Approximate Query Processing over Static Sets and Sliding Windows

Ran Ben Basat[1]
Harvard University, Cambridge, USA
ran@seas.harvard.edu

Seungbum Jo[2]
University of Siegen, Germany
Seungbum.Jo@uni-siegen.de
https://orcid.org/0000-0002-8644-3691

Srinivasa Rao Satti
Seoul National University, South Korea
ssrao@cse.snu.ac.kr
https://orcid.org/0000-0003-0636-9880

Shubham Ugare
IIT Guwahati, Guwahati, India
ugare.dipak@iitg.ac.in

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.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Streaming, Algorithms, Sliding window, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.54

Related Version A full version of the paper is available at [1], `https://arxiv.org/abs/1809.05419`.

1 Introduction

Given a bit-string $B[1\ldots n]$ of size n, one of the fundamental and well-known problems proposed by Jacobson [12], 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.

[1] Supported by the Zuckerman Foundation, the Technion Hiroshi Fujiwara cyber security research center, and the Israel Cyber Directorate.
[2] The author of this paper is supported by the DFG research project LO748/11-1.

- $\mathsf{rank}_b(i, B)$: 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 [13, 15, 16, 17, 19]. Some of these results also explore trade-offs that allow more query time while reducing the space.

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 $i \in \{1, \dots, n\}$, 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 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 [14] 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. Jacobson [12] proposed a data structure for answering rank queries on B in constant time using $n + o(n)$ bits. Clark and Munro [5] 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 $\mathcal{B}(n, m)$ bits[3] are necessary

[3] $\mathcal{B}(n, m) = \lg \left\lceil \binom{n}{m} \right\rceil$ bits is the information-theoretic lower bound on space for storing a subset of size $m \leq n$ from the universe $\{1, 2, \dots n\}$.

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$_1$, selectA$_1$	$\lfloor n/\delta \rfloor$		δ, additive
drankA$_1$, selectA$_1$	$\mathcal{B}(\lfloor n/\delta \rfloor, \lfloor m/\delta \rfloor)$		
rankA$_1$, dselectA$_1$	$\lfloor n/2\delta \rfloor \lg \delta$		
dselectA$_1$	$O((n/\delta) \lg^{O(1)} \delta)$	$\Omega(\lg\lg n)$	
	Upper bounds		
drankA$_1$, selectA$_1$	$n/\delta + o(n/\delta)$	$O(1)$	δ, additive
drankA$_1$, selectA$_1$	$\mathcal{B}(n/\delta, m/\delta) + o(n/\delta)$		
rankA$_1$	$(n/\delta) \lg \delta + o((n/\delta) \lg \delta)$		
dselectA$_1$	$(n/\delta) \lg \delta + o((n/\delta) \lg \delta)$	$t(n/\delta, n)$	

to support rank and select on B. Raman et al. [19] 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. [10] gave an asymptotically optimal time-space trade-off for supporting rank and select queries on B. A slightly related problem of *approximate color counting* has been considered in El-Zein et al. [7].

Algorithms that Sum over Sliding Windows. 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 $i \le n$ 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. [6]. 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 amortized time $O(\lg \ell/\lg n)$ or $O(\lg(\ell \cdot n))$ worst case. In [8], Gibbons and Tirthapura presented a $(1 + \varepsilon)$ multiplicative approximation algorithm that operates in constant worst case time while using similar space for $\ell = n^{O(1)}$. [3] 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, [2] 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. [4] 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 [6] and [8]. We can also use the data structure of [3] for an additive approximation of δ in $O(n\ell/\delta)$ time.

