Theoretical Computer Science 885 (2021) 1–14

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Approximate query processing over static sets and sliding windows ☆

Ran Ben Basat [a], Seungbum Jo [b], Srinivasa Rao Satti [c,*], Shubham Ugare [d]

[a] University College London, UK
[b] Chungbuk National University, South Korea
[c] Norwegian University of Science and Technology, Norway
[d] IIT Guwahati, India

ARTICLE INFO

Article history:
Received 15 March 2021
Received in revised form 8 June 2021
Accepted 13 June 2021
Available online 17 June 2021
Communicated by M. Zehavi

Keywords:
Streaming
Algorithms
Sliding window
Lower bounds

ABSTRACT

Indexing of static and dynamic sets is fundamental to a large set of applications such as information retrieval and caching. Denoting the characteristic vector of the set by B, we consider the problem of encoding sets and multisets to support *approximate* versions of the operations $\mathsf{rank}(i)$ (i.e., computing $\sum_{j \le i} B[j]$) and $\mathsf{select}(i)$ (i.e., finding $\min\{p \mid \mathsf{rank}(p) \ge i\}$) queries. We study multiple types of approximations (allowing an error in the query or the result) and present lower bounds and succinct data structures for several variants of the problem. We also extend our model to sliding windows, in which we process a stream of elements and compute *suffix sums*. This is a generalization of the window summation problem that allows the user to specify the window size *at query time*. Here, we provide an algorithm that supports updates and queries in constant time while requiring just $(1 + o(1))$ factor more space than the fixed-window summation algorithms.

1. Introduction

Given a bit-string $B[1 \dots n]$ of size n, one of the fundamental and well-known problems proposed by Jacobson [16], is to construct a space-efficient data structure which can answer rank and select queries on B efficiently. For $b \in \{0, 1\}$, these queries are defined as follows.

- $\mathsf{rank}_b(i, B)$: returns the number of b's in $B[1 \dots i]$.
- $\mathsf{select}_b(i, B)$: returns the position of the i-th b in B.

A bit vector supporting a subset of these operations is one of the basic building blocks in the design of various succinct data structures. Supporting these operations in constant time, with close to the optimal amount of space, both theoretically and practically, has received a wide range of attention [17,19–21,24]. Some of these results also explore trade-offs that allow more query time while reducing the space.

☆ Preliminary version of these results have appeared in the proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 18) [7].

* Corresponding author.
E-mail addresses: r.benbasat@cs.ucl.ac.uk (R.B. Basat), sbjo@chungbuk.ac.kr (S. Jo), srinivasa.r.satti@ntnu.no (S.R. Satti), ugare.dipak@iitg.ac.in (S. Ugare).

https://doi.org/10.1016/j.tcs.2021.06.015

We also consider related problems in the streaming model, where a quasi-infinite sequence of integers arrives, and our algorithms need to support the operation of appending a new item to the end of the stream. For $i \in \{1, \ldots, n\}$, let S_i be the sum of the last i integers. Here, n is the maximal suffix size we support queries for. For streaming, we consider processing a stream of elements, and answering two types of queries, *suffix sum* (ss) and *inverse suffix sum* (iss), defined as:

- $\mathsf{ss}(i, n)$: returns S_i for any $1 \leq i \leq n$.
- $\mathsf{iss}(i, n)$: returns the smallest j, $1 \leq j \leq n$, such that $\mathsf{ss}(j, n) \geq i$.

In this paper, our goal is to obtain space efficient data structures for supporting a few relaxations of these queries efficiently using an amount of space below the theoretical minimum (for the unrelaxed versions), ideally. To this end, we define *approximate* versions of rank and select queries, and propose data structures for answering *approximate rank and select queries* on multisets and bit-strings. We consider the following approximate queries with an *additive* error $\delta > 0$.

- $\mathsf{rankA}_b(i, B, \delta)$: returns any value r which satisfies $\mathsf{rank}_b(i - \delta, B) < r \leq \mathsf{rank}_b(i, B)$. If $\mathsf{rank}_b(i - \delta, B) = \mathsf{rank}_b(i, B)$, then $\mathsf{rankA}_b(i, B, \delta) = \mathsf{rank}_b(i, B)$.
- $\mathsf{drankA}_b(i, B, \delta)$: returns any value r which satisfies $\mathsf{rank}_b(i, B) - \delta < r \leq \mathsf{rank}_b(i, B)$.
- $\mathsf{selectA}_b(i, B, \delta)$: returns any position p which satisfies $\mathsf{select}_b(i - \delta, B) < p \leq \mathsf{select}_b(i, B)$.
- $\mathsf{dselectA}_b(i, B, \delta)$: returns any position p which satisfies $\mathsf{select}_b(i, B) - \delta < p \leq \mathsf{select}_b(i, B)$.
- $\mathsf{ssA}(i, n, \delta)$: returns any value r which satisfies $\mathsf{ss}(i, n) - \delta < r \leq \mathsf{ss}(i, n)$.
- $\mathsf{issA}(i, n, \delta)$: returns any value r which satisfies $\mathsf{iss}(i - \delta, n) < r \leq \mathsf{iss}(i, n)$.

We propose data structures for supporting approximate rank and select queries on bit-strings efficiently. Our data structures uses less space than that is required to answer the exact queries, and most of the data structures use optimal space. We also propose a data structure for supporting ssA and issA queries on binary streams while supporting updates efficiently. Finally, we extend some of these results to the case of larger alphabets. For all these results, we assume the standard word-RAM model [18] with word size $\Theta(\lg n)$ if it is not explicitly mentioned.

1.1. Previous work

Rank and Select over bit-strings. Given a bit-string B of size n, it is clear that at least n bits are necessary to support rank and select queries on B (since the bit vector can be reconstructed by using the answers to these queries). Jacobson [16] proposed a data structure for answering rank queries on B in constant time using $n + o(n)$ bits. Clark and Munro [8] extended it to support both rank and select queries in constant time with $n + o(n)$ bits. For the case when there are m 1's in B, at least $\mathcal{B}(n, m) = \lg \left\lceil \binom{n}{m} \right\rceil$ bits[1] are necessary to support rank and select on B. Raman et al. [24] proposed a data structure that supports both operations in constant time while using $\mathcal{B}(n, m) + o(n) + O(\lg \lg m)$ bits. Golynski et al. gave an asymptotically optimal time-space trade-off for supporting rank and select queries on B [14]. A slightly related problem of *approximate color counting* has been considered in El-Zein et al. [10].

Algorithms that Sum over Sliding Windows. A natural generalization of the static case is answering queries with respect to a *sliding window* over a data stream. The sliding window model was extensively studied for multiple problems including summing [4,9], heavy hitters [2,5], Bloom filters [1] and counting distinct elements [11]. Our ss queries for streaming are a generalization of the problem of summing over sliding windows. That is, window summation is a special case of the suffix sum problem where the algorithm is always asked for the sum of the last $i \leq n$ elements. Approximating the sum of the last n elements over a stream of integers in $\{0, 1, \ldots, \ell\}$, was first introduced by Datar et al. [9]. They proposed a $(1 + \varepsilon)$ multiplicative approximation algorithm that uses $O\left(\varepsilon^{-1}\left(\lg^2 n + \lg \ell \cdot (\lg n + \lg \lg \ell)\right)\right)$ bits and operates in $O(\lg \ell / \lg n)$ amortized time, or $O(\lg(\ell \cdot n))$ worst case time. In [12], Gibbons and Tirthapura presented a $(1 + \varepsilon)$ multiplicative approximation algorithm that operates in constant worst case time while using similar space for $\ell = n^{O(1)}$. Ben-Basat et al. [4] studied the potential memory savings one can get by replacing the $(1 + \varepsilon)$ multiplicative guarantee with a δ additive approximation. They showed that $\Theta(\ell \cdot n/\delta + \lg n)$ bits are required and sufficient. Recently, [3] showed the potential memory saving of a bi-criteria approximation, which allows error in both the sum and the time axis, for sliding window summation. The data structure of [6] looks at a generalization of the ssA queries to general alphabet, where at query time we also receive an element x and return an estimate for the frequency of x in the last i elements.

It is worth mentioning that these data structures *do* allow computing the sum of a window whose size is given at the query time. Alas, the query time will be slower as they do not keep aggregates that allow quick computation. Specifically, we can compute a $(1+\epsilon)$ multiplicative approximation in $O(\varepsilon^{-1} \lg(\ell n \varepsilon))$ time using the data structures of [9] and [12]. We can also use the data structure of [4] for an additive approximation of δ in $O(n\ell/\delta)$ time.

[1] $\mathcal{B}(n, m)$ bits are the information-theoretic lower bound on space for storing a subset of size $m \leq n$ from the universe $\{1, 2, \ldots, n\}$. Note that $\mathcal{B}(n, m) = m \lg(en/m) - O(\lg m) - \Theta(m^2/n)$ [24].

Table 1
Summary of results of upper and lower bounds for approximate rank and select queries on bit-string of size n (m is the number of 1's in B). The function $t(n,u)$ is defined as $t(n,u) = O(\min\{\lg\lg n \lg\lg u / \lg\lg\lg u, \sqrt{\lg n / \lg\lg n}\})$.

Query	Space (in bits)	Query time	Error
Lower bounds			
drankA_1, $\mathsf{selectA}_1$	$\lfloor n/\delta \rfloor$		δ, additive
drankA_1, $\mathsf{selectA}_1$	$\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor)$		
rankA_1, $\mathsf{dselectA}_1$	$\lfloor n/2\delta \rfloor \lg\delta$		
$\mathsf{dselectA}_1$	$O((n/\delta)\lg^{O(1)}\delta)$	$\Omega(\lg\lg n)$	
Upper bounds			
drankA_1, $\mathsf{selectA}_1$	$n/\delta + o(n/\delta)$		δ, additive
drankA_1, $\mathsf{selectA}_1$	$\mathcal{B}(n/\delta, m/\delta) + o(n/\delta)$	$O(1)$	
rankA_1	$(n/\delta)\lg\delta + o((n/\delta)\lg\delta)$		
$\mathsf{dselectA}_1$	$(n/\delta)\lg\delta + o((n/\delta)\lg\delta)$	$t(n/\delta, n)$	

Table 2
Comparison of data structures for ss queries over stream of integers in $\{0,\ldots,\ell\}$. All works can answer fixed-size window queries (where $i \equiv n$) in $O(1)$ time. Worst case times are specified.

