Better Space Bounds for Parameterized Range Majority and Minority

Djamal Belazzougui[1], Travis Gagie[1,2], and Gonzalo Navarro[3]

[1] Department of Computer Science, University of Helsinki
[2] Helsinki Institute of Information Technology
[3] Department of Computer Science, University of Chile

Abstract. Karpinski and Nekrich (2008) introduced the problem of parameterized range majority, which asks to preprocess a string of length n such that, given the endpoints of a range, one can quickly find all the distinct elements whose relative frequencies in that range are more than a threshold τ. Subsequent authors have reduced their time and space bounds such that, when τ is given at preprocessing time, we need either $\mathcal{O}(n \lg(1/\tau))$ space and optimal $\mathcal{O}(1/\tau)$ query time or linear space and $\mathcal{O}((1/\tau) \lg \lg \sigma)$ query time, where σ is the alphabet size. In this paper we give the first linear-space solution with optimal $\mathcal{O}(1/\tau)$ query time. For the case when τ is given at query time, we significantly improve previous bounds, achieving either $\mathcal{O}(n \lg \lg \sigma)$ space and optimal $\mathcal{O}(1/\tau)$ query time or compressed space and $\mathcal{O}\big((1/\tau) \lg \frac{\lg(1/\tau)}{\lg \lg n}\big)$ query time. Along the way, we consider the complementary problem of parameterized range minority that was recently introduced by Chan et al. (2012), who achieved linear space and $\mathcal{O}(1/\tau)$ query time even for variable τ. We improve their solution to use either nearly optimally compressed space with no slowdown, or optimally compressed space with nearly no slowdown. Some of our intermediate results, such as density-sensitive query time for one-dimensional range counting, may be of independent interest.

1 Introduction

Finding frequent elements in a dataset is a fundamental operation in data mining. Finding the most frequent elements can be challenging when all the distinct elements have nearly equal frequencies and we do not have the resources to compute all their frequencies exactly. In some cases, however, we are interested in the most frequent elements only if they really are frequent. For example, Misra and Gries [20] showed how, given a string and a threshold τ with $0 < \tau \leq 1$, with two passes and $\mathcal{O}(1/\tau)$ words of space we can find all the distinct elements in a string whose relative frequencies are at least τ. These elements are called the τ-majorities of the string. Misra and Gries' algorithm was rediscovered by Demaine, López-Ortiz and Munro [9], who noted it can be made to run in $\mathcal{O}(1)$ time per element on a word RAM with $\Omega(\lg n)$-bit words, where n is the length of the string, which is the model we use; it was then rediscovered again by Karp, Shenker and Papadimitriou [16]. As Cormode and Muthukrishnan [8] put it, "papers on frequent items are a frequent item!"

Krizanc, Morin and Smid [18] introduced the problem of preprocessing the string such that later, given the endpoints of a range, we can quickly return the mode of that range (i.e., the most frequent element). They gave two solutions, one of which takes $\mathcal{O}\big(n^{2-2\epsilon}\big)$ space for any fixed positive $\epsilon \leq 1/2$, and answers queries in $\mathcal{O}(n^{\epsilon} \lg \lg n)$ time; the other takes $\mathcal{O}\big(n^2 \lg \lg n / \lg n\big)$ space and answers queries in $\mathcal{O}(1)$ time. Petersen [22] reduced Krizanc et al.'s first time bound to $\mathcal{O}(n^{\epsilon})$ for any fixed non-negative $\epsilon < 1/2$, and Petersen and Grabowski [23] reduced the second space bound to $\mathcal{O}\big(n^2 \lg \lg n / \lg^2 n\big)$. Chan et al. [6] recently gave a linear-space solution that answers queries in $\mathcal{O}\Big(\sqrt{n / \lg n}\Big)$ time. They also gave evidence suggesting we cannot easily achieve query time substantially smaller than $\sqrt{n}$ using linear space; however, the best known lower bound, by Greve et al. [15], says only that we cannot achieve query time $o\left(\lg(n) / \lg(sw/n)\right)$ using s words of w bits each. Because of the difficulty of supporting range mode queries, Bose et al. [5] and Greve et al. [15] considered the problem of approximate range mode, for which we are asked to return an element whose frequency is at least a constant fraction of the mode's frequency.