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. In this case, we first show that $\lfloor n/\delta \rfloor$ bits are necessary for answering drankA$_1$ and selectA$_1$ queries on B and propose a $(\lceil n/\delta \rceil + o(n/\delta))$-bit data structure that supports drankA$_1$ and selectA$_1$ queries on B in constant time. For the

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 [6]	$(1+\varepsilon)$-multiplicative	$O(\varepsilon^{-1}\lg(\ell n)\lg(n\lg\ell))$	$O(\lg(\ell n))$	$O(\varepsilon^{-1}\lg(\ell n\varepsilon))$
GT02 [8]	$(1+\varepsilon)$-multiplicative	$O(\varepsilon^{-1}\lg^2(\ell n))$	$O(1)$	$O(\varepsilon^{-1}\lg(\ell n\varepsilon))$
BEFK16 [3]	δ-additive, for $\delta=\Omega(\ell)$	$\Theta(\ell\cdot n/\delta+\lg n)$	$O(1)$	$O(\ell\cdot n/\delta)$
BEFK16 [3]	δ-additive, for $\delta=o(\ell)$	$\Theta(n\lg(\ell/\delta))$	$O(1)$	$O(n)$
This paper	δ-additive	Same as in [3]	$O(1)$	$O(1)$

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. For rankA_1 and $\mathsf{dselectA}_1$ queries on B, we show that $\lfloor n/2\delta\rfloor\lg\delta$ bits are necessary for answering both queries, 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 $|\{j\in S|j\le i\}|$, 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 elements in S is at most ℓ. Furthermore 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 an $(2n/\delta\lg(\sigma+1)+o((n/\delta)\lg(\sigma+1))$-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 [5] 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 [3], the space usage of our data structure is succinct (i.e., optimal upto 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 $n/\delta=\omega(\lg n/\ell)$, and is asymptotically optimal otherwise, and improves the query time of [3] while using the same space. Table 2 presents this comparison.

2 Queries on bit-strings

In this section, we first consider the 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 the sliding windows on binary streams, for answering approximate ss and iss queries.

2.1 Approximate rank and select queries on bit-strings

We now 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 1** ([5]). *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** ([19]). *For 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 query 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 [11] and [18], which describe efficient data structures for supporting the following queries on integer arrays. For a 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 $\mathsf{sum}(i) > x$. We use the following function to state the running time of some of the (*Searchable Partial Sum*) queries in the lemma below, and in the rest of the paper.

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

▶ **Lemma 3** ([11], [18]). *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 $SPS(n, n2^{\alpha})$ 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. We first prove a lower bound on space used by any data structure that supports either of these two queries.

▶ **Theorem 4.** *Any data structure that supports drankA_1 or $\mathsf{selectA}_1$ 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 $B_1, B_2, \ldots B_{\lfloor n/\delta \rfloor}$ such that 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 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., either all zeros or all ones). Then it is easy to see that $|S| = 2^{\lfloor n/\delta \rfloor}$. 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 index of the leftmost block such that $B_i \neq B'_i$. Then it is easy to show that there is no value which is the answer of both $\mathsf{drankA}_1(i\delta, B, \delta)$ and $\mathsf{drankA}_1(i\delta, B', \delta)$ queries and also there is no position of B which is the 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, and hence $\lfloor n/\delta \rfloor$ 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 and $\mathsf{selectA}_1$ queries in this case. ◀

Now we describe a data structure for supporting drankA_1 and $\mathsf{selectA}_1$ queries in constant time, using optimal space.