	Guarantee	Space (in bits)	Update time	Query time
DGIM02 [9]	$(1+\varepsilon)$-multiplicative	$O(\varepsilon^{-1}\lg(\ell n)\lg(n\lg\ell))$	$O(\lg(\ell n))$	$O(\varepsilon^{-1}\lg(\ell n\varepsilon))$
GT02 [12]	$(1+\varepsilon)$-multiplicative	$O(\varepsilon^{-1}\lg^2(\ell n))$	$O(1)$	$O(\varepsilon^{-1}\lg(\ell n\varepsilon))$
BEFK16 [4]	δ-additive, for $\delta = \Omega(\ell)$	$\Theta(\ell \cdot n/\delta + \lg n)$	$O(1)$	$O(\ell \cdot n/\delta)$
BEFK16 [4]	δ-additive, for $\delta = o(\ell)$	$\Theta(n\lg(\ell/\delta))$	$O(1)$	$O(n)$
This paper	δ-additive	Same as in [4]	$O(1)$	$O(1)$

1.2. Our results

In this paper, we obtain the following results for the approximate rank, select, ss and iss queries with additive error. Let B be a bit-string of size n.

1. rank and select queries with additive error δ:

- We first show that $\lfloor n/\delta \rfloor$ bits are necessary for answering drankA_1 and $\mathsf{selectA}_1$ queries on B and propose a $(\lceil n/\delta \rceil + o(n/\delta))$-bit data structure that supports drankA_1 and $\mathsf{selectA}_1$ queries on B in constant time. For the case when there are m 1's in B, we show that $\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor)$ bits are necessary for answering drankA_1 and $\mathsf{selectA}_1$ queries on B, and obtain $\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor) + o(n/\delta)$-bit data structure that supports drankA_1 and $\mathsf{selectA}_1$ queries on B in constant time.
- We show that $\lfloor n/2\delta \rfloor \lg\delta$ bits are necessary for answering rankA_1 and $\mathsf{dselectA}_1$ queries on B, and obtain an $((n/\delta)\lg\delta + o((n/\delta)\lg\delta))$-bit data structure that supports rankA_1 queries in $O(1)$ time, and $\mathsf{dselectA}_1$ queries in $O(\min\{\lg\lg(n/\delta)\lg\lg n/\lg\lg\lg n, \sqrt{\lg(n/\delta)/\lg\lg(n/\delta)}\})$ time. Furthermore, we show that there exists an additive error δ such that any $O((n/\delta)\lg^{O(1)}\delta)$-bit data structure requires at least $\Omega(\lg\lg n)$ time to answer $\mathsf{dselectA}_1$ queries on B.
- Using the above data structures, we also obtain data structures for answering approximate rank and select queries on a given multiset S from the universe $U = \{1, 2 \ldots n\}$ with additive error δ, where $\mathsf{rank}(i, S)$ query returns the value $|\{j \in S | j \leq i\}|$, and $\mathsf{select}(i, S)$ query returns the i-th smallest element in S. We consider two different cases: (i) rankA, drankA selectA, and dselectA queries when $|S| = m$, and (ii) drankA and selectA queries when the frequency each element in S is at most ℓ. For case (ii), we first show that at least $\lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1) + 1)$ bits are necessary for answering drankA queries, and obtain an optimal space structure that supports drankA queries in constant time, and an asymptotically optimal space structure that supports both drankA and selectA queries in constant time when $\ell = O(\delta)$.
- We also consider the drankA and selectA queries on strings over large alphabets. Given a string A of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ of size σ, we obtain a $((2n/\delta)\lg(\sigma+1) + o((n/\delta)\lg(\sigma+1))$-bit data structure that supports drankA and selectA on A in $O(\lg\lg\sigma)$ time. We summarize our results for bit-strings in Table 1.

2. ss and iss queries with additive error δ:

- We first consider a data structure for answering ss and iss queries on binary stream, i.e., all integers in the stream are 0 or 1. For exact ss and iss queries on the stream, we propose an $n + o(n)$-bit data structure for answering those queries in constant time while supporting constant time updates whenever a new element arrives from the stream. This data structure is obtained by modifying the data structure of Clark and Munro [8] for answering rank and select queries on bit-strings. Using the above structure, we obtain an $(n/\delta + o(n/\delta) + O(\lg n))$-bit structure that supports ssA and issA queries on the stream in constant time while supporting constant time updates. Since at least $\lfloor n/\delta \rfloor$ bits are

necessary for answering drankA_1 (or $\mathsf{selectA}_1$) queries on bit-strings, and $\lfloor \lg n \rfloor$ bits are necessary for answering $\mathsf{ss}(n, n)$ queries [4], the space usage of our data structure is succinct (i.e., optimal up to lower-order terms) when $n/\delta = \omega(\lg n)$, and asymptotically optimal otherwise.

- We then consider the generalization that allows integers in the range $\{0, 1, \ldots, \ell\}$, for some $\ell \in \mathbb{N}$. First, we present an algorithm that uses the optimal $n \lg(\ell + 1)(1 + o(1))$ bits for exact suffix sums. Then, we provide a second algorithm that uses $\lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1) + 1)(1 + o(1)) + O(\lg n)$ bits for solving ssA. Specifically, our data structure is succinct when $n\ell/\delta = \omega(\lg n)$, and is asymptotically optimal otherwise, and improves the query time of [4] while using the same space. Table 2 presents this comparison.

2. Queries on binary strings and streams

In this section, we first consider data structures for answering approximate rank and select queries on bit-strings and multisets. We also show how to extend our data structures on static bit-strings to sliding windows on binary streams, for answering approximate ss and iss queries.

2.1. Approximate rank and select queries on bit-strings

We consider the approximate rank and select queries on bit-strings with additive error δ. We only show how to support rankA_1, drankA_1, $\mathsf{dselectA}_1$, and $\mathsf{selectA}_1$ queries. To support rankA_0, drankA_0, $\mathsf{dselectA}_0$, and $\mathsf{selectA}_0$ queries, one can construct the same data structures on the bit-wise complement of the original bit-string. We first introduce a few previous results which will be used in our structures. The following lemmas describe the optimal structures for supporting rank and select queries on bit-strings.

Lemma 2.1 *([8]). For a bit-string B of length n, there is a data structure of size $n + o(n)$ bits that supports rank_0, rank_1, select_0, and select_1 queries in $O(1)$ time.*

Lemma 2.2 *([24]). For a bit-string B of length n with m 1's, there is a data structure of size*

- *(a) $\mathcal{B}(n, m) + o(m)$ bits that supports select_1 queries in $O(1)$ time, and*
- *(b) $\mathcal{B}(n, m) + o(n)$ bits that supports rank_0, rank_1, select_0, and select_1 queries in $O(1)$ time.*

We use results from [15] and [23], which describe efficient data structures for supporting the following queries on integer arrays. For the standard word-RAM model with word size $O(\lg U)$ bits, let A be an array of n non-negative integers. For $1 \le i \le n$ and any non-negative integer x, (i) $\mathsf{sum}(i)$ returns the value $\sum_{j=1}^{i} A[j]$, and (ii) $\mathsf{search}(x)$ returns the smallest i such that $\mathsf{sum}(i) > x$. In what follows, we use the following function to state the running time of some of the (*Searchable Partial Sum*) queries.

$$\mathsf{SPS}(n, U) = \begin{cases} O(1) & \text{if } n = \mathit{polylog}(U) \\ O\left(\min\left\{\lg\lg n \frac{\lg\lg U}{\lg\lg\lg U}, \sqrt{\frac{\lg n}{\lg\lg n}}\right\}\right) & \text{otherwise} \end{cases}$$

Lemma 2.3 *([15], [23]). An array of n non-negative integers, each of length at most α bits, can be stored using $\alpha n + o(\alpha n)$ bits, to support sum queries on A in constant time, and search queries on A in $\mathit{SPS}(n, n2^{\alpha})$ time. Moreover, when $\alpha = O(\lg\lg n)$, we can answer both queries in $O(1)$ time.*

Supporting drankA and selectA queries. We first consider the problem of supporting drankA_1 or $\mathsf{selectA}_1$ queries with additive error δ on a bit-string B of length n, and prove a lower bound on space used by any data structure that supports either of these two queries.

Theorem 2.4. *Any data structure that supports drankA_1 or $\mathit{selectA}_1$ queries with additive error δ on a bit-string of length n requires at least $\lfloor n/\delta \rfloor$ bits. Also if the bit-string has m 1's in it, then at least $\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor)$ bits are necessary for answering the above queries.*

Proof. Consider a bit-string B of length n divided into $\lfloor n/\delta \rfloor$ blocks $B_1, B_2, \ldots, B_{\lfloor n/\delta \rfloor}$ where for $1 \le i < \lfloor n/\delta \rfloor$, $B_i = B[\delta(i-1)+1, \ldots, \delta i]$ and $B_{\lfloor n/\delta \rfloor} = B[\delta(\lfloor n/\delta \rfloor - 1) + 1, \ldots, n]$ (the last block may contain more than δ, but less than 2δ bits). Let S be the set of all possible bit-strings satisfying the condition that all the bits within a block are the same (i.e., each block contains either all zeros, or all ones), and hence $|S| = 2^{\lfloor n/\delta \rfloor}$. We now show that any two distinct bit-strings in S will have different answers for some drankA_1 query (and also some $\mathsf{selectA}_1$ query). Consider two distinct bit-strings B and B' in S, and let i be the leftmost index with $B_i \neq B'_i$. Then by the construction of B and B', all the possible answers of $\mathsf{drankA}_1(i\delta, B, \delta)$ are different from all the possible answers of $\mathsf{drankA}_1(i\delta, B', \delta)$, for all values of i (note that the answer to a drankA_1 query is not unique). By the same argument, there exists no position which can be an answer of both

$\mathsf{selectA}_1(j, B, \delta)$ and $\mathsf{selectA}_1(j, B', \delta)$ queries, where j is the number of 1's in $B[1 \dots i\delta]$. Thus, any structure that supports either of these queries must distinguish between every element in S, which implies $\lfloor n/\delta \rfloor$ bits are necessary to answer drankA_1 or $\mathsf{selectA}_1$ queries.

