Optimal Succinct Rank Data Structure via Approximate Nonnegative Tensor Decomposition

Huacheng Yu*

Abstract

Given an n-bit array A, the succinct rank data structure problem asks to construct a data structure using space $n + r$ bits for $r \ll n$, supporting rank queries of form $\texttt{rank}(u) = \sum_{i=0}^{u-1} A[i]$. In this paper, we design a new succinct rank data structure with $r = n/(\log n)^{\Omega(t)} + n^{1-c}$ and query time $O(t)$ for some constant $c > 0$, improving the previous best-known by Pătraşcu [Pät08], which has $r = n/(\frac{\log n}{t})^{\Omega(t)} + \tilde{O}(n^{3/4})$ bits of redundancy. For $r > n^{1-c}$, our space-time tradeoff matches the cell-probe lower bound by Pătraşcu and Viola [PV10], which asserts that r must be at least $n/(\log n)^{O(t)}$. Moreover, one can avoid an n^{1-c}-bit lookup table when the data structure is implemented in the cell-probe model, achieving $r = \lceil n/(\log n)^{\Omega(t)} \rceil$. It matches the lower bound for the full range of parameters.

En route to our new data structure design, we establish an interesting connection between succinct data structures and approximate nonnegative tensor decomposition. Our connection shows that for specific problems, to construct a space-efficient data structure, it suffices to approximate a particular tensor by a sum of (few) nonnegative rank-1 tensors. For the rank problem, we explicitly construct such an approximation, which yields an explicit construction of the data structure.

*Harvard University. yuhch123@gmail.com. Supported in part by ONR grant N00014-15-1-2388, a Simons Investigator Award and NSF Award CCF 1715187.

1 Introduction

Given an array $A[0..n-1]$ of bits, the partial sums problem (a.k.a, the rank problem) asks to preprocess A into a data structure using as little space as possible, supporting queries of form $\texttt{rank}(u) = \sum_{i=0}^{u-1} A[i]$ efficiently. One trivial solution is to explicitly write down all prefix sums, which uses n *words* of space and constant query time. In succinct data structures, one seeks data structures using space close to the information theoretical limit, n *bits* for partial sums, with an efficient query time.

Succinct rank data structures are central building blocks in many succinct data structure problems with a rich history [Jac89, CM96, Mun96, Cla97, MRR98, RRR02, GMR06, Gol07b, Păt08]. Jacobson [Jac89], Clark and Munro [CM96] gave the first succinct rank data structures using $n + o(n)$ space with constant query time. After a series of improvements [Mun96, MRR98, RRR02], Golynski et al. [GGG$^+$07] achieved space $n + O(\frac{n \log\log n}{\log^2 n})$ for constant query time. Later, the seminal paper "Succincter" by Pătraşcu [Păt08] proposed a data structure using space $n + n/(\frac{\log n}{t})^t + \tilde{O}(n^{3/4})$ and query time $O(t)$, showing that the redundant bits can be any $n/\mathrm{poly}\log n$ when query time is constant.

Lower bounds for this problem also have received attention from researchers in the area [Mil05, GM07, Gol07a, PV10]. Most lower bounds are for "systematic encodings." In systematic encoding, we are given an input that is stored explicitly in the raw form, and we may then build a (sublinear) auxiliary data structure, which will be stored on the side. The query algorithm has access to both the raw input and the auxiliary data structure. Golyski [Gol07a] showed a space lower bound of $n + (n \log t)/t$ for query time t, for any systematic encoding of the rank problem. For general data structures, Pătraşcu and Viola [PV10] proved a space lower bound of $n + n/w^{O(t)}$ for query time t in the cell-probe model with word-size w (implying the same RAM lower bound). In the standard regime where $w = \Theta(\log n)$, this space lower bound matches the "Succincter" upper bound for constant query times.

However, if one insists on, say $c_0 \log n/\log\log n$ query time, for sufficiently large constant c_0, then the best-known data structure occupies at least $n + n^{1-o(1)}$ bits of space on a worst-case n-bit input, whereas the state-of-the-art lower bound does not even rule out an $(n+1)$-bit data structure! Closing this gap is referred to as "a difficult problem" in [PV10].

Interestingly, we show such an $(n+1)$-bit data structure *does exist*, if we allow arbitrary $O(w)$-bit word operations.

Theorem 1 (informal)**.** *Given an n-bit array, one can construct a data structure using*

$$n + \lceil \frac{n}{w^{\Omega(t)}} \rceil$$

bits of memory supporting rank queries in $O(t)$ time, where $w \geq \Omega(\log n)$ is the word-size, assuming the data structure can perform arbitrary $O(w)$-bit word operations.

By applying a standard trick for self-reducible problems and storing lookup tables for the necessary $O(w)$-bit word operations, this data structure can also be implemented in word RAM with an n^{1-c} space overhead.