▶ **Theorem 5.** *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.*

Proof. We divide the B into $\lceil n/\delta \rceil$ blocks B_1, B_2, $\dots B_{\lceil n/\delta \rceil}$ such that 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 a new bit-string B' of length $\lceil n/\delta \rceil$ such that for $1 \leq i \leq \lceil n/\delta \rceil$, $B'[i] = 1$ if B_i contains $j\delta$-th 1 in B for any $j \leq i$, and otherwise $B'[i] = 0$ (note that for any $1 \leq j \leq \lceil n/\delta \rceil$, any block of B has at most one position of $j\delta$-th 1 in B). By Lemma 1, we can support rank_1 and select_1 queries on B' in constant time, using $n/\delta + o(n/\delta)$ bits. Now we claim that $C = \delta \cdot \mathsf{rank}_1(\lfloor i/\delta \rfloor) + (i \mod \delta) B'[\lceil i/\delta \rceil]$ gives an answer of the $\mathsf{drankA}_1(i, B, \delta)$ query. Let $D = \delta \cdot \mathsf{rank}_1(\lfloor i/\delta \rfloor)$, and let d be the position of D-th 1 in B. From the definition of B', we can easily show that 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 $B[d \dots i]$ 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 $B[d \dots i]$ 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. By the similar argument, we can show that one can answer the $\mathsf{selectA}_1(i, B, \delta)$ query in $O(1)$ time by returning $\delta(\mathsf{select}_1(\lfloor i/\delta \rfloor, B') - 1) + (i \mod d)$.

Finally, in the case when there are m 1's in B, there are at most $\lfloor m/\delta \rfloor$ 1's in B'. Therefore by Lemma 2(b), we can support drankA_1 and $\mathsf{selectA}_1$ queries (as before) in $O(1)$ time, using $\mathcal{B}(\lceil n/\delta \rceil, \lfloor m/\delta \rfloor) + o(n/\delta)$ bits. ◀

Note that in the above proof, we can answer drankA_1 (or $\mathsf{selectA}_1$) queries on B using any data structure that supports rank_1 (or select_1) queries on B'. Thus, 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 Theorem 5 is sub-optimal), one can use the structure of [17] 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 6** ($\star$[4])**.** *Any data structures 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.*

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 7** ($\star$)**.** *Any* $((n/\delta) \lg^{O(1)} \delta)$*-bit data structure that supports* $\mathsf{dselectA}_1$ *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.*

The following theorem describes a simple data structure for supporting $\mathsf{dselectA}_1$ queries.

▶ **Theorem 8** ($\star$)**.** *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* $\mathsf{dselectA}_1$ *queries on* B *using* $SPS(n/\delta, n)$ *time.*

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.

- $\mathsf{rank}(i, S)$: returns the value $|\{j \in S | j \leq i\}|$.
- $\mathsf{select}(i, S)$: 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. Any multiset S of size m can be represented as a *characteristic vector* B_S of size $m+n$, such that $B_S = 1^{m_1}01^{m_2}0 \ldots 1^{m_n}0$ when the element k has multiplicity m_k in the multiset S, for $1 \leq k \leq n$. It is easy to show that by answering rank_b and select_b queries on B_S, for $b \in \{0, 1\}$, one can answer rank and select queries on S. We now describe efficient structures for the following two cases.

(1) rankA, drankA, selectA, and dselectA queries when $|S| = m$ is fixed. We construct a new string B'_S of length $\lfloor m/\delta \rfloor + n$ such that B'_S only keeps every $i\delta$-th 1 from B_S, for $1 \leq i \leq \lfloor n/\delta \rfloor$ (and removes all other 1's). To answer the query $\mathsf{drankA}(i, S, \delta)$, we first compute $\mathsf{select}_0(i, B'_S) - i = \lfloor \mathsf{rank}(i, S)/\delta \rfloor$, and return $\delta(\mathsf{select}_0(i, B'_S) - i)$ as the answer. It is easy to see that $\delta \lfloor \mathsf{rank}(i, S)/\delta \rfloor$ is an answer to the $\mathsf{drankA}(i, S, \delta)$ query. Similarly, we can answer the $\mathsf{selectA}(i, S, \delta)$ query by returning $\mathsf{rank}_0(\mathsf{select}_1(\lfloor i/\delta \rfloor, B'_S), B'_S) + 1$. We represent B'_S using the structure of Lemma 2(b), which uses $\mathcal{B}(n + \lfloor m/\delta \rfloor, \lfloor m/\delta \rfloor) + o(n + \lfloor m/\delta \rfloor)$ bits and supports rank_0, rank_1, select_0 and select_1 queries on B'_S in constant time. Thus, both drankA and selectA queries on S can be supported in constant time.