For the case when the number of 1's in the bit-string is fixed to be m, we choose $\lfloor m/\delta \rfloor$ blocks from each bit-string and make all bits in the chosen blocks to be 1's (and the rest of the bits as 0's). Since there are $\binom{\lfloor n/\delta \rfloor}{\lfloor m/\delta \rfloor}$ ways for select such $\lfloor m/\delta \rfloor$ blocks in a bit-string of length n, it implies that $\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor)$ bits are necessary to answer drankA_1 or $\mathsf{selectA}_1$ queries in this case. □

The following theorem gives a data structure for supporting drankA_1 and $\mathsf{selectA}_1$ queries in constant time, using optimal space.

Theorem 2.5. *For a bit-string B of length n, if there is an $s(n)$-bit data structure which supports rank_1 and select_1 queries in $t(n)$ time, then there exists an $s(n/\delta)$-bit data structure which supports drankA_1 and $\mathsf{selectA}_1$ queries with additive error δ in $t(n)$ time.*

Proof. We divide B into $\lceil n/\delta \rceil$ blocks $B_1, B_2, \dots, B_{\lceil n/\delta \rceil}$ where for $1 \leq i < \lceil n/\delta \rceil$, $B_i = B[\delta(i-1)+1 \dots \delta i]$ and $B_{\lceil n/\delta \rceil} = B[\delta(\lceil n/\delta \rceil - 1) + 1 \dots n]$. Now we define another bit-string B' of length $\lceil n/\delta \rceil$ where for $1 \leq i \leq \lceil n/\delta \rceil$, $B'[i] = 1$ if B_i contains $j\delta$-th 1 in B for any integer $j \leq i$, and $B'[i] = 0$ otherwise. Note that any block of B has at most one such position in B. Then by constructing the $s(n)$-bit data structure on B', we can support rank_1 and select_1 queries on B' in $t(n)$ time.

Now we claim that $C = \delta \cdot \mathsf{rank}_1(\lfloor i/\delta \rfloor, B') + (i \mod \delta) B'[\lceil i/\delta \rceil]$ gives an answer of the $\mathsf{drankA}_1(i, B, \delta)$ query, which can be computed in $t(n)$ time. Let $D = \delta \cdot \mathsf{rank}_1(\lfloor i/\delta \rfloor, B')$, and let d be the position of D-th 1 in B. By the definition of B', if $B'[\lceil i/\delta \rceil] = 0$ or $(i \mod \delta) = 0$, the claim holds since there are less than δ 1's in $B[d \dots i]$. Now consider the case when $B'[\lceil i/\delta \rceil] = 1$ and $(i \mod \delta) \neq 0$. Then there are at most $(\delta + (i \mod \delta) - 1)$ 1's in $B[d \dots i]$, which is the case when $(\delta \lfloor i/\delta \rfloor + 1)$ is the position of the $(D+\delta)$-th 1 in B, and all the values in $B[(\delta \lfloor i/\delta \rfloor + 2) \dots i]$ are 1. Also there are at least $\delta - (\delta - (i \mod \delta)) = (i \mod \delta)$ 1's in $B[d \dots i]$, which is the case when $(\delta \lceil i/\delta \rceil)$ is the position of the $(D+\delta)$-th 1 in B and all the values in $B[\delta \lfloor i/\delta \rfloor + (i \mod \delta) + 1 \dots \delta \lceil i/\delta \rceil]$ are 1. Thus, C gives is a valid answer for $\mathsf{drankA}_1(i, B, \delta)$ in this case. Also by the same argument, one can answer the $\mathsf{selectA}_1(i, B, \delta)$ query in $t(n)$ time by returning $\delta(\mathsf{select}_1(\lfloor i/\delta \rfloor, B') - 1) + (i \mod d)$. □

By combining the data structures of Lemma 2.1 and 2.2 with Theorem 2.5, we obtain the following corollary.

Corollary 2.6. *For a bit-string B of length n, there is a data structure that uses $n/\delta + o(n/\delta)$ bits and supports drankA_1 and $\mathsf{selectA}_1$ queries with additive error δ, in constant time. If there are m 1's in B, the data structure uses $\mathcal{B}(n/\delta, m/\delta) + o(n/\delta)$ bits and supports the queries in $O(1)$ time.*

Note that the proof of Theorem 2.5 implies that any data structure that supports rank_1 (or select_1) queries on B' can be used to answer drankA_1 (or $\mathsf{selectA}_1$) queries on B. For example if B is very sparse, i.e., when $\mathcal{B}(n/\delta, m/\delta) \ll o(n/\delta)$ (in this case, the space usage of the structure of Corollary 2.6 is sub-optimal), one can use the structure of [21] that uses $(m/\delta)\lg(n/m) + O(m/\delta)$ bits (asymptotically optimal space), to support drankA_1 queries in $O(\min\{\lg m, \lg(n/m)\})$ time, and $\mathsf{selectA}_1$ queries in constant time.

Supporting rankA and dselectA queries. Now we consider the problem of supporting rankA_1 and $\mathsf{dselectA}_1$ queries with additive error δ on bit-strings of length n. The following theorem describes a lower bound on space.

Theorem 2.7. *Any data structure that supports rankA_1 or $\mathsf{dselectA}_1$ queries with additive error δ on a bit-string of length n requires at least $\lfloor n/2\delta \rfloor \lg \delta$ bits.*

Proof. We first construct a set V of bit-strings of length n as follows. We divide each bit-string B into $\lfloor n/\delta \rfloor$ blocks $B_1, B_2, \dots, B_{\lfloor n/\delta \rfloor}$ such that for $1 \leq i < \lfloor n/\delta \rfloor$, $B_i = B[\delta(i-1)+1, \dots, \delta i]$ and $B_{\lfloor n/\delta \rfloor} = B[\delta(\lfloor n/\delta \rfloor - 1) + 1, \dots, n]$. Now for every $1 \leq i \leq \lfloor n/\delta \rfloor$, we set all bits in B_i to 0 if i is odd. If i is even, we fill B_i to $k \leq \delta$ 1's followed $(\delta - k)$ 0's. Thus there's only one choice of blocks B_i (if i is odd), and δ choices for blocks B_i (if i is even). Hence $|V| = \delta^{\lfloor n/2\delta \rfloor}$. Now consider two distinct bit-strings B and B' in V, and let i be the smallest even index which satisfies $B_i \neq B'_i$, where B_i and B'_i has k and k' 1s. Without loss of generality, assume k is less than k'. Then by the construction of B and B', there exists no value which can be an answer of both $\mathsf{rankA}_1((i+1)\delta, B, \delta)$ and $\mathsf{rankA}_1((i+1)\delta, B', \delta)$ queries. By the same argument, there exists no position which can be an answer of both $\mathsf{dselectA}_1(\ell, B, \delta)$ and $\mathsf{dselectA}_1(\ell, B', \delta)$ queries, where ℓ is number of 1's upto i-th block in B'. Thus, any structure that supports either of these queries must distinguish between every element in S, which implies $\lg|V| = \lfloor n/2\delta \rfloor \lg \delta$ bits are necessary to answer rankA_1 or $\mathsf{dselectA}_1$ queries. □

We now show that for some values of δ, any data structure that uses up to a $\lg^{O(1)} \delta$ factor more than the optimal space cannot support $\mathsf{dselectA}_1$ queries in constant time.

Theorem 2.8. *Any $((n/\delta)\lg^{O(1)}\delta)$-bit data structure that supports dselectA_1 queries with an additive error $\delta = O(n^c)$, for some constant $0 < c \leq 1$ on a bit-string of length n requires $\Omega(\lg\lg n)$ query time.*

Proof. We reduce the predecessor search problem to the problem of supporting dselectA_1 queries. Given a set $S \subseteq \{1, \ldots, n\}$, a predecessor query, pred(i, S) returns the largest elements in S that is smaller than i, for $1 \leq i \leq n$. Now suppose we want to support pred queries on S with $|S| = n/\delta$, where $\delta = O(n^c)$ with some constant $0 < c \leq 1$. For this range of parameters, Patrascu and Thorup [22] showed that any data structure that represents S using $O(n\lg^{O(1)} n)$ bits needs $\Omega(\lg\lg n)$ time to support pred queries. We now show that any data structure that supports dselectA_1 queries can be used to obtain a data structure that supports pred queries, using asymptotically the same space and query time. The theorem immediately follows from this reduction.

Let $S' = \{k\delta | 1 \leq k \leq \lfloor n/\delta \rfloor\} \cup \{n\}$. We call the elements in S' as the *dummy elements*. Next, let $S_1 = S \cup S'$ with $|S_1| = \ell$, and let $x_1, x_2, \ldots, x_\ell = n$ be the set of all elements of S_1 in increasing order (note that $n/\delta \leq \ell \leq 2n/\delta$ since both S and S' have size n/δ). The dummy elements in S_1 ensure that $x_1 \leq \delta$ and $x_i - x_{i-1} \leq \delta$, for $1 < i \leq \ell$. Now consider the bit-string $B = B_1 B_2 \ldots B_\ell$, where the block $B_1 = 0^{2\delta - x_1} 1^{x_1}$ and $B_i = 0^{2\delta - x_i + x_{i-1}} 1^{x_i - x_{i-1}}$ for $1 < i \leq \ell$ (i.e., B encodes the differences between successive elements of S_1 using fixed-length right-justified unary codes of size 2δ). Note that B contains $x_\ell = n$ 1's, and has length $2\delta\ell \leq 2n$. In addition, we store an array A of length ℓ where $A[i] = \text{pred}(x_i, S)$ using $O((n/\delta)\lg n) = O((n/\delta)\lg\delta$ bits, since $\delta = O(n^c)$.

Suppose that there is a data structure X that uses $s(n, \delta)$ space, and supports dselectA_1 queries on B in $t(n, \delta)$ time. To answer the query pred(x, S), we first perform the dselect$A_1(x, B, \delta)$ on X. Let B_i be the block to which this answer belongs. Since each block starts with a sequence of at least δ zeros, and since dselect$A_1(x, B, \delta) \leq \text{select}_1(x, B)$, it follows that $x_i \leq x < x_{i+1}$. Hence we return $A[x_i]$ as the answer of pred(x). Thus, from the assumption about the data structure X, we can obtain a structure that uses $s(n, \delta) + O((n/\delta)\lg n)$ bits and supports pred queries in $t(n, \delta) + O(1)$ time. The theorem follows from this reduction, and the predecessor lower bound mentioned above. □

The following theorem describes a simple data structure for supporting rankA_1 and dselectA_1 queries.