Theorem 2 (informal)**.** *Given an n-bit array, one can construct a data structure using*

$$n + \frac{n}{(\log n)^{\Omega(t)}} + n^{1-c}$$

bits of memory supporting rank queries in $O(t)$ time, in a word RAM with word-size $\Theta(\log n)$, for some universal constant $c > 0$.

In particular, for query time $t = c_0 \log n / \log\log n$ for sufficiently large c_0, Theorem 1 gives us a data structure using only $n + 1$ bits of memory. Moreover, our new cell-probe data structure matches the Pătraşcu-Viola lower bound for any bits of redundancy, up to a constant factor in query time. It settles the space-time tradeoff for succinct rank in the cell-probe model. One may also observe if only exactly n bits of memory are allowed, then there is nothing one can do beyond storing A explicitly. It is because in this case, the data structure has to be a bijection between A and the memory contents. In particular, even to verify whether $\texttt{rank}(n) = 0$, one has to check if the memory content corresponds to the exact all-zero input, which requires a linear scan. However, when $n+1$ bits are allowed, half of the 2^{n+1} memory configurations may be unused, which could potentially facilitate the query algorithm.

En route to our new succinct rank data structure construction, we establish an interesting connection between succinct data structures and *approximate nonnegative tensor decomposition*. Tensors are generalizations of matrices. An order-B tensor can be viewed as a B-dimensional array. Analogous to the rank of a matrix, a tensor $\mathbf{T}$ has rank 1 if its entries can be written as

$$\mathbf{T}_{x_1,\ldots,x_B} = a^{(1)}_{x_1} \cdot a^{(2)}_{x_2} \cdots a^{(B)}_{x_B},$$

for vectors $a^{(1)}, \ldots, a^{(B)}$. The rank of a tensor $\mathbf{T}$ is the minimum number of rank-1 tensors, of which $\mathbf{T}$ can be expressed as a sum. The nonnegative rank of a nonnegative tensor further requires each rank-1 tensor (or equivalently, each $a^{(i)}$) to be nonnegative, hence is at least as large as the rank. Given a nonnegative tensor $\mathbf{T}$ and parameters r, ϵ, the problem of *approximate nonnegative tensor decomposition* asks to find r nonnegative rank-1 tensors $\mathbf{T}_1, \ldots, \mathbf{T}_r$ such that $\|\mathbf{T} - (\mathbf{T}_1 + \cdots + \mathbf{T}_r)\| \le \epsilon$ under certain norm, if exists.

Connections to tensor decomposition. As we mentioned in the beginning, explicitly storing all prefix sums takes too much space. One inherent reason is that the prefix sums are very correlated. Denote by T_i the number of ones in the first i bits. One may verify that for uniformly random inputs (which maximizes the input entropy) and $i < j$, $I(T_i; T_j) \approx \frac{1}{2} \log \frac{j}{j-i}$. Even just storing T_{100} and T_{110} in separate memory words would already introduce a redundancy of $\frac{1}{2} \log 11 > 1.5$ bits, because the "same 1.5 bits of information" is stored in two different locations. Hence, in order to achieve low redundancy, only mutually (almost) independent variables could be stored separately.

The key observation used in our new data structure is the following. Suppose we were to store B (correlated) numbers $y_1, \ldots, y_B \in [n]$, if we could find another variable η such that conditioned on η, $y_1, \ldots, y_B$ become (almost) mutually independent, then one could hope to first store η, then store these B numbers *conditioned on* η. To retrieve one y_i, one always first reads η, which reveals the representation of y_i, then reads y_i. This strategy is only possible if the support size of η is not too large, and can be encoded using few bits, since its value needs to be retrieved prior to reading any y_i. If we are aiming at constant retrieval time, then the support size of η must be at most $2^{O(w)}$.

The joint distribution of $(y_1, \ldots, y_B) \in [n]^B$ can be described by an order-B tensor $\mathbf{T}$ of size n^B, where the entries describe the probability masses. Any nonnegative rank-1 tensor of this size would correspond to an independent distribution. Suppose we could find r nonnegative rank-1 tensors $\mathbf{T}_1, \ldots, \mathbf{T}_r$ such that

$$\|\mathbf{T} - (\mathbf{T}_1 + \cdots + \mathbf{T}_r)\|_1 \le \epsilon.$$

This would imply that $\mathbf{T}$ can approximately be viewed as a convex combination of r independent distributions. Let η indicate which independent distribution we are sampling from, then $y_1, \ldots, y_B$ become conditionally independent conditioned on η (except for a small probability of ϵ). More importantly, the support size of η is equal to r. If such decomposition is possible for $r = 2^{O(w)}$ and $\epsilon = 1/\mathrm{poly}\, n$, then we will have hope to store $(y_1, \ldots, y_B)$ with constant retrieval time and (negligible) redundancy of $1/\mathrm{poly}\, n$.

