \cdot s^{d-1})^{1/d}$. For each cell γ in the cutting, we store the mode of the hyperplanes that lie strictly below γ. This component of our data structure requires $O(r^d) = O(n \cdot s^{d-1})$ space. In primal space, we build a simplex range reporting data structure [4, 17] for all of the points with $S = O(n \cdot s^{d-1})$ space. This data structure reports the k points in a query simplex in $O((n/S^{1/d}) \operatorname{polylog} n + k) = O((n/s)^{1-1/d} \operatorname{polylog} n + k)$ time. Also, for each colour i, we build a separate halfspace range counting data structure [4, 17] for the n_i points of colour i, with $S_i = O(n_i \cdot s^{d-1})$ space and $O(n_i/S_i^{1/d} + \log n_i) = O((n_i/s)^{1-1/d} + \log n)$ query time. The total space is $O(n \cdot s^{d-1})$.

Given a dual query point q, we first identify the cell γ of the $(1/r)$-cutting that contains q in $O(\log r)$ time. The mode of the hyperplanes below q is either the colour stored at cell γ or one of the colours of the hyperplanes intersecting γ. We can find the $O(n/r)$ hyperplanes intersecting γ by simplex range reporting in primal space in $O((n/s)^{1-1/d} \operatorname{polylog} n + n/r)$ time, since the set of all hyperplanes intersecting a simplex dualizes to a polyhedron of $O(1)$ size. For each hyperplane that intersects γ and lies below q, we perform a halfspace range counting query for the points of the colour of the hyperplane in primal space to determine the actual mode. The running time of this step is $O\left(\sum_{i=1}^{O(n/r)} (n_i/s)^{1-1/d} + (n/r) \log n\right)$. By Hölder's inequality, the sum in the first term is bounded by $O((n/r)^{1/d} \cdot (n/s)^{1-1/d}) = O((n/s)^{1-1/d^2})$ for $r = (n \cdot s^{d-1})^{1/d}$. The second term $(n/r) \log n = (n/s)^{1-1/d} \log n$ does not dominate except when $n/s = O(\operatorname{polylog} n)$.

▶ **Theorem 13.** *Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, and any fixed $s \in \{1, \ldots, n\}$, there exists a data structure requiring $O(n \cdot s^{d-1})$ space that supports halfspace range mode queries in $O((n/s)^{1-1/d^2} + \operatorname{polylog} n)$ time. In particular, by setting $s = 1$, there exists a data structure requiring $O(n)$ space that supports halfspace range mode queries in $O(n^{1-1/d^2})$ time.*

A similar approach works for other ranges (e.g., simplices, balls, and other constant-degree semialgebraic sets) by transforming query ranges to query points in a higher dimension, and using cuttings in this higher-dimensional space.

9 Discussion and Directions for Future Research

We close by mentioning a few interesting open problems. A useful generalization of the problem is to return the kth most frequently occurring element (or the k most frequent elements) in a query range. Due to its dependence on precomputed modes stored in array S, an analogous generalization of our methods (except for the third method) seems unlikely without a significant increase in space, if k is large.

▶ Open Problem 1. Construct an $O(n)$-space data structure for identifying the kth most frequently occurring element (or the k most frequent elements) in the range $A[i : j]$ in time $O(n^{1-\epsilon})$ (or $O(n^{1-\epsilon} + k)$) for some constant $\epsilon > 0$, where i, j, and k are given at query time.

We have given (near-)linear-space data structures for multiple variants of range mode, including orthogonal range mode for a d-dimensional point set and halfspace range mode for a d-dimensional point set. Our results, in various ways, build on and generalize the techniques of Krizanc et al. [16]. It is unknown whether there are entirely different approaches that can achieve smaller exponents on n in the query times.