Theorem 2.9. *For a bit-string B of length n, there is a data structure of size $(n/\delta)\lg\delta + o((n/\delta)\lg\delta)$ bits, which supports rankA_1 queries on B using $O(1)$ time and dselectA_1 queries on B using SPS$(n/\delta, n)$ time.*

Proof. We divide the B into $\lceil n/\delta \rceil$ blocks B_1, B_2, …, $B_{\lceil n/\delta \rceil}$, as in the proof of Theorem 2.5. Also we define an array $C[1 \ldots \lceil n/\delta \rceil]$ of length $\lceil n/\delta \rceil$ where for $1 \leq i \leq \lceil n/\delta \rceil$, $C[i]$ is the number of 1's in B_i. We store the structure of Lemma 2.3 on C using $(n/\delta)\lg\delta + o((n/\delta)\lg\delta)$ bits, to support sum and search queries on C. Then since $j - \delta \leq \delta\lfloor j/\delta \rfloor \leq j$, the answer of sum$(\lfloor j/\delta \rfloor)$ query on C gives the answer of rank$A_1(j, B, \delta)$, which can be answered in $O(1)$ time by Lemma 2.3. To answer the query dselect$A_1(j, B, \delta)$, we first find the block B_i in B which contains the position $\text{select}_1(j, B)$ by answer the $i = \text{search}(j)$ query on C. Since $\text{select}_1(j, B) - \delta < (i-1)\delta \leq \text{select}_1(j, B)$, $(i-1)\delta$ gives an answer of the dselect$A_1(j, B, \delta)$ query. □

2.2. Approximate rank and select queries on multisets

In this section, we describe data structures for answering approximate rank and select queries on a multiset with additive error δ. Given a multiset S where each element is from the universe $U = \{1, 2 \ldots n\}$, the rank and select queries on S are defined as follows.

- rank(i, S): returns the number of elements in S whose value is at most i.
- select(i, S): returns the i-th smallest element in S.

One can define approximate rank and select queries on multisets (also denoted as rankA, drankA, selectA, dselectA) analogously to the queries on strings [24]. Any multiset S of size m from a universe of size n can be represented as a *characteristic vector* B_S of size $m+n$, where $B_S = 1^{m_1}01^{m_2}0 \ldots 1^{m_n}0$ when the element k has multiplicity m_k in S, for $1 \leq k \leq n$. Then one can answer rank(i, S) and select(i, S) queries using rank and select queries on B_S, by $\text{rank}(i, S) = \text{rank}_1(\text{select}_0(i, B_S), B_S) = \text{select}_0(i, B_S) - i$ and $\text{select}(i, S) = \text{rank}_0(\text{select}_1(i, B_S), B_S)$. We now describe our data structures for answering approximate rank and select queries on S, in the following two settings:

(1) rankA, drankA, selectA, and dselectA queries when $|S|=m$ is fixed: We construct a bit-string B'_S of length $\lfloor m/\delta \rfloor + n$ where B'_S only keeps every $i\delta$-th 1 in B_S, for $1 \leq i \leq n/\delta$, and removes all the other 1's in B_S. To answer the drankA(i, S, δ) query, we first compute $\text{select}_0(i, B'_S) - i = \lfloor \text{rank}(i, S)/\delta \rfloor$. Since $\text{rank}(i, S) - \delta \leq \delta \cdot \lfloor \text{rank}(i, S)/\delta \rfloor \leq \text{rank}(i, S)$, we return $\delta(\text{select}_0(i, B'_S) - i)$ as the answer. Similarly, we can answer the selectA(i, S, δ) query by returning $\text{rank}_0(\text{select}_1(\lfloor i/\delta \rfloor, B'_S), B'_S) + 1$. By storing the structure of Lemma 2.2(b) on B'_S using $\mathcal{B}(n + \lfloor m/\delta \rfloor, \lfloor m/\delta \rfloor) + o(n + \lfloor m/\delta \rfloor)$ bits, we can support rank_0, rank_1, select_0 and select_1 queries on B'_S in constant time, which implies both drankA and selectA queries on S also can be supported in constant time.

For answering rankA and dselectA queries on S, we first construct the data structure of Theorem 2.9 on B_S to support dselectA_1 queries. In addition to that, we maintain the data structure of Lemma 2.3 to support sum and search queries on the arrays $D[1 \dots \lceil (n+m)/\delta \rceil]$ and $E[1 \dots \lceil (n+m)/\delta \rceil]$ where for $1 \leq i \leq \lceil (n+m)/\delta \rceil$, $D[i]$ and $E[i]$ store the number of 0's and 1's, respectively, in the block B_{S_i} (as defined in the proof of Theorem 2.9). Thus, the total space of the data structure is $O(((n+m)/\delta)\lg\delta)$ bits. To answer the rankA(i, S, δ) query, we first find the block B_{S_j} of B_S which contains i-th 0 by computing $j = \text{search}(i)$ on D, and then return sum$(j-1)$ on E, which gives an answer of the rankA(i, S, δ) query by the similar argument in the proof of Theorem 2.9. To answer dselectA(i, S, δ), we first find the j-th block B_{S_j} which contains an answer of the dselect$A_1(i, B_S, \delta)$ query, and then return sum$(j-1)$ on D. Note that if $j = 1$, we return 0 for both queries. By Theorem 2.9, the total running time is $\mathsf{SPS}(n+m/\delta, n+m)$ for both rankA and dselectA queries. For special case when $\min\{(n+m)/\delta, \delta\} = polylog(n+m)$, we can answer rankA and dselectA queries on S in constant time.

(2) drankA and selectA queries when the frequency of each element in S is at most ℓ: We describe a data structure for answering drankA and selectA queries on S in $O(1)$ time. We then show that at least $\lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1)+1)$ bits are necessary for supporting drankA queries on S. Thus, for supporting only drankA queries, the data structure uses the optimal space. We consider the following two cases: (a) $\delta \leq \ell$, and (b) $\delta > \ell$.

- **Case 2a. $\delta \leq \ell$:** In this case, we first observe that $|S| \leq n\ell$. Hence, B_S is a bit-string with n 0's and at most $n\ell$ 1's. Let B'_S be a bit-string defined as in the Case (1); B'_S has n 0's and at most $n\ell/\delta$ 1's. To support drankA on S, we need to support select$_0$ on B'_S. We represent the bit-wise complement of B'_S using the structure of Lemma 2.2(a), which takes at most $\mathcal{B}(n + \lfloor n\ell/\delta \rfloor, \lfloor n\ell/\delta \rfloor) + o(n)$ bits and supports select$_0$ on B'_S in $O(1)$ time. Using this structure, we can achieve optimal space usage, and support drankA queries on S in $O(1)$ time. Alternatively, we can represent B'_S using the structure of Lemma 2.2(b), which takes at most $\mathcal{B}(n + \lfloor n\ell/\delta \rfloor, \lfloor n\ell/\delta \rfloor) + o(n + \lfloor n\ell/\delta \rfloor)$ bits, and supports rank$_0$, rank$_1$, select$_0$ and select$_1$ queries on B'_S in $O(1)$ time. Using this structure, we can support both drankA and selectA queries on S in $O(1)$ time, while using asymptotically optimal space when $\ell = \Theta(\delta)$.
- **Case 2b. $\delta > \ell$:** In this case, we first set $\mu = \lfloor \delta/\ell \rfloor$, and define a bit-string $B'[1 \dots \lceil n/\mu \rceil]$ of length $\lceil n/\mu \rceil$ where $B'[i] = 1$ if and only if there exists a 1 between the positions of the $((i-1)\mu)$-th 0 and the $(\min(i\mu, n))$-th 0 in B'_S. Note that there exists at most a single 1 between those two positions since $\mu\ell \leq \delta$. Now using Lemma 2.1, we construct an $(n/\mu + o(n/\mu)) = (n\ell/\delta + o(n\ell/\delta))$-bit data structure which supports rank$_1$ and select$_1$ queries on B' in constant time. Then one can show that $\delta(\text{rank}_1(\lfloor i/\mu \rfloor, B')) + \ell(i \mod \mu)B'[\lceil i/\mu \rceil]$ is an answer to the query drankA(i, S, δ), using an argument similar to the one in the proof of Theorem 2.5. For selectA(i, S, δ) queries, we set $\mu = \lfloor \delta/2\ell \rfloor$ and construct a same structure as above, using $2n\ell/\delta + o(n\ell/\delta)$ bits. Since $\text{select}(i, S) - \mu(\text{select}_1(\lfloor i/\delta \rfloor, B') - 1) \leq 2\mu\ell \leq \delta$, we can answer selectA query in constant time by returning $\mu(\text{select}_1(\lfloor i/\delta \rfloor, B') - 1)$. Therefore, our data structure supports drankA queries in constant time with optimal space, and twice the optimal space for supporting both drankA and selectA queries in constant time (note that at least $\lfloor n/\lceil \delta/\ell \rceil \rfloor$ bits are necessary in this case).

We now show a matching lower bound on space for supporting drankA queries on S.