Computing tensor decomposition. In general, nonnegative tensor decomposition is computationally difficult. Even for tensor order $B = 2$ (i.e., nonnegative *matrix* factorization), any algorithm with running time subexponential in r would yield a subexponential time algorithm for 3-SAT [AGKM16, Moi16], breaking Exponential Time Hypothesis. Fortunately, the tensors that we obtain from the data structure problem are not arbitrary. For the rank problem, the values $\mathbf{T}_{x_1,\ldots,x_B}$ are relatively smooth as a function of $(x_1, \ldots, x_B)$. Given such a tensor, we may partition it into small cubes, and approximate the values within each cube by a low degree polynomial $P(x_1, \ldots, x_B)$. The key observation here is that the tensor corresponding to a monomial $x_1^{e_1} \cdots x_B^{e_B}$ has rank 1. If P has low degree, thus has a small number of monomials, then the tensor restricted to the cube must have low approximate rank. To make this approximation nonnegative, we apply the following transformation to a negative monomial $-x^a y^b$ (plus a large constant):

$$M^{a+b} - x^a y^b \equiv \frac{1}{2}(M^a - x^a)(M^b + y^b) + \frac{1}{2}(M^a + x^a)(M^b - y^b). \tag{1}$$

When $x, y \in [0, M]$, both terms on the RHS become nonnegative. One may also generalize this equation to monomials with more than two variables. The polynomial obtained from each small cube has a sufficiently large constant term so that all negative monomials can be transformed simultaneously via the above equation. Finally, by summing up the approximations within each cube, we obtain a low rank nonnegative approximation for the whole tensor.

1.1 Organization of the paper

In Section 2, we give preliminary and define notations. In Section 3, we give an overview of the new succinct rank data structure, as well as a summary of [Păt08]. In Section 4, we present our data structure construction. Finally, we conclude with discussions and open questions in Section 5.

2 Preliminary and Notations

2.1 Notations

Let $a, b \in \mathbb{R}$ and $b \geq 0$, denote by $a \pm b$ the set $[a - b, a + b]$. Similarly, denote by $c(a \pm b)$ the set $[c(a - b), c(a + b)]$. Throughout the paper, $\log x = \log_2 x$ is the binary logarithm.

2.2 Spillover representation

One important technique in succinct data structures is the *spillover representation*, formally introduced by Pătraşcu [Păt08]. Similar ideas also appeared in Munro et al. [MRRR03] and Golyski et al. [GGG+07]. It allows one to use "fractional bits" of memory in each component of the data structure construction so that the fractions can be added up before rounding.

A data structure is said to use m bits of memory with spillover size K, if it can be represented by a pair consisting of m bits and a number (called the *spillover*) in $[K]$ (usually $K \leq 2^{O(w)}$). We may also say that the data structure uses space $[K] \times \{0, 1\}^m$. At the query time, we assume that the spillover $k \in [K]$ is given for free, and *any* consecutive w bits of the representation can be accessed in constant time. These m bits are usually stored explicitly in the memory. Hence, any consecutive w bits can be retrieved by reading one or two consecutive words.

Intuitively, such representations use "$m + \log K$ bits" of space. It avoids the problem of "rounding to the next integer bit" in designing a subroutine, as now we can round up the spillover, which wastes much

less space: $\log(K+1)$ and $\log K$ only differ by $\log(1+1/K) \approx 1/K$ bit. Setting $K = \Omega(n^2)$ makes it negligible.

2.3 The cell-probe model

The cell-probe model proposed by Yao [Yao78] is a powerful non-uniform computational model, primarily used in data structure lower bound proofs. In the cell-probe model, we only measure the number of memory accesses. The memory is divided into *cells* of w bits. The data structure may read or write the content of a memory cell by *probing* this cell. In each step of the query algorithm, it may probe one memory cell (or in the last step, it returns the answer to the query), based on all information it has obtained so far, including the query and all previous contents the algorithm has seen. The running time is defined to be the number of memory cells probed. We assume computation is free.

3 A Streamlined Overview

In this section, we overview our new succinct rank data structure. To avoid heavy technical details, we are going to present the data structure in a stronger model, where each word may store "$O(w)$ bits of information about the data", rather than an exact w-bit string. We refer to this model of computation as the *information cell-probe model*. The data structure for standard word RAM can be found in Section 4.