For answering rankA and dselectA queries on S, we first construct the data structure of Theorem 8 to support $\mathsf{dselectA}_1$ queries on B_S. In addition, we maintain the data structure of Lemma 3 to support sum and search queries on arrays $D[1 \ldots \lceil (n+m)/\delta \rceil]$ and $E[1 \ldots \lceil (n+m)/\delta \rceil]$ which are defined as follows. For $1 \leq i \leq \lceil (n+m)/\delta \rceil$, $D[i]$ and $E[i]$ stores the number of 0's and 1's in the block B_{S_i} respectively (as defined in the proof of Theorem 8). By Lemma 3 and Theorem 8, the total space for this data structure is $O((n'/\delta) \lg \delta)$ bits. To answer $\mathsf{rankA}(i, S, \delta)$, we first find the block B_{S_j} of B_S which contains i-th 0 by answering $\mathsf{search}(i)$ query on D, and then return $\mathsf{sum}(j-1)$ query on E. To

[4] Proofs of the results marked ($\star$) are omitted due to space limitation and appear in the full version [1].

answer dselectA(i, S, δ), we first find the block B_{S_j} of B_S which contains an answer of the dselectA$_1(i, B_S, \delta)$ query, and then return sum$(j-1)$ as the answer for dselectA(i, S, δ). Note that if $j = 1$, we return 0 for both queries. The total running time is $SPS(n'/\delta, n')$ for both rankA and dselectA queries on S, by Lemma 3 and Theorem 8. 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 first show that at least $\lfloor n/\lceil \delta/\ell \rceil \rfloor \lg(\max(\lfloor \ell/\delta \rfloor, 1)+1)$ bits are are necessary for supporting drankA queries on S.

▶ **Theorem 9** ($\star$). *Given a multiset S where each element is from the universe $U = \{1, 2, \ldots, n\}$ of size n, 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, where ℓ is a bound on the maximum frequency of each element in S.*

We describe a data structure which answers drankA and selectA queries on S in $O(1)$ time. For drankA queries, it uses the optimal space. The details are omitted due to space limitation.

2.3 Approximate ss and iss queries on binary streams

In this section, we consider a data structure for answering ssA and issA queries on a binary stream. We first show how to modify the data structure of the Lemma 1, for answering ss(i, n) and iss(i, n) queries in constant time using $n + o(n)$ bits, while supporting updates in constant time. We break the stream into *frames*, which is n-bit consecutive elements in the stream. Since one can construct a data structure of Lemma 1 in online [5], it is easy to show that we can answer ss and iss queries in constant time using $2n + o(n)$ bits while supporting constant-time updates by maintaining two data structure of Lemma 1 such as one for the current frame and other for the previous frame of the stream. To make this data structure using $n + o(n)$ bits, we construct a data structure of Lemma 1 on the new frame while replacing the oldest part of the data structure constructed on the previous frame. The details of the succinct data structure are omitted due to space limitation.

Next, we consider a data structure for answering ssA(i, n, δ) and issA(i, n, δ) queries on the binary stream in constant time using $\lceil n/\delta \rceil + O(\lg n) + o(n/\delta)$ bits. We first split each frame $f = f_1 \ldots f_n$ into $\lceil n/\delta \rceil$ chunks $g_1 \ldots g_{\lceil n/\delta \rceil}$ such that for $1 \le i \le \lceil n/\delta \rceil$, $g_i = 1$ if and only if $f_{(i-1)\delta+1} \cdots f_{\min(n, i\delta)}$ contains $j\delta$-th 1 in f for any integer $j \le i$. Now consider a (virtual) binary stream of g_i's. Then we can construct an $\lceil n/\delta \rceil + o(n/\delta)$-bit data structure for answering ss(i, n), iss(i, n) queries in constant time while supporting constant-time updates on the such stream (In the rest of this section, all of ss and iss queries are answered on the virtual stream). We also maintain c and tc, which stores the number of 1's in the current frame and chunk of the stream respectively. Finally, we maintain an value t which is an index of the last-arrived element in the current frame. All these additional values can be stored using $O(\lg n)$ bits.