Theorem 2.10. *Given a multiset S where each element is from the universe $U = \{1, 2, \dots, n\}$ of size n such that the maximum frequency of each element in S is at most ℓ, any data structure that supports drankA queries on S requires at least $\lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1)+1)$ bits.*

Proof. Note that S can be represented by a sequence $S_1, S_2 \dots S_n$ of size n, where $S_i \leq \ell$ denotes a frequency of i in S. Now we first set $\mu = \delta/\ell$ and denote I as $\{\min(\delta k, \ell) | k \in \{0, 1, \dots, \max(\lfloor 1/\mu \rfloor, 1)\} \subset \{0, 1, \dots, \ell\}\}$, and denote $\bar{I}$ as $\{\sigma^{\lceil \mu \rceil} | \sigma \in I\}$. Next, consider all inputs that contains a sequence of $\lfloor n/\lceil \mu \rceil \rfloor$ *blocks* padded by zeros, such that each block is a member of $\bar{I}$; that is, consider $\mathcal{I} = \bar{I}^{\lfloor n/\lceil \mu \rceil \rfloor} \cdot 0^{n-(n \mod \lceil \mu \rceil)}$. It is easy to show that every input of $\mathcal{I}$ gives a representation of S. We show that every two distinct inputs in $\mathcal{I}$ must lead to distinct answer of a drankA query, thereby implying a $\lceil \lg|\mathcal{I}| \rceil$ bits lower bound as required. Let two distinct sets S_1 and S_2 be represented by the sequences in $\mathcal{I}$ such as $x_1 = x_{1,1}x_{1,2} \cdots x_{1,\lfloor n/\lceil \mu \rceil \rfloor} 0^{n-(n \mod \lceil \mu \rceil)}$ and $x_2 = x_{2,1}x_{2,2} \cdots x_{1,\lfloor n/\lceil \mu \rceil \rfloor} 0^{n-(n \mod \lceil \mu \rceil)}$ respectively such that $x_{\alpha,\beta} \in \bar{I}$ for any $\alpha \in \{1, 2\}, \beta \in \{1, \dots, \lfloor n/\lceil \mu \rceil \rfloor\}$. Also let t be a leftmost index which satisfies $x_{1,t} \neq x_{2,t}$. Now we consider drankA$(\lceil \mu \rceil t, S_1)$ and drankA$(\lceil \mu \rceil t, S_2)$ queries. If $\mu \leq 1$, then $\lfloor n/\lceil \mu \rceil \rfloor = n$ and (due to the definition of I) $|x_{1,t} - x_{2,t}| \geq \delta$, which implies that there is no answer which satisfies both drankA$(\lceil \mu \rceil t, S_1)$ and drankA$(\lceil \mu \rceil t, S_2)$ queries. On the other hand, if $\mu > 1$ then $I = \{0, \ell\}$ and thus either $x_{1,t} = 0^{\lceil \mu \rceil}, x_{2,t} = \ell^{\lceil \mu \rceil}$ or $x_{1,t} = \ell^{\lceil \mu \rceil}, x_{2,t} = 0^{\lceil \mu \rceil}$. In either case, $|(\text{drankA}(\lceil \mu \rceil t, S_1) - \text{drankA}(\lceil \mu \rceil t, S_2)| \geq \delta$. We established that if two inputs in $\mathcal{I}$ lead to the same configuration of drankA queries, the error for one of them would be at least δ while we assumed it is strictly lower. □

2.3. Approximate ss and iss queries on binary streams

In this section, we consider a data structure for answering ssA and issA queries over a stream of binary symbols $\{0, 1\}$, when sliding window size is fixed to n. We first describe a data structure for answering exact ss(i, n) and iss(i, n) queries in constant time using $n + o(n)$ bits, while supporting updates in constant time, which is based on the data structure of Lemma 2.1.

Theorem 2.11. *There exists a $n + o(n)$-bit data structure which supports $\mathsf{ss}(i, n)$ and $\mathsf{iss}(i, n)$ queries in $O(1)$ time while supporting $O(1)$-time updates whenever a new element arrives from the stream.*

Proof. We partition the stream into frames of size n, and maintain the current n elements in the sliding window, which span at most two consecutive frames, in a circular array $\mathcal{A}$ using n bits. Clark and Munro [8] showed that one can answer both ss and iss queries in $O(1)$ time on a static bit-string of size n by storing substructures of total size $o(n)$ bits, which consist of (i) some auxiliary arrays, and (ii) a fixed lookup (precomputed) table. The lookup tables are independent of the frames, and can be shared between the frames. The auxiliary arrays store the answers to the ss and iss queries for a subset of the positions/values (i.e., a subset of all possible queries). Furthermore, these auxiliary arrays can be constructed in $O(1)$ time per entry by scanning the original bit-string. Thus, the auxiliary arrays can be easily computed incrementally (i.e., by appending elements at the end), using $O(1)$ time per insertion. Thus, by maintaining the substructures for the two frames spanning the current sliding window using $o(n)$ bits in addition to the $O(\lg n) = o(n)$-bit counter for storing the number of 1's in the current frame, we can support both ss and iss queries in $O(1)$ time, while supporting updates in $O(1)$ time. □

Next, we consider a data structure for answering $\mathsf{ssA}(i, n, \delta)$ and $\mathsf{issA}(i, n, \delta)$ queries on a binary stream in constant time, using $\lceil n/\delta \rceil + O(\lg n) + o(n/\delta)$ bits, while supporting constant time updates. We first divide the stream into consecutive chunks of size δ. For each chunk, we assign a bit as follows: a chunk is assigned the bit 1 if it contains $j\delta$-th 1 of the stream, for some $j \leq n/\delta$, and 0 otherwise. Next, we consider the values assigned to the chunks as a stream, and maintain the data structure of Theorem 2.11 for supporting ss and iss queries in $O(1)$ time over the stream of values of chunks. Also, to support $O(1)$-time update, we maintain two counters $c = (f_i \mod \delta)$ and $t_c = (n_i \mod \delta)$ using $O(\lg \delta)$ bits, where f_i and n_i are the number of elements and the number of 1's in the current from the (original) stream up to position i, respectively. Now we describe how to answer ss and iss queries.

- $\mathsf{ssA}(i, n, \delta)$: We return 0 if $i \leq \delta$. Otherwise, we return $t_c + \delta \cdot \mathsf{ss}(\lfloor ((i - f_i))/\delta \rfloor, \lceil n/\delta \rceil) + \kappa(\lceil \lfloor ((i - f_i))/\delta \rfloor \rceil) \cdot ((i - f_i) \mod \delta)$, where κ is an indicator function defined as $\kappa(k) = 1$ if and only if $k \mod \delta \neq 0$, and the value assigned to the k-th latest chunk is 1. The correctness can be proved by the same argument as in the proof of Theorem 2.5.
- $\mathsf{issA}(i, n, \delta)$: We return 0 if $i \leq \delta$. Otherwise, we return $f_i + \delta \cdot \mathsf{iss}(\lfloor (i - t_c)/\delta \rfloor, \lceil n/\delta \rceil) + ((i - t_c) \mod \delta)$. Again, the correctness can be proved by the same argument as in the proof of Theorem 2.5.

In conclusion, we obtain the following theorem.

Theorem 2.12. *For a binary stream, there exists a data structure that uses $\lceil n/\delta \rceil + O(\lg \delta) + o(n/\delta)$ bits and supports ssA and issA queries on the stream with additive error δ, in constant time, while supporting constant time updates.*

Compared to the lower bound of Theorem 2.4 for answering rankA and $\mathsf{selectA}$ queries on bit-strings (which also gives a lower bound for answering ssA and issA queries), the above data structure takes $\Omega(n/\delta)$ bits when $n/\delta = o(\lg \delta)$. However for a sliding window of size n, at least $\lfloor \lg n \rfloor$ bits are necessary [4] for answering ssA queries, even in the case when i is always equal to n. Therefore the data structure of Theorem 2.12 supports ssA and issA queries with optimal space when $n/\delta = \omega(\lg \delta)$, and with asymptotically optimal space otherwise.

3. Queries on strings and streams over a large alphabet

In this section, we consider non-binary inputs. First, we look at general alphabet and derive results for approximate rank and select. Then we consider approximate ss queries over integer streams.

3.1. drankA and selectA queries on strings over general alphabet

Let A be a string of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$ of size σ. Then, for $1 \leq j \leq \sigma$, the query $\mathsf{rank}_j(i, A)$ returns the number of j's in $A[1 \ldots i]$, and the query $\mathsf{select}_j(i, A)$ returns the position of the i-th j in A (if it exists). Similarly, the queries $\mathsf{drankA}_j(i, A, \delta)$ and $\mathsf{selectA}_j(i, A, \delta)$ are defined analogous to the queries drankA and $\mathsf{selectA}$ on bit-strings. By extending the proof of Theorem 2.4 to strings over larger alphabets, one can show the following.

Corollary 3.1. *Any data structure that supports drankA or selectA queries with additive error δ on a string of length n over an alphabet of size σ requires at least $\lfloor n/\delta \rfloor \lg \sigma$ bits.*

Proof. We divide the string B of length n into $\lfloor n/\delta \rfloor$ blocks $B_1, B_2, \ldots, B_{\lfloor n/\delta \rfloor}$ where for $1 \leq i < \lfloor n/\delta \rfloor$, $B_i = B[\delta(i-1) + 1, \ldots, \delta i]$ and $B_{\lfloor n/\delta \rfloor} = B[\delta(\lfloor n/\delta \rfloor - 1) + 1, \ldots, n]$ (the last block may contain more than δ, but less than 2δ characters). Let S be the set of all possible strings satisfying the condition that all the characters within a block are the same. Then there are $\sigma^{\lfloor n/\delta \rfloor}$ possible strings in S, where for any two distinct strings from S, they have different answers for some rankA_1 query (and also some $\mathsf{selectA}_1$ query), by the same argument as the proof of Theorem 2.4. □

In this section, we describe a data structure that supports drankA and $\mathsf{selectA}$ queries in $O(\lg\lg\sigma)$ time, using twice the optimal space. We make use of the following result from [13] for supporting rank and select queries on strings over large alphabets.

Lemma 3.2 *([13]). Given a string of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, one can support rank_j queries in $O(\lg\lg\sigma)$ time and select_j queries in $O(1)$ time, using $n\lg\sigma + o(n\lg\sigma)$ bits, for any $1 \le j \le \sigma$.*

The following theorem shows we can construct a simple data structure for supporting drankA_j and $\mathsf{selectA}_j$ queries on A using the above lemma.