Let us first assume that the input n-bit array A is uniformly random.[1] The information that each memory word (content) C_i reveals about the input $I(C_i; A)$ is measured under this input distribution. This mutual information, in some sense, measures "the space usage of word C_i." We will focus on the case where $w = \Theta(\log n)$, and the goal is to design a data structure that

- stores no more than w bits of information in every memory cell C_i: $I(C_i; A) \le w$,
- uses no more than "$n+1$ bits" of space: $\sum_i I(C_i; A) \le n+1$, and
- supports rank queries $\texttt{rank}(u)$ in $O(\log_w n) = O(\log n / \log\log n)$ time.

The top-level of the data structure is similar to [Păt08], which is a standard range tree with branching factor B, here for $B = w^{1/3}$. For simplicity, we assume n is a power of B (and $n = B^t$). Given an input array $A[0, \ldots, n-1]$, we construct a tree with branching factor B and depth t by recursion. Each node v at level i in the tree is associated with a subarray A_v of length B^{t-i}. For $j \in [B]$, the j-th child of v is associated with the subarray containing $((j-1)B^{t-i-1}+1)$-th to (jB^{t-i-1})-th element of A_v. Figure 1 presents the top-level structure (a standard range tree).

Suppose both **retrieve_prefix_sum** and **retrieve** take constant time, then rank queries can be answered in $O(t) = O(\log_w n)$ time. Hence, the task boils down to implementing **aggregate_sums** efficiently, which constitutes our main technical contribution.

Let $T_i = s_1 + \cdots + s_i$. There are two natural implementations.

[1] Uniform distribution maximizes the input entropy, making the same space bound more difficult to achieve (at least in expectation). The data structure in Section 4 for standard word RAM does not rely on this assumption, and the space bound there is for worst-case input.

Constructing the data structure:

initialization(A)

1. store the sum in a word $s := \sum_{i=0}^{n-1} A[i]$
2. store $\texttt{DS(A)} :=$ **preprocess**(A, s)

preprocess(A, s) // preprocess A, provided that its sum $s = \sum_{i=0}^{n-1} A[i]$ has already been stored

1. if $|A| = 1$, return `NULL` // nothing else to store
2. divide A evenly into $A_1, \ldots, A_B$, let s_i be the sum of A_i
3. $\texttt{sums(A)} :=$ **aggregate_sums**($A, s_1, s_2, \ldots, s_B$) // preprocess the sums
4. for each $i = 1, \ldots, B$, $\texttt{DS}(\texttt{A}_\texttt{i}) :=$ **preprocess**(A_i, s_i) // recurse on each subarray
5. return $(\texttt{sums(A)}, \texttt{DS}(\texttt{A}_1), \ldots, \texttt{DS}(\texttt{A}_\texttt{B}))$

aggregate_sums($A, s_1, s_2, \ldots, s_B$): preprocess B numbers, such that in constant time, one may retrieve s_i using **retrieve**(s, i), and retrieve $s_1 + \cdots + s_i$ using **retrieve_prefix_sum**(s, i), provided that $s = s_1 + \cdots + s_B$. (implemented in Section 3.2)

Query algorithm:

rank(u)

1. read s
2. $\texttt{DS(A)}$.**query**(s, u)

$\texttt{DS(A)}$.**query**(s, u) // output $A[0] + \cdots + A[u-1]$ provided that the sum of A is s

1. if $u = 0$, return 0
2. if $u = |A|$, return s // must have returned by now, if $|A| = 1$
3. compute (i, u') for integer i and $0 \le u' < |A|/B$ such that $u = i(|A|/B) + u'$ // u is in the $(i+1)$-th block
4. let $s_{i+1} = \texttt{sums(A)}$.**retrieve**$(s, i+1)$
5. return $\texttt{sums(A)}$.**retrieve_prefix_sum**(s, i) // $s_1 + \cdots + s_i$
   $+\texttt{DS}(\texttt{A}_{\texttt{i+1}})$.**query**$(s_{i+1}, u')$ // recurse on A_{i+1}

Figure 1: Succinct rank data structure

- We may store the prefix sums $T_1, T_2, T_3, \ldots$, which allows one to retrieve prefix sum and each s_i in constant time. However, under uniform input distribution, we have

$$I(T_{k-1}; T_k) \approx \frac{1}{2} \log k.$$

  "The same $\frac{1}{2} \log k$ bits of information" will be stored in different locations. Since the goal is to use no more than one extra bit, we fail as soon as the first 5 prefix sums are written down.

- Or we may store the numbers $s_1, s_2, s_3, \ldots$ The same issue may still exist. Moreover, storing the numbers explicitly would not let us retrieve prefix sums efficiently.

It is worth noting that if we set $B = 2$ instead of $w^{1/3}$, one could jointly store the pair (s_1, s_2) *conditioned on* $s_1 + s_2$ in one memory word, which introduces no redundancy and allows one to retrieve both sums in constant time. However, the depth of the tree becomes $\log n$ instead of $\log_w n$, so does the query time. It is essentially what the previous data structure [Păt08] does, after "projected" into the information cell-probe model. The major effort of [Păt08] is spent on transforming this "standard" solution in information cell-probe model to word RAM, which we will briefly discuss in Section 3.3. Hence, the subroutine **aggregate_sums** is where our solution deviates from the previous one.