Karpinski and Nekrich [17] took a different direction, analogous to Misra and Gries' approach, when they introduced the problem of preprocessing the string such that later, given the endpoints of a range, we can quickly return the τ-majorities of that range. We refer to this problem as parameterized range majority. Assuming τ is given when we are preprocessing the string, they showed how we can store the string in $\mathcal{O}(n(1/\tau))$ space and answer queries in $\mathcal{O}\big((1/\tau)(\lg \lg n)^2\big)$ time. They also gave bounds for dynamic and higher-dimensional versions. Durocher et al. [10] independently posed the same problem and showed how we can store the string in $\mathcal{O}(n \lg(1/\tau + 1))$ space and answer queries in $\mathcal{O}(1/\tau)$ time. Notice that, because there can be up to $1/\tau$ distinct elements to return, this time bound is worst-case optimal. Gagie et al. [14] showed how to store the string in compressed space — i.e., $\mathcal{O}(n(H + 1))$ bits, where H is the entropy of the distribution of elements in the string — such that we can answer queries in $\mathcal{O}((1/\tau) \lg \lg n)$ time. They also showed how to drop the assumption that τ is fixed and simultaneously achieve optimal query time, at the cost of increasing the space bound by a $(\lg n)$-factor. That is, they gave a data structure that stores the string in $\mathcal{O}(n(H + 1))$ space such that later, given the endpoints of a range and τ, we can return the τ-majorities of that range in $\mathcal{O}(1/\tau)$ time. Chan et al. [7] recently gave another solution for variable τ, which also has $\mathcal{O}(1/\tau)$ query time but uses $\mathcal{O}(n \lg n)$ space. As far as we know, these are all the relevant bounds for Karpinski and Nekrich's original exact, static, one-dimensional problem, both for fixed and variable τ; they are summarized in Table 1 together with our own results. Related work includes Elmasry et al.'s [11] solution for the dynamic version and Lai, Poon and Shi's [19] and Wei and Yi's [26] approximate solutions for the dynamic version.

In this paper we first consider the complementary problem of parameterized range minority, which was recently introduced by Chan et al. [7]. For this problem we are asked to preprocess the string such that later, given the endpoints of a range, we can return (if one exists) a distinct element that occurs in that range

Table 1. Results for the problem of parameterized range majority on a string of length n over an alphabet of size σ in which the distribution of the elements has entropy H.

source	space	time	variable τ
Karpinski and Nekrich [17]	$\mathcal{O}(n(1/\tau))$ words	$\mathcal{O}\big((1/\tau)(\lg\lg n)^2\big)$	no
Durocher et al. [10]	$\mathcal{O}(n\lg(1/\tau))$ words	$\mathcal{O}(1/\tau)$	no
Gagie et al. [14]	$\mathcal{O}(n(H+1))$ bits	$\mathcal{O}((1/\tau)\lg\lg\sigma)$	no
Theorem 3	$\mathcal{O}(n)$ words	$\mathcal{O}(1/\tau)$	no
Gagie et al. [14]	$\mathcal{O}(n(H+1))$ words	$\mathcal{O}(1/\tau)$	yes
Chan et al. [7]	$\mathcal{O}(n\lg n)$ words	$\mathcal{O}(1/\tau)$	yes
Theorem 4	$\mathcal{O}(n\lg\lg\sigma)$ words	$\mathcal{O}(1/\tau)$	yes
Theorem 5	$nH + o(n\lg\sigma)$ bits	$\mathcal{O}((1/\tau)\lg\lg\sigma)$	yes
Theorem 7	$(1+\epsilon)nH + o(n\lg\sigma)$ bits	$\mathcal{O}\big((1/\tau)\lg\frac{\lg(1/\tau)}{\lg\lg n}\big)$	yes

but is not one of its τ-majorities. Such an element is called a τ-minority for the range. At first, finding a τ-minority might seem harder than finding a τ-majority because, e.g., we are less likely to find a τ-minority by sampling. Nevertheless, Chan et al. gave a linear-space solution with $\mathcal{O}(1/\tau)$ query time even when τ is given at query time. In Section 3 we give two results, also for the case of variable τ:

1. for any positive constant ϵ, a solution with $\mathcal{O}(1/\tau)$ query time that takes $(1+\epsilon)nH + \mathcal{O}(n)$ bits;
2. for any function $f(n) = \omega(1)$, a solution with $\mathcal{O}((1/\tau)\, f(n))$ query time that takes $nH + \mathcal{O}(n) + o(nH)$ bits.

In the full version of this paper we will reduce the space bound in point 2 above to $nH + o(n(H+1))$ bits. That is, we improve Chan et al.'s solution to use either nearly optimally compressed space with no slowdown, or optimally compressed space with nearly no slowdown. We reuse ideas from this section in our solutions for parameterized range majority.

In Section 4 we return to Karpinski and Nekrich's original problem of parameterized range majority with fixed τ and give the first linear-space solution with worst-case optimal $\mathcal{O}(1/\tau)$ query time. In Section 5 we adapt this solution to the more challenging case of variable τ and give three results:

1. a solution with $\mathcal{O}(1/\tau)$ query time that takes $\mathcal{O}(n\lg\lg\sigma)$ space, where σ is the size of the alphabet;
2. a solution with $\mathcal{O}((1/\tau)\lg\lg\sigma)$ query time that takes $nH + o(n\lg\sigma)$ bits;
3. for any positive constant ϵ, a solution with $\mathcal{O}\Big((1/\tau)\lg\frac{\lg(1/\tau)}{\lg\lg n}\Big)$ query time that takes $(1+\epsilon)nH + o(n\lg\sigma)$ bits.