Theorem 3.3. *Given a string of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$, one can support drankA_j and $\mathsf{selectA}_j$ queries for any $1 \le j \le \sigma$ on the string in $O(\lg\lg\sigma)$ time, using $2n\lg(\sigma+1)/\delta + o((n/\delta)\lg(\sigma+1))$ bits.*

Proof. We first divide the input string, say A, into $\lceil n/\delta\rceil$ blocks $A_1, A_2 \ldots A_{\lceil n/\delta\rceil}$ where for $1 \le i \le \lfloor n/\delta\rfloor$, $A_i = A[\delta(i-1)+1 \ldots \delta i]$ and $A_{\lceil n/\delta\rceil} = A[\delta(\lceil n/\delta\rceil - 1) + 1 \ldots n]$ (if n is not multiple of δ). Then we construct a new string $A' = A_1\$A_2\$\ldots\$A_{\lceil n/\delta\rceil}\$$ of length $n + \lceil n/\delta\rceil$, where \$ is a symbol not in Σ. Now we construct yet another string A'' of length at most $\lfloor n/\delta\rfloor + \lceil n/\delta\rceil$, which is a subsequence of A', obtained by only keeping every $i\delta$-th occurrence of each symbol from Σ, for $1 \ldots i \le \lfloor n/\delta\rfloor$ in A', and also all the occurrences of \$ in A', while removing all the other characters in A'. We then represent A'' using the structure of Lemma 3.2, using $2n\lg(\sigma+1)/\delta + o((n/\delta)\lg(\sigma+1))$ bits to support rank and select queries in $O(\lg\lg(\sigma))$ and $O(1)$ time, respectively.

Now we describe how to support the queries. For answering the $\mathsf{drankA}_j(i, A, \delta)$ query, we first compute the position, $b_{\lfloor i/\delta\rfloor}$, of the $\lfloor i/\delta\rfloor$-th \$ in A'', in constant time, using $b_{\lfloor i/\delta\rfloor} = \mathsf{select}_{\$}(\lfloor i/\delta\rfloor, A'')$. Then by an argument similar to the one in the proof of Theorem 2.5, one can show that $\delta \cdot \mathsf{rank}_j(b_i, A'') + (i \mod \delta)\kappa_j(b_{\lfloor i/\delta\rfloor})$ gives an answer of the $\mathsf{drankA}_j(i, A, \delta)$ query, where $\kappa_j(b_{\lfloor i/\delta\rfloor}) = \mathsf{rank}_j(b_{\lfloor i/\delta\rfloor+1}, A'') - \mathsf{rank}_j(b_{\lfloor i/\delta\rfloor}, A'')$, which is either 0 or 1. Thus, $\mathsf{drankA}_j(i, A, \delta)$ query can be answered in $O(\lg\lg\sigma)$ time. Similarly, it is easy to see that we can answer the $\mathsf{selectA}_j(i, A, \delta)$ query in $O(\lg\lg\sigma)$ time by returning $\delta \cdot \mathsf{rank}_{\$}(\mathsf{select}_j(\lfloor i/\delta\rfloor - 1, A''), A'') + (i \mod \delta)$. □

3.2. Supporting ssA queries over non-binary streams

In this section, we consider the problem of computing suffix sums over a stream of integers in $\{1, 2, \ldots, \ell\}$. This generalizes the result of the Theorem 2.12 for ssA. For such streams, one can use ssA binary search to solve issA, while a constant time issA queries are left as future work. Specifically, we show a data structure that requires $\lfloor n/\lceil\delta/\ell\rceil\rfloor\lg(\max(\lfloor\ell/\delta\rfloor, 1)+1)(1+o(1)) + O(\lg n)$; i.e., it requires $1 + o(1)$ times as many bits as the static-case lower bound of Theorem 2.10 when $\delta = o(\ell \cdot n/\lg n)$.

We note that this model was studied in [4,9,12] for window-sum queries. That is, our work generalizes this model to allow the user to specify the window size $i \le n$ at query time while previous works only considered the sum of the last n elements. In fact, all previous data structure implicitly supports ssA queries but with slower run time. The algorithms in [12,9] require $O(\epsilon^{-1}\lg(\ell n\varepsilon))$ time to compute a $(1+\varepsilon)$-approximation for the sum of the last n elements while the algorithm of [4] needs $O(\ell \cdot n/\delta)$ for a δ-additive one. Here, we show how to compute a δ-additive error for the sum of the last $i \le n$ elements in constant time for both updates and queries.

Exact ss queries En route to ssA, we first discuss how to compute an exact answer for suffix sums queries. It is known, even for fixed window sizes, that one must use $n\lg(\ell+1)$ bits for tracking the sum of a sliding window [4]. Here, we show how to compute exact ssA using succinct space of $n\lg(\ell+1)(1+o(1))$ bits. We start by discussing why the approaches used in Theorem 2.11 cannot work for a large ℓ value. If we use sub-blocks of size $\Theta(\lg n)$ as in [8], then the lookup table will require $(\ell+1)^{\Theta(\lg n)} = n^{\Theta(\lg(\ell+1))}$ bits, which is not even asymptotically optimal for non-constant ℓ values. While one may think that this is solvable by further breaking the sub-blocks into sub-sub-blocks, sub-sub-sub-blocks, etc., it is not the case. To see this, consider a lookup table for sequences of length 2. Then its space requirement will be $(\ell+1)^2$ bits. If ℓ is large (say, $\ell \ge n$) then this becomes $\Omega(n\ell) = \omega(n\lg(\ell+1))$, which is not even asymptotically optimal.

Theorem 3.4. *There exists a data structure that requires $n\lg(\ell+1)(1+o(1))$ bits and support $O(1)$ time ss queries and updates.*

Proof. Same as in the proof of Theorem 2.11, we break the stream into n-sized frames, and store auxiliary structures for each frame to support the queries efficiently. We first divide each frame into $\left\lceil n/\lg^2 n\right\rceil$ equal-sized blocks. The blocks are then sub-divided into $\sqrt{\lg n}$ sized sub-blocks. For each frame, we keep a $\left\lceil n/\lg^2 n\right\rceil$ sized array $\mathcal{C}$ that stores the *sum* of elements from the beginning of the frame, and a $\left\lceil n/\sqrt{\lg n}\right\rceil$-sized array $\mathcal{SC}$ with the sums from the beginning of the corresponding blocks. The number of bits required for $\mathcal{C}$ is $\left\lceil n/\lg^2 n\right\rceil\lg(n\ell+1) \le \lceil n/\lg n\rceil\lg(\ell+1) = o(n\lg(\ell+1))$. Similarly, $\mathcal{SC}$ requires $\left\lceil n/\sqrt{\lg n}\right\rceil\lg\left(\ell\lg^2 n+1\right) = O\left(\frac{n\lg\lg n}{\sqrt{\lg n}}\lg(\ell+1)\right) = o(n\lg(\ell+1))$ bits. Next, we consider the two cases (i)

$\ell+1 \leq 2^{\sqrt[3]{\lg n}}$, and (ii) $\ell+1 > 2^{\sqrt[3]{\lg n}}$, and give separate data structures to handle each case. In both the cases, we store the arrays $\mathcal{C}$ and $\mathcal{SC}$ for the two frames spanning the current (sliding) window. In case (i), we also maintain all the elements in the current window in a circular array $\mathcal{A}$, using $n\lg(\ell+1)$ bits. Since the sub-block size is $\sqrt{\lg n}$, we can store a look-up table for all sequences of size $\leq \sqrt{\lg n}$ as this only requires $O((\ell+1)^{\sqrt{\lg n}}\lg(\ell\lg n)) = O(2^{5\lg n/6}\lg(\ell\lg n)) = o(n\lg(\ell+1))$. To handle an update, we simply need to add the value of the newly added element to the last two entries in the $\mathcal{C}$ and $\mathcal{SC}$ arrays (or create new entries if needed). Thus, by maintaining the array $\mathcal{A}$ and the lookup table in addition to the $\mathcal{C}$ and $\mathcal{SC}$ arrays for the two frames, we get a solution that requires $n\lg(\ell+1)(1+o(1))$ bits and supports suffix sums queries and updates in constant time. In case (ii), the lookup-table approach fails. Thus, we do not use such a table, *nor do we keep the last n-elements window*. Instead, we keep an n-sized circular array $\mathcal{B}$ in which every entry is the cumulative sum from the beginning of the sub-block. The space required for keeping the array is then $n\lg\left(\ell\sqrt{\lg n}+1\right) \leq n\lg(\ell+1)\left(1+\frac{\sqrt{\lg n}}{\ell+1}\right) = n\lg(\ell+1)(1+o(1))$. Thus by keeping this array $\mathcal{B}$ in addition to the $\mathcal{C}$ and $\mathcal{SC}$ arrays for the two frames, we get a solution that takes $n\lg(\ell+1)(1+o(1))$ bits and supports constant time updates and queries. □

General ssA queries Here, we consider the general problem of computing ssA (i.e., up to an additive error of δ). Intuitively, we apply the exact solution from the previous section on a compressed stream that we construct on the fly. A simple approach would be to divide the streams into consecutive chunks of size $\max(\lfloor\mu\rfloor, 1) = \max(\lfloor\delta/\ell\rfloor, 1)$ and represent each chunk's sum as an input to an exact suffix sum algorithm, same as in the binary stream case. However, this fails to achieve succinct space. For example, summing $\lceil\delta/\ell\rceil$ integers requires $O(\lceil\delta/\ell\rceil\lg(\ell+1)) = \Omega(\lg\ell)$ bits. However, $\lg\ell$ bits may be asymptotically larger than the $\lfloor n/\lceil\mu\rceil\rfloor\lg(\max(\lfloor 1/\mu\rfloor, 1)+1)$ bits lower bound of Theorem 2.10. We alleviate this problem by *rounding* the arriving elements. Namely, when adding an input $x \in \{0,1,\ldots,\ell\}$, we first round its value to $Round_{\mathfrak{b}}(x) = 2^{-\mathfrak{b}}\ell \cdot \left\lfloor \frac{x2^{\mathfrak{b}}}{\ell} \right\rfloor$ so it will require $\mathfrak{b} = \lceil\lg(n/\mu) + \lg\lg n\rceil$ bits.[2] The rounding allows us to sum elements in a chunk (using a variable denoted by $\mathfrak{r}$), but introduces a rounding error. To compensate for the error, we use chunks of size $\nu = \max\{\lfloor\mu\cdot(1-1/\lg n)\rfloor, 1\}$. We also consider $\widetilde{\delta} = \lfloor\delta\cdot(1-1/\lg n)\rfloor$ that is slightly lower than δ to compensate for the rounding error when $\mu \leq 1$.[3] We then employ the exact suffix sums construction of Theorem 3.4 for window size of $s = \lceil n/\nu + 1\rceil$ (the number of chunks that can overlap with the window) over a stream of integers in $\{0,1,\ldots,z\}$, where $z = \max(\lfloor\mu^{-1}\nu\rfloor, 1)$ is a upper bound on the resulting items. We use ρ to denote the input that represents the current block.