3.1 Aggregating sums

To implement these subroutines, the main idea is to store correlated variables $\{T_i\}$ as we discussed in the introduction: Find a random variable η such that conditioned on η (and the sum of all numbers $T_B = s$), the prefix sums $\{T_i\}$ are approximately independent; we will first store η, then store the prefix sums *conditioned on* η. To retrieve a T_i, we first read η, then read T_i, which is stored according to conditional distribution. Each number s_i can be retrieved by simply taking the difference of T_i and T_{i-1}.

To find such η, we analyze the joint distribution of $(T_1, \ldots, T_B)$, which can be described by an order-B tensor $\mathbf{T}$ of size $(n+1)^B$. Each entry $(x_1, \ldots, x_B)$ describes the probability that $s_1 = x_1, s_2 = x_2, \ldots, s_B = x_B$. As we mentioned in the introduction, finding such a random variable η is (approximately) equivalent to decomposing $\mathbf{T}$ into a sum of few nonnegative rank-1 tensors, since a nonnegative rank-1 tensor describes an independent distribution. Then the joint distribution can be viewed as a convex combination of these independent distributions, where η decides which independent distribution to sample from. The number of such rank-1 tensors corresponds to the support size of η, hence provides an upper bound on its entropy. Since η needs to be stored in one word, the goal is to decompose $\mathbf{T}$ into $2^w = n^{O(1)}$ rank-1 nonnegative tensors. In the next subsection, we elaborate this idea, and show how to find this η for our problem.

3.2 Tensor decomposition

The tensor corresponding to the joint distribution of $(T_1, \ldots, T_B)$ can be written in an explicit form as follows. Suppose each subarray $A_1, \ldots, A_B$ has size l, we have

$$\mathbf{T}_{x_1,\ldots,x_B} := \Pr[T_1 = x_1 \wedge T_2 = x_2 \wedge \cdots \wedge T_B = x_B] = 2^{-lB} \cdot \prod_{i=1}^{B} \binom{l}{x_i - x_{i-1}}, \tag{2}$$

where we assumed $x_0 = 0$.

We first show that the binomial coefficients $\binom{l}{x_i - x_{i-1}}$ appeared in Equation (2) can be piecewise-approximated by a low degree polynomial in x_i and x_{i-1} (multiplied by an exponential function) (see Lemma 9 in Section 4.2). After handling the negative terms via Equation (1) and putting together the piecewise approximation, Lemma 10 in Section 4.2 implies that the binomial coefficients can be expressed as[2]

$$\binom{l}{x_i - x_{i-1}} \cdot 2^{-l} = E(x_i, x_{i-1}) + \sum_{j=1}^{r_0} Q_j(x_{i-1}) \cdot R_j(x_i),$$

where E, Q_j, R_j are all nonnegative, $\mathbb{E}_{x_{i-1}}[\sum_{x_i} E(x_i, x_{i-1})] \le \epsilon$ (the error term is small) and $r_0 = (\log 1/\epsilon)^{O(1)}$ (the number of terms is small). By multiplying the above approximation over all $i = 1, \ldots, B$, we obtain an approximation for $\mathbf{T}$:

$$\mathbf{T}_{x_1,\ldots,x_B} = \tilde{E}(x_1, \ldots, x_B) + \sum_{j_1,\ldots,j_B=1}^{r_0} Q_{j_1}(0) R_{j_1}(x_1) Q_{j_2}(x_1) R_{j_2}(x_2) \cdots Q_{j_B}(x_{B-1}) R_{j_B}(x_B).$$

One may verify that the overall error term $\tilde{E}$ is small, $\sum_{x_1,\ldots,x_B} \tilde{E}(x_1, \ldots, x_B) \le B\epsilon$, and the total number of terms r is at most $r_0^B = (\log 1/\epsilon)^{O(B)}$. Note that each term $Q_{j_1}(0) R_{j_1}(x_1) Q_{j_2}(x_1) R_{j_2}(x_2) \cdots Q_{j_B}(x_{B-1}) R_{j_B}(x_B)$

[2]The lemma provides a few extra guarantees on the approximation that is needed for the word RAM data structure, and is omitted here.

corresponds to a rank-1 tensor (i.e., an independent distribution). By normalizing each rank-1 tensor, we obtain the following lemma on nonnegative approximate tensor decomposition of $\mathbf{T}$.