With (2), we can support $\mathcal{O}(1)$-time access to the string and $\mathcal{O}(\lg\lg\sigma)$-time rank and select (see definitions in Section 2.1); with (3), select also takes $\mathcal{O}(1)$ time.

In the full version of this paper we will reduce the space bounds in (2) and (3) to $nH + o(n(H+1))$ and $(1+\epsilon)nH + \mathcal{O}(n)$ bits, respectively. While proving (3) we introduce a compressed data structure with density-sensitive query time for one-dimensional range counting, which may be of independent interest; due to space constraints, however, we leave the description of this data structure to the full version of this paper. We will also show in the full version how to use our data structures for (2) or (3) to find a range mode quickly when it is actually reasonably frequent. We leave as an open problem reducing the space bound in (1) or the time bound in (2) or (3), to obtain linear or compressed space with optimal query time.

2 Preliminaries

2.1 Access, select and (partial) rank

Let $S[1..n]$ be a string over an alphabet of size σ and let H be the entropy of the distribution of elements in S. An access query on S takes a position k and returns $S[k]$; a rank query takes a distinct element a and a position k and returns the number of occurrences of a in $S[1..k]$; a select query takes a distinct element a and a rank r and returns the position of the rth occurrence of a in S. A partial rank query is a rank query with the restriction that the given distinct element must occur in the given position; i.e., $S[k] = a$. These are among the most well-studied operations on strings, so we state here only the results most relevant to this paper.

For $\sigma = 2$ and any constant c, Pătraşcu [24] showed how we can store S in $nH + \mathcal{O}(n/\lg^c n)$ bits. For $\sigma = \lg^{\mathcal{O}(1)} n$, Ferragina et al. [12] showed how we can store S in $nH + o(n)$ bits and support access, rank and select in $\mathcal{O}(1)$ time. For $\sigma < n$, Barbay et al. [1] showed how, for any positive constant ϵ, we can store S in $(1+\epsilon)nH + o(n)$ bits and support access and select in $\mathcal{O}(1)$ time and rank in $\mathcal{O}(\lg\lg\sigma)$ time. Belazzougui and Navarro [3] showed how to support $\mathcal{O}(1)$-time partial rank using $\mathcal{O}(n(\lg H + 1))$ bits; in the full version of their paper [2] they reduced that space bound to $o(n)(H+1)$ bits. In another paper, Belazzougui and Navarro [4] showed how, for any function $f(n) = \omega(1)$, we can store S in $nH + o(n(H+1)$ bits and support access in $\mathcal{O}(1)$ time, select in $\mathcal{O}(f(n))$ time and rank in $\mathcal{O}(\lg\lg\sigma)$ time. They also proved, via a reduction from the predecessor problem, that we cannot support general rank queries in $o(\lg(\lg\sigma/\lg\lg n))$ time while using $n\lg^{\mathcal{O}(1)} n$ space.

2.2 Coloured range listing

Motivated by the problem of document listing, Muthukrishnan [21] showed how we can store $S[1..n]$ such that, given the endpoints of a range, we can quickly list the distinct elements in that range and the positions of their leftmost occurrences therein. This is the special case of one-dimensional coloured range listing in which the points' coordinates are the integers from 1 to n. Let $C[1..n]$ be the array in

which $C[k]$ is the position of the last occurrence of the distinct element $S[k]$ in $S[1..k-1]$ — i.e., the last occurrence before $S[k]$ itself — or 0 if there is no such occurrence. Notice $S[k]$ is the first occurrence of that distinct element in a range $S[i..j]$ if and only if $i \leq k \leq j$ and $C[k] < i$. We store C, implicitly or explicitly, and a data structure supporting $\mathcal{O}(1)$-time range-minimum queries on C that return the position of the leftmost occurrence of the minimum in the range.

To list the distinct elements in a range $S[i..j]$ given i and j, we find the position m of the leftmost occurrence of the minimum in the range $C[i..j]$; check whether $C[m] < i$; and, if so, output $S[m]$ and m and recurse on $C[i..m-1]$ and $C[m+1..j]$. This procedure is online — i.e., we can stop it early if we want only a certain number of distinct elements — and the time it takes per distinct element is $\mathcal{O}(1)$ plus the time to access C.