▶ Open Problem 2. Is there a linear-space dynamic data structure for range mode in an array that supports queries and updates in $O(\sqrt{n} \operatorname{polylog} n)$ time? In the full paper we give respective dynamic data structures with $O(n)$ space and $O(n^{3/4} \operatorname{polylog} n)$ query and update times, and with $O(n^{4/3})$ space and $O(n^{2/3} \operatorname{polylog} n)$ query and update times.

▶ Open Problem 3. Is there a (near-)linear-space data structure for orthogonal range mode in $\mathbb{R}^d$ that supports queries in $o(n^{1-1/2d})$ time?

▶ Open Problem 4. Is there a linear-space data structure for halfspace range mode in $\mathbb{R}^d$ that supports queries in $o(n^{1-1/d^2})$ time?

Lastly, the following open problem is likely difficult since currently no techniques seem capable of proving unconditional super-polylogarithmic cell probe lower bounds:

▶ Open Problem 5. Prove a tight, unconditional lower bound on the worst-case query time required by any $O(n)$-space data structure that supports range mode queries on an array of n items.

Acknowledgements. The authors thank Peyman Afshani, Francisco Claude, Meng He, Ian Munro, Patrick Nicholson, Matthew Skala, and Norbert Zeh for discussing various topics related to range searching.

References

1 N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. IEEE FOCS*, pages 745–754, 2009.

2 P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proc. STACS*, volume 3404 of *LNCS*, pages 377–388. Springer, 2005.

3 G. S. Brodal, B. Gfeller, A. G. Jørgensen, and P. Sanders. Towards optimal range medians. *Theor. Comp. Sci.*, 412(24):2588–2601, 2011.

4 T. M. Chan. Optimal partition trees. In *Proc. ACM SoCG*, pages 1–10, 2010.

5 T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. ACM-SIAM SODA*, pages 161–173, 2010.

6 B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Disc. Comp. Geom.*, 9(2):145–158, 1993.

7 B. Chazelle. Cuttings. In *Handbook of Data Structures and Applications*, pages 25.1–25.10. CRC Press, 2005.

8 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 3rd edition, 2008.

9 S. Durocher, M. He, J. I. Munro, P. K. Nicholson, and M. Skala. Range majority in constant time and linear space. In *Proc. ICALP*, volume 6755 of *LNCS*, pages 244–255. Springer, 2011.

10 T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *Proc. SPIRE*, volume 5721 of *LNCS*, pages 1–6. Springer, 2009.

11 B. Gfeller and P. Sanders. Towards optimal range medians. In *Proc. ICALP*, volume 5555 of *LNCS*, pages 475–486. Springer, 2009.

12 M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proc. ICALP*, volume 6198 of *LNCS*, pages 605–616. Springer, 2010.

13 J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. ISAAC*, volume 3341 of *LNCS*, pages 558–568. Springer, 2004.

14 A. G. Jørgensen and K. D. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. ACM-SIAM SODA*, pages 805–813, 2011.

15 D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. In *Proc. ISAAC*, volume 2906 of *LNCS*, pages 517–526. Springer, 2003.

16 D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12:1–17, 2005.

17 J. Matoušek. Range searching with efficient hierarchical cuttings. *Disc. Comp. Geom.*, 10(2):157–182, 1993.

18 J. I. Munro. Tables. In V. Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *LNCS*, pages 37–42. Springer, 1996.

19 J. I. Munro and M. Spira. Sorting and searching in multisets. *SIAM J. Comp.*, 5(1):1–8, 1976.

20 H. Petersen. Improved bounds for range mode and range median queries. In *Proc. SOFSEM*, volume 4910 of *LNCS*, pages 418–423. Springer, 2008.

21 H. Petersen and S. Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Inf. Proc. Let.*, 109:225–228, 2009.

22 P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Proc. Let.*, 6(3):80–82, 1977.

23 V. Vassilevska Williams. Breaking the Coppersmith-Winograd barrier. http://www.cs.berkeley.edu/˜virgi/matrixmult.pdf, 2011.