Lemma 1. *One can find $r \leq (\log 1/\epsilon)^{O(B)}$ rank-1 tensors $\mathbf{T}_1, \ldots, \mathbf{T}_r$ such that the above tensor $\mathbf{T}$ can be expressed as*

$$\mathbf{T} = p_E \cdot \mathbf{E} + \sum_{j=1}^{r} p_j \cdot \mathbf{T}_j,$$

where $\mathbf{E}$, p_E, $\mathbf{T}_j$ and p_j are all nonnegative, $\|\mathbf{E}\|_1 = 1, \|\mathbf{T}_j\|_1 = 1$ and $p_E \leq B\epsilon$ for all $j = 1, \ldots, r$.

Note that for technical reasons, the final data structure for word RAM requires extra guarantees on the decomposition, and the above lemma is not directly used in Section 4. Hence, we only state it here without a formal proof.

By setting $\epsilon = 1/n^2$, we have $r = 2^{O((\log n)^{1/3} \log\log n)} = n^{o(1)}$. Thus, one can view the joint distribution $\mathbf{T}$ of $(T_1, \ldots, T_B)$ as a convex combination of $\mathbf{E}$, whose probability is tiny, and r mutually independent distributions $\mathbf{T}_1, \ldots, \mathbf{T}_r$.[3] Now let η be the random variable indicating the distribution we are currently sampling from, we have $H(\eta) \leq o(\log n)$, and the prefix sums are almost independent conditioned on η.

To generate η given input array A, we first partition the sample space $\{0,1\}^{lB}$, such that each part corresponds to one distribution in the convex combination. More specifically, we fix a partition of $\{0,1\}^{lB}$, the domain of A, into $\mathcal{K}_E, \mathcal{K}_1, \ldots, \mathcal{K}_r$, such that $|\mathcal{K}_E| \approx p_E \cdot 2^{lB}$ and $|\mathcal{K}_j| \approx p_j \cdot 2^{lB}$ for $j = 1, \ldots, r$. Moreover, this partition guarantees that for every $j = 1, \ldots, r$, the prefix sums $(T_1, \ldots, T_B)$ of an array A that is sampled uniformly from $\mathcal{K}_j$, is approximately distributed according to $\mathbf{T}_j$. Also, the distribution of $(T_1, \ldots, T_B)$ when A is sampled uniformly from $\mathcal{K}_E$ is roughly $\mathbf{E}$. Given such a partition, it suffices to set η to the part that contains input A. See Figure 2 for the detailed implementations of **aggregate_sums**, **retrieve** and **retrieve_prefix_sum**.

Intuitively, storing a random variable μ takes $H(\mu)$ bits of space, and storing μ conditioned ν should take $H(\mu \mid \nu)$ bits of space. We will briefly discuss how to store a variable conditioned on another in the next subsection, and the details are deferred to the final construction in Section 4.

One may verify that η generated by this algorithm satisfies

$$\sum_{i=1}^{B} H(T_i \mid \eta) \leq H(T_1, \ldots, T_B \mid \eta) + o(n^{-1}).$$

Hence, the total space usage of **aggregate_sums** is at most

$$\begin{aligned} H(\eta \mid T_B) + \sum_{i=1}^{B-1} H(T_i \mid \eta) &\leq H(\eta \mid T_B) + H(T_1, \ldots, T_B \mid \eta) - H(T_B \mid \eta) + o(n^{-1}) \\ &= H(T_1, \ldots, T_{B-1}, \eta \mid T_B) + o(n^{-1}). \end{aligned}$$

Note that the above implementation does not give the right space bound, since the correct space benchmark is $H(T_1, \ldots, T_{B-1} \mid T_B)$ rather than $H(T_1, \ldots, T_{B-1}, \eta \mid T_B)$. In fact, here η encodes extra information about the input, which is going to be encoded one more time in the recursion. This issue is resolved in the final data structure for word RAM by "adding η to the recursion." That is, instead of recursing on a subarray provided that its sum has been stored (as what we do here), we will recurse provided that both

[3] We abuse the notation of a tensor for the corresponding joint distribution.

Fix a partition $\mathcal{K}_E, \mathcal{K}_1, \ldots, \mathcal{K}_r$ of $\{0,1\}^{lB}$ with the above guarantees.

Aggregating the sums:

aggregate_sums$(A, s_1, \ldots, s_B)$

1. for $i = 1, \ldots, B$, compute $T_i = s_1 + \cdots + s_i$
2. $\eta :=$ the index of the set among $\{\mathcal{K}_E, \mathcal{K}_1, \ldots, \mathcal{K}_r\}$ that A belongs to
3. store η conditioned on T_B // T_B, the sum of the entire subarray, is assumed to be stored outside this subroutine
4. for $i = 1, \ldots, B-1$, store T_i conditioned on η
5. return the B stored variables ("$\eta \mid T_B$" , "$T_1 \mid \eta$" , ... , "$T_{B-1} \mid \eta$")