Suppose we already have data structures supporting access, select and partial rank queries on S, all in $\mathcal{O}(t)$ time. Notice $C[k] = S.\text{select}_{S[k]}\big(S.\text{rank}_{S[k]}(k) - 1\big)$, so we can also support access to C in $\mathcal{O}(t)$ time. Sadakane [25] and Fischer [13] gave $\mathcal{O}(n)$-bit data structures supporting $\mathcal{O}(1)$-time range-minimum queries. Therefore, we can implement Muthukrishnan's solution using $\mathcal{O}(n)$ extra bits such that it takes $\mathcal{O}(t)$ time per distinct element listed.

3 Parameterized Range Minority

Recall from Section 1 that a τ-minority for a range is a distinct element that occurs in that range but is not one of its τ-majorities. The problem of parameterized range minority is to preprocess a string such that later, given the endpoints of a range and τ, we can quickly return a τ-minority for that range if one exists. Chan et al. gave a linear-space solution with $\mathcal{O}(1/\tau)$ query time even for the case of variable τ. They first build a list of $\lfloor 1/\tau \rfloor + 1$ distinct elements that occur in the given range (or as many as there are, if fewer) and then check those elements' frequencies to see which are τ-minorities. There cannot be more than $\lfloor 1/\tau \rfloor$ τ-majorities so, if there exists a τ-minority for that range, then at least one must be in the list. In this section we show how to implement this idea using compressed space.

To support parameterized range minority on $S[1..n]$ in $\mathcal{O}(1/\tau)$ time, we store data structures supporting $\mathcal{O}(1)$-time access, select and partial rank queries on S and a data structure supporting $\mathcal{O}(1)$-time range-minimum queries on C. For any positive constant ϵ, we can store these data structures in a total of $(1+\epsilon)nH + \mathcal{O}(n)$ bits. Given τ and endpoints i and j, in $\mathcal{O}(1/\tau)$ time we use Muthukrishnan's algorithm to build a list of $\lfloor 1/\tau \rfloor + 1$ distinct elements that occur in $S[i..j]$ (or as many as there are, if fewer) and the positions of their leftmost occurrences therein. We check whether these distinct elements are τ-minorities using the following lemma:

Lemma 1. *Suppose we know the position of the leftmost occurrence of a distinct element in a range. We can check whether that distinct element is a τ-minority or a τ-majority using a partial rank query and a select query on S.*

Proof. Let k be the position of the first occurrence of a in $S[i..j]$. If $S[k]$ is the rth occurrence of a in S, then a is a τ-minority for $S[i..j]$ if and only if the $(r + \lceil \tau(j - i + 1) \rceil - 1)$th occurrence of a in S is strictly after $S[j]$; otherwise a is a τ-majority. That is, we can check whether a is a τ-minority for $S[i..j]$ by checking whether

$$S.\text{select}_a \left(S.\text{rank}_a(k) + \lceil \tau(j - i + 1) \rceil - 1 \right) > j \,;$$

since $S[k] = a$, computing $S.\text{rank}_a(k)$ is only a partial rank query. □

This gives us the following theorem, which improves Chan et al.'s solution to use nearly optimally compressed space with no slowdown.

Theorem 1. *For any positive constant ϵ, we can store S in $(1+\epsilon)nH+\mathcal{O}(n)$ bits such that later, given the endpoints of a range and τ, we can return a τ-minority for that range (if one exists) in $\mathcal{O}(1/\tau)$ time.*

Alternatively, for any function $f(n) = \omega(1)$, we can store our data structures for access, select and partial rank on S and range-minimum queries on C in a total of $nH + \mathcal{O}(n) + o(nH)$ at the cost of select queries taking $\mathcal{O}(f(n))$ time.

Theorem 2. *For any function $f(n) = \omega(1)$, we can store S in $nH + \mathcal{O}(n) + o(nH)$ bits such that later, given the endpoints of a range and τ, we can return a τ-minority for that range (if one exists) in $\mathcal{O}((1/\tau)\, f(n))$ time.*

In the full version of this paper we will reduce the space bound of Theorem 2 to $nH + o(n(H + 1))$ bits. That is, we improve Chan et al.'s solution to use optimally compressed space with nearly no slowdown.

4 Parameterized Range Majority with Fixed τ

The standard approach to finding τ-majorities, going back to Misra and Gries' work, is to build a list of $\mathcal{O}(1/\tau)$ candidate elements and then verify them. For parameterized range majority, an obvious way to verify candidates is to use rank queries. The problem with this approach is that, as noted in Subsection 2.1, we cannot support general rank queries in $o(\lg(\lg \sigma / \lg \lg n))$ time while using $n \lg^{\mathcal{O}(1)} n$ space; e.g., with only linear space, we cannot support general rank queries in $\mathcal{O}(1)$ time when the alphabet is super-polylogarithmic. If we can find the position of candidates' first occurrences in the range, however, then by Lemma 1 we can check them using only partial rank and select queries.