The query procedure is also a bit tricky. First, we introduce the following variables used in our algorithm for ssA queries:

- $\mathbb{A}$ - an exact suffix sum algorithm, as described in the previous section. It allows computing suffix sums over the last $s = \lceil n/\nu + 1\rceil$ elements on a stream of integers in $\{0,1,\ldots,z\}$.
- $\mathfrak{r}$ - tracks the sum of elements that is not yet recorded in $\mathbb{A}$.
- o - the offset within the chunk.

Intuitively, we can estimate the sum of the last i items by querying $\mathbb{A}$ for the sum of the last i/ν inserted values and multiplying the result by $\widetilde{\delta}$; but there are a few things to keep in mind which were not considered in the binary case. First, i/ν may not be an integer. Next, the values within the current chunk (that has not ended yet) are not recorded in $\mathbb{A}$. Finally, we are not allowed to overestimate, so $\mathfrak{r}$'s propagation may be an issue. To address the first issue, we weigh the oldest chunk's ρ value by the fraction of that chunk that is still in the window. For the second, we add the value of $\mathfrak{r}$ to the estimation, where $\mathfrak{r}$ is the sum of rounded elements. Notice that we do not reset the value of $\mathfrak{r}$ but rather propagate it between chunks. Finally, to assure that our algorithm never overestimates we subtract $\widetilde{\delta} - 1/2$ from the result. A pseudo code of our method appears in Algorithm 1. The space usage of the algorithm is analyzed in the following lemma.

Lemma 3.5. *Algorithm 1 requires* $(1+o(1))\cdot\lfloor n/\max(\lfloor\mu\rfloor, 1)\rfloor\cdot\lg\left(\lceil\mu^{-1}\rceil+1\right) + O(\lg n)$ *bits.*

Proof. The algorithm utilizes three variables: $\mathbb{A}$ that requires $(1+o(1))\cdot s\log(z+1)$ bits, $\mathfrak{r}$ that uses $O(\mathfrak{b}\log\nu)$ space, and o that is allocated with $\lceil\log n\rceil$ bits. Recall that $s = \lceil n/\nu+1\rceil$ is the number of blocks that can overlap with the maximal n-sized window and $z = \lfloor\mu^{-1}\nu\rfloor$ is a bound on ρ. Overall, the number of bits used by our construction is

$$\begin{aligned}&(1+o(1))\cdot s\log(z+1) + O(\mathfrak{b}\log\nu) + \lceil\log n\rceil\\ =&(1+o(1))\cdot\lceil n/\nu+1\rceil\log\left(\lfloor\mu^{-1}\nu+1\rfloor+1\right) + O(\lceil\log(n/\mu)+\log\log n\rceil\log\nu) + O(\log n).\end{aligned}$$

Since $\nu = \max(\lfloor\mu\cdot(1-o(1))\rfloor, 1)$, we get the desired bound. □

[2] That is, we encode the integer $\left\lfloor\frac{x2^{\mathfrak{b}}}{\ell}\right\rfloor$ and whenever needed multiply the value by $2^{-\mathfrak{b}}\ell$.

[3] If $\widetilde{\delta} = 1$, then we simply apply the exact algorithm from the previous subsection.

Algorithm 1 Algorithm for ssA and update on stream.

```
1: Initialization: τ ← 0, o ← 0, 𝔸.init()
2: function Add(element x)                                  ▷ Add an element x into sliding window
3:     o ← (o + 1) mod ν
4:     τ ← τ + ⌊x2^b / ℓ⌋
5:     if o = 0 then                                        ▷ End of a chunk
6:         ρ ← ⌊2^{-b} · ℓ · δ̃^{-1} · τ⌋
7:         τ ← τ − ⌊δ̃ · 2^b · ℓ^{-1} · ρ⌋
8:         𝔸.Add(ρ)
9: function ssA(i, n, δ)
10:    if i ≤ o then                                        ▷ Queried within the current chunk
11:        return τ − (δ̃ − 1/2)
12:    else
13:        numElems ← ⌈(i − o)/ν⌉
14:        totalSum ← 𝔸.ss(numElems, s)
15:        oldest_ρ ← totalSum − 𝔸.ss(numElems − 1, s)
16:        out ← (ν − ((i − o) mod ν))
17:        return τ · 2^{-b} · ℓ − (δ̃ − 1/2) + δ̃ · totalSum − ℓ · oldest_ρ · out
```

Thus, we conclude that our algorithm is succinct if the (additive) error δ satisfies $o(\ell \cdot n/\lg n)$. We note that a $\lfloor \lg n \rfloor$ bits lower bound for Basic-Summing with an additive error was shown in [4], even when only fixed sized windows (where i is always n) are considered. Thus, our algorithm always requires $O(\mathcal{B}_{\ell,n,\delta})$ space, even if $\delta = \Omega(\ell \cdot n/\lg n)$. Here, $\mathcal{B}_{\ell,n,\delta} = \lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1) + 1)$ is the lower bound for static data shown in Theorem 2.10.

Corollary 3.6. *Let $\ell, n, \delta \in \mathbb{N}^+$ such that $\mu = \delta/\ell$ satisfies*

$$(\mu = o(n/\lg n)) \wedge \left[(\mu = o(1)) \vee (\mu = \omega(1)) \vee (\mu \in \mathbb{N}) \vee (\mu^{-1} \in \mathbb{N})\right],$$

then Algorithm 1 is succinct. For other parameters, it uses $O(\mathcal{B}_{\ell,n,\delta})$ space.

We now state the correctness of our algorithm.

Theorem 3.7. *Algorithm 1 solves ssA while processing elements and answering queries in $O(1)$ time.*

Proof. For the proof, we recall a few quantities that we also use in Algorithm 1: $numElems = \left\lceil \frac{i-o}{\nu} \right\rceil$, $totalSum = \mathbb{A}.\text{ss}(numElems, s)$, $oldest_\rho = totalSum - \mathbb{A}.\text{ss}(numElems - 1, s)$ and $out = (\nu - ((i - o) \mod \nu))$. We assume that the index of the most recent element is $h = out + i$, such that x_1 is the first element in the chunk of $oldest_\rho$ and $o = (oldest_\rho \mod \nu)$ is the offset within the current chunk. We also denote $g = h - o$, such that x_g is the last element of the most recently completed chunk. Fig. 1 illustrates the setting. By the correctness of the $\mathbb{A}$ exact suffix sum algorithm, and as illustrated in Fig. 1, we have that *totalSum* is the sum of the last *numElems* added to $\mathbb{A}$, that $oldest_\rho$ is the value of the element that represents the last chunk that overlaps with the queried window. Also, notice that *out* is the number of elements in that chunk that are not a part of the window. For any $t \in \mathbb{N}$, we denote by τ_t the value of τ *after* the t^{th} item was added; e.g., $\tau_\mathfrak{h}$ is the value of τ at the time of the query and $\tau_\mathfrak{g}$ is its value before the current chunk. Notice that $\tau_\mathfrak{o}$ is also at the end of a chunk (that does not overlap with the queried interval). For other variables, we consider their value at query time.

When a chunk ends (Line 5), we effectively perform $\tau \leftarrow \tau \mod \widetilde{\delta}$ (lines 6 and 7), thus:

$$0 \leq \tau_\mathfrak{o} \leq \widetilde{\delta} - 1. \tag{1}$$

Our goal is to estimate the quantity

$$S_i = \sum_{d=h-i+1}^{h} x_d = \sum_{d=out+1}^{h} x_d. \tag{2}$$

Recall that our estimation (Line 17) is:

$$\begin{aligned}\widehat{S_i} &= \tau_\mathfrak{h} \cdot 2^{-\mathfrak{b}} \cdot \ell - \left(\widetilde{\delta} - 1/2\right) + \widetilde{\delta} \cdot totalSum - \ell \cdot oldest_\rho \cdot out \\ &= \tau_\mathfrak{g} + \sum_{d=g+1}^{h} Round_\mathfrak{b}(x_d) - \left(\widetilde{\delta} - 1/2\right) + \widetilde{\delta} \cdot totalSum - \ell \cdot oldest_\rho \cdot out, \end{aligned} \tag{3}$$

where the last equality follows from the fact that within a chunk we simply sum the rounded values (Line 4). Next, observe that we sum the rounded values in each chunk and that if τ is decreased by $k \cdot \widetilde{\delta}$ (for some $k \in \mathbb{N}$) at Line 7, then we set one of the last *numElems* elements added to $\mathbb{A}$ to k. This means that:

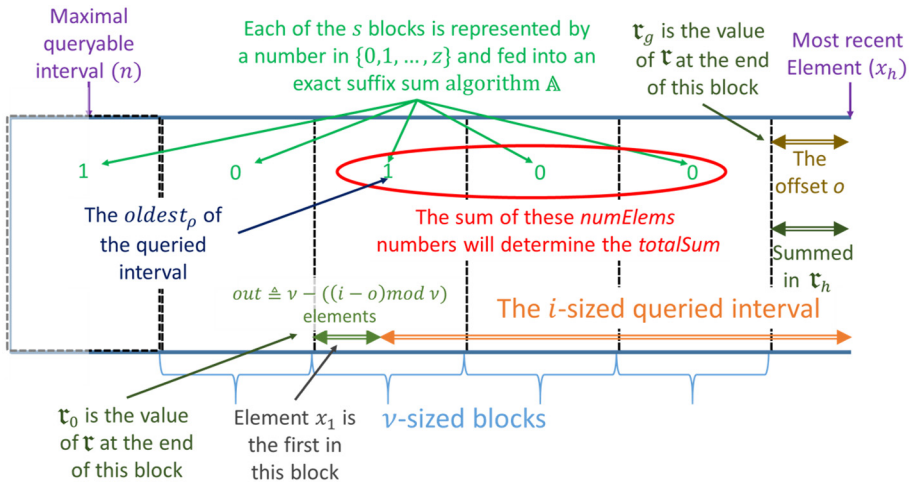

Fig. 1. Theorem 3.7 proof's setting, with all relevant quantities that Algorithm 1 uses illustrated.

$$\mathfrak{r}_o + \sum_{d=1}^{g} Round_{\mathfrak{b}}(x_d) = \mathfrak{r}_{\mathfrak{g}} + \widetilde{\delta} \cdot \mathbb{A}.\mathtt{ss}(numElems, s) = \mathfrak{r}_{\mathfrak{g}} + \widetilde{\delta} \cdot totalSum. \tag{4}$$

Plugging (4) into (3) gives us

$$\widehat{S_i} = \mathfrak{r}_o + \sum_{d=1}^{g} Round_{\mathfrak{b}}(x_d) + \sum_{d=g+1}^{h} Round_{\mathfrak{b}}(x_d) - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot oldest_\rho \cdot out. \tag{5}$$

Joining (5) with (2), we can express the algorithm's error as:

$$\begin{aligned}\widehat{S_i} - S_i &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_{\mathfrak{b}}(x_d) + \sum_{d=out+1}^{h} \left(Round_{\mathfrak{b}}(x_d) - x_d\right) - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot oldest_\rho \cdot out \\ &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_{\mathfrak{b}}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot oldest_\rho \cdot out, \end{aligned} \tag{6}$$

where ξ is the rounding error, defined as $\xi = \sum_{d=out+1}^{h} \left(Round_{\mathfrak{b}}(x_d) - x_d\right)$.