Query algorithms:

retrieve_prefix_sum(s, i) // output T_i provided that $s_1 + \cdots + s_B = s$

1. if $i = 0$, return 0
2. if $i = B$, return s
3. read η conditioned s // $T_B = s$
4. read T_i conditioned on η
5. return T_i

retrieve(s, i) // output s_i provided that $s_1 + \cdots + s_B = s$

1. return **retrieve_prefix_sums**(s, i) $-$ **retrieve_prefix_sums**$(s, i-1)$

Figure 2: **aggregate_sums** "implementation"

the sum and "this extra information encoded by η" have already been stored. Hence, the data structure can avoid storing duplicated information during the recursion. See Section 3.3 and Section 4 for more details. When no information is stored both in η and in the recursion, this subroutine introduces only $o(n^{-1})$ bit of redundancy. Since it is invoked no more than $O(n)$ times in total, the overall space usage is at most $n + 1$ bits.

3.3 Previous data structure and transforming into RAM

As mentioned earlier, the main effort in the previous best-known construction is to transform a "standard" information cell-probe data structure into word RAM. To this end, the *spillover representation* was introduced (see Section 2 for its definition), and it is also heavily used in our data structure.

Spillover representation. One simple example of the spillover representation is to represent the sum of a 0-1 string. Given a 0-1 string of length n, one can design a data structure using space $[K] \times \{0,1\}^m$, such that $K = n^{O(1)}$, the spillover $k \in [K]$ encodes the sum of all n bits, and $m + \log K \leq n + O(n^{-2})$, i.e., it has $O(n^{-2})$ bit of redundancy. To do this, we first permute the representation of all n-bit binary strings. That is, instead of storing the n-bit string as is, we are going to sort all n-bit strings based on their sums, and store the index in the sorted order as an n-bit integer. The idea is that for most inputs, the top $O(\log n)$ bits of the index already reveals the sum. Only when the top bits correspond to a boundary between two adjacent sums, the lower bits will need to be read, in order to determine the actual sum. Let $v = \lceil 3 \log n \rceil$. This problem can be resolved by assigning an integer multiple of 2^{n-v} many indices to each sum. In the other word, all strings with sum equal to 0 are encoded to the interval $[0, 2^{n-v} - 1]$, all strings with sum equal to 1 are encoded to $[2^{n-v}, 2^{n-v+1} - 1]$, etc. For each sum x, all strings with sum equal to x are encoded to an

interval of length

$$\left\lceil \binom{n}{x} \cdot 2^{v-n} \right\rceil \cdot 2^{n-v},$$

aligned to integer multiples of 2^{n-v}. Therefore, the sum can be retrieved without accessing the lower $n-v$ bits of the encoding. There are $n+1$ different values for the sum, the largest number needed for this encoding is at most

$$2^n + 2^{n-v} \cdot (n+1).$$

Now, we set $m = n - v$ and use the m bits of the memory to store the lower $n - v$ bits, and set the spillover to

$$K = (2^n + 2^{n-v} \cdot (n+1))/2^{n-v} = 2^v + n + 1,$$

which stores the top bits of the encoding. The size of the spillover is bounded by $n^{O(1)}$, and the space usage is $m + \log K = \log(2^n + 2^{n-v} \cdot (n+1)) \leq n + O(n^{-2})$. This trick is also used later in the formal proof, e.g., see Lemma 3.

Pătraşcu's rank data structure. The high-level structure of the previous best-known word RAM data structure is similar to the one stated earlier in this section (e.g., see Figure 1): a range tree with branching factor $B = 2$. To construct a rank data structure on an array of length n, one first recurses on the two halves, and obtains two data structures with spillover. It is guaranteed that the two spillover sets are both bounded by $O(n^3)$, and the sum of each half can be decoded from solely the corresponding spillover. To combine the two data structures, the memory bits obtained from the two recursions are concatenated directly. The two spillovers can be combined using the above trick for representing the sum, by sorting all pairs of spillovers by the sum of the whole array (this is possible, since each spillover determines the sum of each half). As we argued above, this step introduces redundancy of $O(n^{-2})$ bit, and the new spillover size is again bounded by $O(n^3)$. By writing down the final spillover in its binary representation, the overall redundancy is no more than one bit.

Adapt our new data structure to RAM. Compared to the previous data structure, our new data structure will use a different algorithm to combine the spillovers (corresponding to **aggregate_sums**), and can afford to set the branching factor B to $(\log n)^{1/3}$. The preprocessing algorithm still recursively constructs a data structure with spillover for each subarray, and provides the same guarantees as in the previous solution: the spillover size is bounded by $\operatorname{poly} n$ and the spillover determines the sum. Then the algorithm combines these B spillovers $k_1, \ldots, k_B$, such that one can retrieve each spillover k_i and compute the sum of first i subarrays in constant time. Observe that when the space usage is very close to the information theoretical lower bound, for each subarray, the distribution of the sum encoded by a random spillover is close to that of the sum of a random input (Fact 1). That is, for a subarray of length l, roughly $\binom{l}{x} \cdot 2^{-l}$ fraction of the spillovers encode the sum equal to x. Therefore, the tensor decomposition argument in the previous subsection also applies when encoding these B spillovers.