Suppose we want to support parameterized range majority on $S[1..n]$ for a fixed threshold τ. We first store data structures that support access, select and partial rank on S in $\mathcal{O}(1)$ time, which takes $\mathcal{O}(n)$ space. For $0 \leq b \leq \lfloor \lg n \rfloor$, let $F_b[1..n]$ be the binary string in which $F_b[k] = 1$ if the distinct element $S[k]$ occurs at least $\tau 2^b$ times in $S[k..k + 2^{b+1} - 1]$; and let S_b and C_b be the subsequences of S and C, respectively, consisting of those elements flagged by 1s in F_b. We store

F_b in $\mathcal{O}(n)$ bits such that we can support access, rank and select queries on F_b in $\mathcal{O}(1)$ time. Notice we can implement an access query on S_b or C_b as a select query on F_b and access queries on S or C, respectively. As described in Subsection 2.2, we can implement an access query to C as access, select and partial rank queries on S. We also store an $\mathcal{O}(1)$-time range-minimum data structure for C_b, which takes $\mathcal{O}(|S_b|)$ bits.

With these data structures, given endpoints i and j with $\lfloor \lg(j-i+1) \rfloor = b$, we use Muthukrishnan's algorithm to list the distinct elements in $S_b[F_b.\mathrm{rank}_1(i)..F_b.\mathrm{rank}_1(j)]$ and the positions of their leftmost occurrences therein; we then use select queries on F_b to find the positions of those elements in S. That is, we list the distinct elements in $S[i..j]$ that are flagged by 1s in F_b and the positions of their leftmost flagged occurrences therein. We then apply Lemma 1 to each of these elements, treating the positions of their leftmost flagged occurrences as the positions of their leftmost occurrences. Since each distinct element in $S[i..j]$ that is flagged in F_b occurs at least $\tau 2^b$ times in $S[i..j+2^{b+1}-1] \subset S[i..i+2^{b+2}]$, there are $\mathcal{O}(1/\tau)$ of them and we use a total of $\mathcal{O}(1/\tau)$ time.

Notice that the leftmost flagged occurrences of a distinct element a in $S[i..j]$ may not necessarily be the leftmost occurrence therein. However, if a is a τ-majority in $S[i..j]$ then, by definition, a occurs at least $\tau(j-i+1) \geq \tau 2^b$ times in $S[i..j] \subset S[i..i+2^{b+1}-1]$, so a's leftmost occurrence in $S[i..j]$ is flagged by a 1 in F_b and, therefore, we apply Lemma 1 to it. It follows that we return each τ-majority in $S[i..j]$.

We store only one set of data structures supporting access, select and partial rank on S. Summing over b from 0 to $\lfloor \lg n \rfloor$, the data structures for access, select, partial rank and range-minimum queries take a total of $\mathcal{O}(n \lg n)$ bits, which is $\mathcal{O}(n)$ words. Therefore, we have the first linear-space data structure with worst-case optimal $\mathcal{O}(1/\tau)$ query time for Karpinski and Nekrich's original problem of parameterized range majority with fixed τ.

Theorem 3. *Given a threshold τ, we can store a string in linear space and support parameterized range majority in $\mathcal{O}(1/\tau)$ time.*

5 Parameterized Range Majority with Variable τ

5.1 Nearly linear space with optimal query time

Suppose we have an instance of the data structure from Theorem 3 for each threshold $1, 1/2, 1/4, \ldots, 1/2^{\lceil \lg n \rceil}$, which takes a total of $\mathcal{O}(n \lg n)$ space. Given endpoints i and j and a threshold τ, we can use the instance for threshold $1/2^{\lceil \lg(1/\tau) \rceil}$ to build a list of $\mathcal{O}(1/\tau)$ candidate elements and then check them with Lemma 1; this takes a total of $\mathcal{O}(1/\tau)$ time and returns all the τ-majorities in $S[i..j]$. Gagie et al. used a variant of this idea to obtain the first data structure for variable τ. We can easily reduce our space bound to $\mathcal{O}(n \lg \sigma)$ because, if $1/\tau \geq \sigma$, then we can simply use Muthukrishnan's algorithm with S and C to list in $\mathcal{O}(\sigma) = \mathcal{O}(1/\tau)$ time all the distinct elements in $S[i..j]$ and the positions of their leftmost occurrences therein, then check them with Lemma 1.