Since each rounding of an integer $x \in \{0, 1, \ldots, \ell\}$ has an error of at most $\frac{\ell}{2^{\mathfrak{b}}}$, and as we round $i \leq n$ elements, we have that the rounding error satisfies

$$0 \geq \xi \geq 0 - \frac{\ell \cdot n}{2^{\mathfrak{b}}} \geq -\delta / \lg n, \tag{7}$$

where the last inequality is immediate from our choice of the number of bits – $\mathfrak{b} = \lceil \lg(n/\mu) + \lg\lg n \rceil$. We now split to cases based on the value of μ. We start with the simpler $\mu < 2 \cdot (1 - 1/\lg n)$ case, in which $\nu = 1$ (and consequently, $out \equiv 0$). This allows us to express the algorithm's error of (6) as

$$\widehat{S_i} - S_i = \mathfrak{r}_o + \xi - \left(\widetilde{\delta} - 1/2\right). \tag{8}$$

We now use (1), (7), and the definition of $\widetilde{\delta}$ to obtain:

$$\widehat{S_i} - S_i = \mathfrak{r}_o + \xi - \left(\widetilde{\delta} - 1/2\right) \leq -1/2.$$

Similarly, we can bound it from below:

$$\widehat{S_i} - S_i = \mathfrak{r}_o + \xi - \left(\widetilde{\delta} - 1/2\right) \geq \xi - \left(\widetilde{\delta} - 1/2\right) \geq -\delta + 1/2.$$

We established that if $\nu = 1$ we achieve the desired approximation. Henceforth, we focus on the case where $\mu \geq 2 \cdot (1 - 1/\lg n)$, which means that $\nu = \lfloor \mu \cdot (1 - 1/\lg n) \rfloor > 1$ and $oldest_\rho \in \{0, 1\}$. We now consider two cases, based on the value of $oldest_\rho$.

1. ***oldest*$_\rho$=1 case.**
In this case, we know that after the processing of element x_v the value of $\mathfrak{r}$ was at least $\widetilde{\delta}$ (Line 6). This implies that $\mathfrak{r}_o + \sum_{d=1}^{v} Round_\mathfrak{b}(x_d) \geq \widetilde{\delta}$ and equivalently

$$\mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) \geq \widetilde{\delta} - \sum_{d=out+1}^{v} Round_\mathfrak{b}(x_d).$$

Substituting this in (6), and applying (7), we get that:

$$\begin{aligned}\widehat{S}_i - S_i &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot out \\ &\geq \widetilde{\delta} - \sum_{d=out+1}^{v} Round_\mathfrak{b}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot out \\ &\geq -\left(\sum_{d=out+1}^{v} \ell\right) + \xi + 1/2 - \ell \cdot out \\ &\geq -\delta/\lg n - \ell \lfloor \mu \cdot (1 - 1/\lg n) \rfloor + 1/2 \geq -\delta + 1/2.\end{aligned}$$

In order to bound the error from above we use (1) and (7):

$$\begin{aligned}\widehat{S}_i - S_i &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot out \\ &\leq \widetilde{\delta} - 1 + \ell \cdot out - \left(\widetilde{\delta} - 1/2\right) - \ell \cdot out \leq -1/2.\end{aligned}$$

2. ***oldest*$_\rho$=0 case.**
Here, since the value of *oldest*$_\rho$ is 0, we have that $\mathfrak{r}_o + \sum_{d=1}^{v} Round_\mathfrak{b}(x_d) < \widetilde{\delta}$ and thus

$$\mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) \leq \widetilde{\delta} - \sum_{d=out+1}^{v} Round_\mathfrak{b}(x_d) - 1.$$

We use this for the error expression of (6) to get:

$$\begin{aligned}\widehat{S}_i - S_i &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) \\ &\leq \widetilde{\delta} - \sum_{d=out+1}^{v} Round_\mathfrak{b}(x_d) - 1 + \xi - \left(\widetilde{\delta} - 1/2\right) \leq -1/2\end{aligned}$$

We now use (1), (7), and the fact that $out \leq v$ to bound the error from below as follows:

$$\begin{aligned}\widehat{S}_i - S_i &= \mathfrak{r}_o + \sum_{d=1}^{out} Round_\mathfrak{b}(x_d) + \xi - \left(\widetilde{\delta} - 1/2\right) \\ &\geq \xi - \left(\widetilde{\delta} - 1/2\right) \geq -\delta + 1/2.\end{aligned}$$

Finally, we need to cover the case of $i \leq o$. In this case, we can return $\mathfrak{r} - \left(\widetilde{\delta} - 1/2\right)$ as the estimation. This directly follows from (1) and the fact that within a chunk we simply sum the rounded values (Line 4). We established that in all cases $-\delta < \widehat{S}_i - S_i \leq 0$. □

4. Conclusion

In this paper, we considered the problem of supporting approximate versions of the well-studied rank/select problems over static sets and suffix sums and inverse suffix sums over sliding windows. We studied the generalization of these problems to multi-sets and examined different error guarantees, such as error in the input vs. error in the result. Most of our results include lower bounds and matching upper bounds that are asymptotically optimal and often succinct. The resulting solutions require considerably less space compared to the space required for supporting exact queries, which makes them more likely to be implemented in practice.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

References

[1] E. Asaf, R. Ben-Basat, G. Einziger, R. Friedman, Optimal elephant flow detection, in: IEEE INFOCOM 2018, 2018, pp. 1–9.
[2] R. Ben-Basat, G. Einziger, R. Friedman, Fast flow volume estimation, Pervasive Mob. Comput. 48 (2018) 101–117.
[3] R. Ben-Basat, G. Einziger, R. Friedman, Give me some slack: efficient network measurements, in: MFCS, 2018, 34.
[4] R. Ben-Basat, G. Einziger, R. Friedman, Y. Kassner, Efficient summing over sliding windows, in: SWAT, 2016, 11.
[5] R. Ben-Basat, G. Einziger, R. Friedman, Y. Kassner, Heavy hitters in streams and sliding windows, in: IEEE INFOCOM, 2016, pp. 1–9.
[6] R. Ben-Basat, R. Friedman, R. Shahout, Heavy hitters over interval queries, CoRR, arXiv:1804.10740 [abs], 2018.
[7] R. Ben-Basat, S. Jo, S.R. Satti, S. Ugare, Approximate query processing over static sets and sliding windows, in: 29th International Symposium on Algorithms and Computation, ISAAC, 2018, 54.
[8] D.R. Clark, J.I. Munro, Efficient suffix trees on secondary storage, in: SODA, 1996, pp. 383–391.
[9] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, SIAM J. Comput. 31 (2002) 1794–1813.
[10] H. El-Zein, J.I. Munro, Y. Nekrich, Succinct color searching in one dimension, in: ISAAC, 2017, 30.
[11] É. Fusy, F. Giroire, Estimating the number of active flows in a data stream over a sliding window, in: ANALCO, 2007, pp. 223–231.
[12] P.B. Gibbons, S. Tirthapura, Distributed streams algorithms for sliding windows, in: SPAA, 2002, pp. 63–72.
[13] A. Golynski, J.I. Munro, S.S. Rao, Rank/select operations on large alphabets: a tool for text indexing, in: SODA, 2006, pp. 368–373.
[14] A. Golynski, A. Orlandi, R. Raman, S.S. Rao, Optimal indexes for sparse bit vectors, Algorithmica 69 (2014) 906–924.
[15] W. Hon, K. Sadakane, W. Sung, Succinct data structures for searchable partial sums with optimal worst-case performance, Theor. Comput. Sci. 412 (2011) 5176–5186.
[16] G.J. Jacobson, Succinct Static Data Structures, Ph.D. thesis, AAI8918056, Carnegie Mellon University, Pittsburgh, PA, USA, 1988.
[17] S. Jo, S. Joannou, D. Okanohara, R. Raman, S.R. Satti, Compressed bit vectors based on variable-to-fixed encodings, Comput. J. 60 (2017) 761–775.
[18] P.B. Miltersen, Cell probe complexity - a survey, in: FSTTCS, 1999.
[19] J. Munro, V. Raman, S.S. Rao, Space efficient suffix trees, J. Algorithms 39 (2001) 205–222.
[20] G. Navarro, E. Providel, Fast, small, simple rank/select on bitmaps, in: SEA, 2012, pp. 295–306.
[21] D. Okanohara, K. Sadakane, Practical entropy-compressed rank/select dictionary, in: ALENEX, 2007, pp. 60–70.
[22] M. Pătraşcu, M. Thorup, Time-space trade-offs for predecessor search, in: ACM STOC, 2006, pp. 232–240.
[23] R. Raman, V. Raman, S.S. Rao, Succinct dynamic data structures, in: WADS, 2001, pp. 426–437.
[24] R. Raman, V. Raman, S.R. Satti, Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets, ACM Trans. Algorithms 3 (2007) 43.