When f_t is arrived, We first increase c and tc by 1 if $f_t = 1$. If $(t \mod \delta) = 0$ or $t = n$, we send 1 to the virtual stream if there is an integer $j \le t$ such that $c - tc \le j\delta \le c$, and send 0 to the virtual stream otherwise. After that, we update the data structure which supports ss and iss queries on the virtual stream, and reset tc to zero (if $t = n$, we also reset c to zero). Since we can update the data structure on the virtual stream in constant time, the above procedure can be done in constant time. Now we describe how to answer ssA and issA queries.

- ssA queries: To answer the $\mathsf{ssA}(i, S, \delta)$ query, we return 0 if $i \leq \delta$. If not, let f'_i be the $(\lceil (i - (t \mod \delta))/\delta \rceil)$-th last element in the virtual stream, Then we return $tc + \delta \mathsf{ss}(\lfloor (i - (t \mod \delta))/\delta \rfloor, \lceil n/\delta \rceil) + (i - (t \mod \delta) \mod \delta) f'_i$, which gives an answer of the $\mathsf{ssA}(i, n, \delta)$ query by the same argument as the proof of Theorem 5.
- issA queries: To answer the $\mathsf{issA}(i, n, \delta)$ query, we return $n - (t - (t \mod \delta))$ if $i \leq tc$. Otherwise, we return $n - (\delta(\mathsf{iss}(\lfloor (i - tc)/\delta \rfloor, \lceil n/\delta \rceil) + ((i - tc) \mod \delta))$ by the same argument as the proof of Theorem 5.

Since ss and iss queries on the virtual stream take $O(1)$ time, we can answer both ssA and issA queries on the stream in $O(1)$ time. Thus we obtain the following theorem.

▶ **Theorem 10.** *For a binary stream, there exists a data structure that uses $\lceil n/\delta \rceil + O(\lg n) + o(n/\delta)$ bits and supports ssA and issA queries on the stream with additive error δ, in constant time. Also, the structure supports updates in constant time.*

Comparing to the lower bound of Theorem 4 for answering drankA and $\mathsf{selectA}$ queries on bit-strings (this also gives a lower bound for answering ssA and issA queries), the above data structure takes $\Omega(n/\delta)$ bits when $n/\delta = o(\lg n)$. However in the sliding window of size n, at least $\lfloor \lg n \rfloor$ bits are necessary [3] for answering ssA queries even the case when i is fixed to n. Therefore the data structure of Theorem 10 supports ssA and issA queries with optimal space when $n/\delta = \omega(\lg n)$, and asymptotically optimal otherwise.

3 Queries on strings over 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 suffix sums 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. One can easily show that at least $\lfloor n/\delta \rfloor \lg \sigma$ bits are necessary to support drankA and $\mathsf{selectA}$ queries on A, by extending the proof of Theorem 4 to strings over larger alphabets. In this section, we describe a data structure that supports drankA and $\mathsf{selectA}$ queries on A in $O(\lg \lg \sigma)$ time, using twice the optimal space. We make use of the following result from [9] for supporting rank and select queries on strings over large alphabets. We now use the following lemma to prove our main result for the section.

▶ **Lemma 11** ([9]). *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 $1 \leq j \leq \sigma$.*

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 12** ($\star$). *Let A be a string of length n over the alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$. Then for any $1 \leq j \leq \sigma$, one can support drankA_j and $\mathsf{selectA}_j$ queries in $O(\lg \lg \sigma)$ time using $2n/\delta \lg(\sigma + 1) + o((n/\delta) \lg(\sigma + 1))$ bits.*

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 10 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 9 when $\delta = o(\ell \cdot n/\lg n)$.