Notice that we need store only one set of data structures supporting access, select and partial rank on S. Also, if $S[k]$ is a $(1/2^t)$-majority in a range, then it is also a $(1/2^{t'})$-majority for all $t' \geq t$. It follows that if, instead of querying only the instance for the threshold $1/2^{\lceil \lg(1/2) \rceil}$, we query the instances for all the thresholds $1, 1/2, 1/4, \ldots, 1/2^{\lceil \lg(1/\tau) \rceil}$ — which still takes $\mathcal{O}\left(\sum_{t=0}^{2^{\lceil \lg(1/\tau) \rceil}} 2^t\right) = \mathcal{O}(1/\tau)$ time — then we can modify the instances to reduce the total number of 1s in their binary strings. Specifically, for $0 \leq t \leq \lceil \lg \sigma \rceil$, let F_b^t be the binary string F_b in the instance for threshold $1/2^t$; we modify F_b^t such that $F_b^t[k] = 1$ if and only if the number of occurrences of the distinct element $S[k]$ in $S[k..k + 2^{b+1} - 1]$ is at least 2^{b-t} times but less than 2^{b-t+1}.

For $0 \leq b \leq \lfloor \lg n \rfloor$ and $1 \leq k \leq n$, we have $F_b^t[k] = 1$ for at most one value of t. Therefore, all the binary strings contain a total of at most $n(\lfloor \lg n \rfloor + 1)$ copies of 1, so all the range-minimum data structures take a total of $\mathcal{O}(n \lg n)$ bits. Since the binary strings have total length $n\lceil \lg n \rceil \lceil \lg \sigma \rceil$, we can use Pătraşcu's data structure to store them in a total of $\mathcal{O}(n \lg(n) \lg \lg \sigma)$ bits. A slightly neater approach is to represent all the binary strings $F_b^0, \ldots, F_b^{\lceil \lg \sigma \rceil}$ as a single string $T_b[1..n]$ in which $T_b[k] = t$ if $F_b^t[k] = 1$, and ∞ if there is no such value t. We can implement access, rank and select queries on $F_b^0, \ldots, F_b^{\lceil \lg \sigma \rceil}$ by access, rank and select queries on T_b. Since T_b is an alphabet of size $\mathcal{O}(\lg \sigma)$, we can use Ferragina et al.'s data structure to store it in $\mathcal{O}(n \lg \lg \sigma)$ bits and support access, rank and select queries in $\mathcal{O}(1)$ time. Either way, in total we use $\mathcal{O}(n \lg \lg \sigma)$ space.

Theorem 4. *We can store S in $\mathcal{O}(n \lg \lg \sigma)$ space such that later, given the endpoints of a range and τ, we can return the τ-majorities for that range in $\mathcal{O}(1/\tau)$ time.*

5.2 Optimally compressed space with nearly optimal query time

To be able to apply Lemma 1, we must be able to find the leftmost occurrence of each τ-majority in a range. For this reason, we may flag many occurrences of the same distinct element even when they appear in close succession, because we cannot know in advance where the query range will start. As discussed in Section 4, however, if we have a data structure that supports rank queries on S, then it is sufficient for us to build a list of $\mathcal{O}(1/\tau)$ candidate elements that includes all the τ-majorities — without any information about positions — and then verify them using rank queries. This lets us flag fewer elements and so reduce our space bound, at the cost of using slightly suboptimal query time.

We store an instance of Belazzougui and Navarro's data structure supporting access on S in $\mathcal{O}(1)$ and rank and select on S in $\mathcal{O}(\lg \lg \sigma)$ time, which takes $nH + o(n(H + 1))$ bits. For $0 \leq t \leq \lceil \lg \sigma \rceil$ and $\lfloor \lg(2^t \lg \lg \sigma) \rfloor \leq b \leq \lfloor \lg n \rfloor$, we divide S into blocks of length 2^{b-1} and store data structures supporting access, rank and select on the binary string $G_b^t[1..n]$ in which $G_b^t[k] = 1$ if, first, the distinct element $S[k]$ occurs at least 2^{b-t} times in $S[k - 2^{b+1}..k + 2^{b+1}]$ and, second, $S[k]$ is the leftmost or rightmost occurrence of that distinct element in its block. We also store an $\mathcal{O}(1)$-time range-minimum data structure for the subsequence of C consisting of elements flagged by 1s in G_b^t.

The number of distinct elements that occur at least 2^{b-t} times in a range of size $\mathcal{O}(2^b)$ is $\mathcal{O}(2^t)$, so there are $\mathcal{O}(2^t)$ elements in each block flagged by 1s in G_b^t. It follows that we can store an instance of Pătraşcu's data structure supporting $\mathcal{O}(1)$-time access, rank and select on G_b^t in $\mathcal{O}(n2^{t-b}(b-t) + n/\lg^3 n)$ bits; we need $\mathcal{O}(2^t)$ bits for the corresponding range-minimum data structure. Summing over t from 0 to $\lceil \lg \sigma \rceil$ and over b from $\lfloor \lg(2^t \lg\lg \sigma) \rfloor$ to $\lfloor \lg n \rfloor$, calculation shows we use a total of $\mathcal{O}\left(\frac{n \lg \sigma \lg\lg\lg \sigma}{\lg\lg \sigma} + \frac{n}{\lg n}\right) = o(n \lg \sigma)$ bits for the binary strings and range-minimum data structures. Therefore, including the instance of Belazzougui and Navarro's data structure for S, we use $nH + o(n \lg \sigma)$ bits altogether.