We note that this model was studied in [3, 6, 8] for window-sum queries. That is, our work generalizes this model to allow the user to specify the window size $i \leq n$ 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. [8, 6] requires $O(\epsilon^{-1} \lg (\ell n \varepsilon))$ time to compute a $(1 + \varepsilon)$ approximation for the sum of the last n elements while [3] 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 $i \leq n$ 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 [3]. 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 current approaches cannot work for a large ℓ value. If we use sub-blocks of size $\Theta(\lg n)$ as in [5, 12], 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, $\ell \geq n$) then this becomes $\Omega(n\ell) = \omega(n \lg (\ell + 1))$, which is not even asymptotically optimal.

▶ **Theorem 13** ($\star$). *There exists a data structure that requires $n \lg (\ell + 1) (1 + o(1))$ bits and support constant time (exact) suffix sums queries and updates.*

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. 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 9.

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 $Round_{\mathfrak{b}}(x) \triangleq 2^{-\mathfrak{b}} \ell \cdot \left\lfloor \frac{x 2^{\mathfrak{b}}}{\ell} \right\rfloor$ so it will require $\mathfrak{b} \triangleq \lceil \lg (n/\mu) + \lg \lg n \rceil$ bits. 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 both consider a smaller chunks; namely, we use chunks of size $\nu \triangleq \max \{\lfloor \mu \cdot (1 - 1/\lg n) \rfloor, 1\}$. We also consider $\widetilde{\delta} \triangleq \lfloor \delta \cdot (1 - 1/\lg n) \rfloor$ that is slightly lower than δ to compensate for the rounding error when $\mu \leq 1$. [5] We then employ the exact suffix sums construction from the

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

Algorithm 1 Algorithm for ssA.

```
Initialization: $\mathfrak{r} \leftarrow 0, o \leftarrow 0, \mathbb{A}$.init()
function ADD(ELEMENT $x$)
    $o \leftarrow (o+1) \mod \nu$
    $\mathfrak{r} \leftarrow \mathfrak{r} + Round_{\mathfrak{b}}(x)$
    if $o = 0$ then                                        ▷ End of a chunk
        $\rho \leftarrow \left\lfloor \widetilde{\delta}^{-1} \cdot \mathfrak{r} \right\rfloor$
        $\mathfrak{r} \leftarrow \mathfrak{r} - \widetilde{\delta} \cdot \rho$
        $\mathbb{A}$.ADD($\rho$)
function QUERY($i$)
    if $i \le o$ then                                      ▷ Queried within the current chunk
        return $\mathfrak{r} - \left(\widetilde{\delta} - 1/2\right)$
    else
        $numElems \leftarrow \left\lceil \frac{i-o}{\nu} \right\rceil$
        $totalSum \leftarrow \mathbb{A}.\text{QUERY}\,(numElems)$
        $oldest_\rho \leftarrow totalSum - \mathbb{A}.\text{QUERY}\,(numElems - 1)$
        $out \leftarrow (\nu - ((i-o) \mod \nu))$
        return $\mathfrak{r} - \left(\widetilde{\delta} - 1/2\right) + \widetilde{\delta} \cdot totalSum - \ell \cdot oldest_\rho \cdot out$
```

previous section for window size of $s \triangleq \lceil n/\nu + 1 \rceil$ (the number of chunks that can overlap with the window) over a stream of integers in $\{1, \dots, z\}$, where $z \triangleq \lfloor \mu^{-1}\nu \rfloor$ is a bound on the resulting items. We use ρ to denote the input that respresents the current block.

The query procedure is also a bit tricky. 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. 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. Our algorithm uses the following variables:

- $\mathbb{A}$ - 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 $\{1, \dots, z\}$.
- $\mathfrak{r}$ - tracks the sum of elements that is not yet recorded in $\mathbb{A}$.
- o - the offset within the chunk.