Given endpoints i and j and a threshold τ, if

$$\lfloor \lg(j - i + 1) \rfloor < \left\lfloor \lg\left(2^{\lceil \lg(1/\tau) \rceil} \lg\lg \sigma\right) \right\rfloor ,$$

then we simply run Misra and Gries' algorithm on $S[i..j]$ in $\mathcal{O}(j - i) = \mathcal{O}((1/\tau) \lg\lg \sigma)$ time. Otherwise, we use Muthukrishnan's algorithm to list the distinct elements flagged by 1s in G_b^t, where $t = \lceil \lg(1/\tau) \rceil$ and $b = \lfloor \lg(j - i + 1) \rfloor \geq \lfloor \lg(2^t \lg\lg \sigma) \rfloor$, and use rank queries on S to check whether each of them is a τ-majority in $S[i..j]$. Since $S[i..j]$ overlaps at most 5 blocks of length 2^{b-1}, it contains $\mathcal{O}(1/\tau)$ distinct elements flagged by 1s in G_b^t; therefore, Muthukrishnan's algorithm takes $\mathcal{O}(1/\tau)$ time and we use a total of $\mathcal{O}((1/\tau) \lg\lg \sigma)$ time for all the rank queries on S.

Since $S[i..j]$ cannot be completely contained in a block of length 2^{b-1}, if $S[i..j]$ overlaps a block then it includes one of that block's endpoints. Therefore, if $S[i..j]$ contains an occurrence of a distinct element a, then it includes the leftmost or rightmost occurrence of a in some block. Suppose a is a τ-majority in $S[i..j]$. For $i \leq k \leq j$, a occurs at least 2^{b-t} times in $S[k - 2^{b+1}..k + 2^{b+1}]$, so some occurrence of a in $S[i..j]$ is flagged by a 1 in G_b^t. Therefore, we return a.

Theorem 5. *We can store S in $nH + o(n \lg \sigma)$ bits such that later, given the endpoints of a range and τ, we can return the τ-majorities for that range in $\mathcal{O}((1/\tau) \lg\lg \sigma)$ time.*

Since our solution includes an instance of Belazzougui and Navarro's data structure, we can also support $\mathcal{O}(1)$-time access to S and $\mathcal{O}(\lg\lg \sigma)$-time rank and select. In the full version of this paper we will reduce the space bound of Theorem 5 to $nH + o(n(H + 1))$ bits.

5.3 Nearly optimally compressed space with very nearly optimal query time

Recall from Subsection 5.1 that, if $1/\tau \geq \sigma$, then we can simply use Muthukrishnan's algorithm to list all the distinct elements in a range and then check them with Lemma 1; therefore, we can assume $1/\tau < \sigma$. In this subsection we use a new data structure with density-sensitive query time for one-dimensional range counting, which may be of independent interest, to obtain a nearly optimally compressed data structure for parameterized range majority with $\mathcal{O}\left((1/\tau) \lg \frac{\lg(1/\tau)}{\lg\lg n}\right)$

query time. Due to space constraints, however, we leave the description of our range-counting data structure to the full version of this paper and merely state our result here:

Theorem 6. *For any positive constant ϵ, we can store S in $(1+\epsilon)nH + \mathcal{O}(n)$ bits such that later, given endpoints i and j and a distinct element a, we can return $\mathrm{occ}(a, S[i..j])$ in $\mathcal{O}\left(\lg \frac{\lg \frac{j-i+1}{\mathrm{occ}(a,S[i..j])}}{\lg \lg n}\right)$ time. We can also support access and select in $\mathcal{O}(1)$ time and rank in $\mathcal{O}(\lg \lg \sigma)$ time.*

To obtain a compressed data structure for parameterized range majority with $\mathcal{O}\left((1/\tau) \lg \frac{\lg(1/\tau)}{\lg \lg n}\right)$ query time, we combine our solution from Theorem 5 with Theorem 6. Instead of using $\mathcal{O}(\lg \lg \sigma)$-time rank queries to check each of the $\mathcal{O}(1/\tau)$ candidate elements returned by Muthukrishnan's algorithm, we use range-counting queries. We can make all $\mathcal{O}(1/\tau)$ range-counting queries each take $\mathcal{O}\left(\lg \frac{\lg(1/\tau)}{\lg \lg n}\right)$ time because, if one starts taking too much time, then the distinct element we are checking cannot be a τ-majority and we can stop the query early. (In fact, as we will show in the full version of this paper, our data structure from Theorem 6 does not need such intervention.) This gives us our final result:

Theorem 7. *We can store S in $(1+\epsilon)nH + o(n \lg \sigma)$ bits such that later, given the endpoints of a range and τ, we can return the τ-majorities for that range in $\mathcal{O}\left((1/\tau) \lg \frac{\lg(1/\tau)}{\lg \lg n}\right)$ time.*

Notice our solution in Theorem 7 takes optimal $\mathcal{O}(1/\tau)$ time when $1/\tau = \lg^{\mathcal{O}(1)} n$. Again, we can also support access and select in $\mathcal{O}(1)$ time and rank in $\mathcal{O}(\lg \lg \sigma)$ time. In the full version of this paper we will reduce the space bound in Theorem 7 to $(1+\epsilon)nH+\mathcal{O}(n)$ bits, and show how to use our data structures from Theorems 5 and 7 to find a range mode quickly when it is actually reasonably frequent.

6 Conclusions

We have given the first linear-space data structure for parameterized range majority with query time $\mathcal{O}(1/\tau)$, which is worst-case optimal in terms of n and τ. Moreover, we have improved the space bounds for parameterized range majority and minority in the important case of variable τ. For parameterized range majority with variable τ, we have achieved nearly linear space and worst-case optimal query time, or compressed space with a slight slowdown. For parameterized range minority, we have improved Chan et al.'s solution to use nearly compressed space with no slowdown or compressed space with nearly no slowdown. In the full version of this paper we will also reduce the lower-order terms in our compressed space bounds to $o(n(H+1))$ with the same slowdowns. We leave as an open problem achieving linear or compressed space with $\mathcal{O}(1/\tau)$ query time for variable τ, or showing that this is impossible.

Acknowledgments

Many thanks to Patrick Nicholson for helpful comments.

References

1. J. Barbay, F. Claude, T. Gagie, G. Navarro, and Y. Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*. To appear.
2. D. Belazzougui and G. Navarro. Alphabet-independent compressed text indexing. *ACM Transactions on Algorithms*. To appear.
3. D. Belazzougui and G. Navarro. Alphabet-independent compressed text indexing. In *Proceedings of the 19th European Symposium on Algorithms (ESA)*, pages 748–759, 2011.
4. D. Belazzougui and G. Navarro. New lower and upper bounds for representing sequences. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 181–192, 2012.
5. P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 377–388, 2005.
6. T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. In *Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 290–301, 2012.
7. T. M. Chan, S. Durocher, M. Skala, and B. T. Wilkinson. Linear-space data structures for range minority query in arrays. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 295–306, 2012.
8. G. Cormode and S. Muthukrishnan. Data stream methods. `http://www.cs.rutgers.edu/~muthu/198-3.pdf`, 2003. Lecture 3 of Rutger's *198:671 Seminar on Processing Massive Data Sets*.
9. E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, pages 348–360, 2002.
10. S. Durocher, M. He, J. I. Munro, P. K. Nicholson, and Matthew Skala. Range majority in constant time and linear space. *Information and Computation*, 222:169–179, 2013.
11. A. Elmasry, J. I. Munro, and P. K. Nicholson. Dynamic range majority data structures. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 150–159, 2011.
12. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms*, 3(2), 2007.
13. J. Fischer. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 158–169, 2010.
14. T. Gagie, M. He, J. I. Munro, and P. K. Nicholson. Finding frequent elements in compressed 2D arrays and strings. In *Proceedings of the 18th Symposium on String Processing and Information Retrieval (SPIRE)*, pages 295–300, 2011.
15. M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 605–616, 2010.

16. R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.
17. M. Karpinski and Y. Nekrich. Searching for frequent colors in rectangles. In *Proceedings of the 20th Canadian Conference on Computational Geometry (CCCG)*, pages 11–14, 2008.
18. D. Krizanc, P. Morin, and M. H. M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12(1):1–17, 2005.
19. Y. K. Lai, C. K. Poon, and B. Shi. Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms*, 6(3):420–432, 2008.
20. J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.
21. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the 13th Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.
22. H. Petersen. Improved bounds for range mode and range median queries. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 418–423, 2008.
23. H. Petersen and S. Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Information Processing Letter*, 109(4):225–228, 2009.
24. M. Pătraşcu. Succincter. In *Proceedings of the 49th Symposium on Foundations of Computer Science (FOCS)*, pages 305–313, 2008.
25. K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.
26. Z. Wei and K. Yi. Beyond simple aggregates: indexing for summary queries. In *Proceedings of the 30th Symposium on Principles of Database Systems (PODS)*, pages 117–128, 2011.