A pseudo code of our method appears in Algorithm 1. Next follows a memory analysis of the algorithm.

▶ **Lemma 14** ($\star$). *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.*

Thus, we conclude that our algorithm is succinct if the error satisfies $\delta = o(\ell \cdot n/\lg n)$. We note that a $\lfloor \lg n \rfloor$ bits lower bound for BASIC-SUMMING with an additive error was shown in [3], even when only fixed sized windows (where $i \equiv n$) 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 9.

▶ **Corollary 15.** *Let* $\ell, n, \delta \in \mathbb{N}^+$ *such that* $\mu \triangleq \delta/\ell$ *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 16** ($\star$). *Algorithm 1 solves* ssA *while processing elements and answering queries in constant time.*

References

1 R. Ben Basat, S. Jo, S. Rao Satti, and S. Ugare. Approximate Query Processing over Static Sets and Sliding Windows. *ArXiv e-prints*, September 2018. arXiv:1809.05419.

2 Ran Ben-Basat, Gil Einziger, and Roy Friedman. Give Me Some Slack: Efficient Network Measurements. In *MFCS*, pages 34:1–34:16, 2018.

3 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient Summing over Sliding Windows. In *SWAT*, pages 11:1–11:14, 2016. doi:10.4230/LIPIcs.SWAT.2016.11.

4 Ran Ben-Basat, Roy Friedman, and Rana Shahout. Heavy Hitters over Interval Queries. *CoRR*, abs/1804.10740, 2018. arXiv:1804.10740.

5 David R. Clark and J. Ian Munro. Efficient Suffix Trees on Secondary Storage. In *SODA*, pages 383–391, 1996.

6 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

7 Hicham El-Zein, J. Ian Munro, and Yakov Nekrich. Succinct Color Searching in One Dimension. In *ISAAC*, pages 30:1–30:11, 2017. doi:10.4230/LIPIcs.ISAAC.2017.30.

8 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002. doi:10.1145/564870.564880.

9 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/Select Operations on Large Alphabets: A Tool for Text Indexing. In *SODA*, pages 368–373, 2006.

10 Alexander Golynski, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. Optimal Indexes for Sparse Bit Vectors. *Algorithmica*, 69(4):906–924, 2014. doi:10.1007/s00453-013-9767-2.

11 Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Succinct data structures for Searchable Partial Sums with optimal worst-case performance. *Theor. Comput. Sci.*, 412(39):5176–5186, 2011. doi:10.1016/j.tcs.2011.05.023.

12 Guy Joseph Jacobson. *Succinct Static Data Structures.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1988. AAI8918056.

13 Seungbum Jo, Stelios Joannou, Daisuke Okanohara, Rajeev Raman, and Srinivasa Rao Satti. Compressed Bit vectors Based on Variable-to-Fixed Encodings. *Comput. J.*, 60(5):761–775, 2017.

14 P. B. Miltersen. Cell probe complexity - a survey. *FSTTCS*, 1999.

15 J.Ian Munro, Venkatesh Raman, and S.Srinivasa Rao. Space Efficient Suffix Trees. *J. Algorithms*, 39(2):205–222, 2001.

16 Gonzalo Navarro and Eliana Providel. Fast, Small, Simple Rank/Select on Bitmaps. In *SEA*, pages 295–306, 2012. doi:10.1007/978-3-642-30850-5_26.

17 Daisuke Okanohara and Kunihiko Sadakane. Practical Entropy-Compressed Rank/Select Dictionary. In *ALENEX*, pages 60–70, 2007. doi:10.1137/1.9781611972870.6.

18 Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct Dynamic Data Structures. In *WADS*, pages 426–437, 2001. doi:10.1007/3-540-44634-6_39.

19 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007. doi:10.1145/1290672.1290680.