More specifically, to compute η given the input, we will partition the set of all possible B-tuples of spillovers, instead of the set of all inputs, according the tensor decomposition. We first store η, and $T_1, \ldots, T_B$ conditioned on η. Then, each spillover k_i is stored conditioned on η and two adjacent prefix sums T_{i-1} and T_i. We will carefully choose the partition $\{\mathcal{K}_i\}$ such that all T_i are almost independent conditioned on η, and moreover, each k_i is almost independent of all other spillovers conditioned on η, T_{i-1}

and T_i. Hence, for a random input, the space usage is

$$\begin{aligned}
&H(\eta) + \sum_{i=1}^{B} H(T_i \mid \eta) + \sum_{i=1}^{B} H(k_i \mid \eta, T_{i-1}, T_i) \\
\approx &H(\eta) + H(T_1, \ldots, T_B \mid \eta) + \sum_{i=1}^{B} H(k_i \mid \eta, T_{i-1}, T_i) \\
\approx &H(T_1, \ldots, T_B, \eta) + \sum_{i=1}^{B} H(k_i \mid \eta, k_1, \ldots, k_{i-1}, T_{i-1}, T_i, k_{i+1}, \ldots, k_B) \\
\leq &H(T_1, \ldots, T_B, \eta) + \sum_{i=1}^{B} H(k_i \mid \eta, k_1, \ldots, k_{i-1}, T_1, \ldots, T_B) \\
= &H(T_1, \ldots, T_B, \eta) + H(k_1, \ldots, k_B \mid \eta, T_1, \ldots, T_B) \\
= &H(k_1, \ldots, k_B),
\end{aligned}$$

which approximately matches the information theoretical lower bound of storing B spillovers.

This solution also handles worst-case input. We will design the partition carefully such that all entropies appeared in the above analysis can be replace by the logarithms of the support sizes. To encode a variable directly (e.g., η), we encode the index within its support, which takes logarithm of the support size many bits. To encode a variable conditioned on another (e.g., $T_i \mid \eta$), the encoding varies based on the value of the conditioning variable. For instance, to encode T_i given η, we examine the value of η, and encode the index of T_i within its support given the value of η, i.e., all values of T_i that are not compatible with the current value of η are removed. When a support size is a perfect power of two, the corresponding variable can be stored using an integer bits of memory. Otherwise, Lemma 6 is applied to produce a data structure with spillover, which introduces no more than n^{-2} bit of redundancy each time. Finally, Lemma 3 ensures that the sum of the whole subarray T_B is encoded in the spillover, to provide the claimed guarantee of the recursion. See the proof of Lemma 5 for more details.

As we can see from the calculation above, it also resolves the issue mentioned in the end of Section 3.2, since after we have encoded the prefix sums, the "remaining information" about each spillover is encoded conditioned on both η and s_i. Each spillover k_i is the only parameter that goes into (or comes back from) the next level of recursion, and k_i contains all relevant information about η and s_i. Hence, no information revealed by η is stored again in the recursion.

Avoid arbitrary word operations. The above construction assumes that the query algorithm can perform arbitrary $O(\log n)$-bit word operations. In Pătraşcu's rank data structure, decoding the two spillovers in constant time also requires non-standard operations. One way to avoid arbitrary word operations is to store a look-up table in the memory. However, the look-up table itself may take $\operatorname{poly} n$ space. Here, we apply a standard trick for self-reducible problems (also applied in the previous data structure) to avoid this issue: divide the input array into blocks of length n^δ, construct the above data structure and look-up table for each block, and store the sum of first i blocks for all i. To answer a prefix sum query in the i-th block, we retrieve the sum of first $i-1$ blocks and make a query in the i-th block. Since all blocks use the same data structure, the corresponding look-up table is also the same, and only one copy needs to be stored. By setting δ to be the right constant, both the size of the look-up table and the total redundancy from all blocks are bounded by n^{1-c} for some constant $c > 0$. See Section 4.3 for details.

4 Succinct Rank Data Structure

Guided by the construction in Section 3, in this section, we present a succinct rank data structure that works in the standard word RAM model. As a starting point, we first present a data structure in the cell-probe model, assuming arbitrary word operations are allowed. In Section 4.3, we show how to implement this solution in word RAM.

Theorem 1 (restated). *Given a 0-1 array of length n for sufficiently large n, for any $t \geq 1$, one can construct a succinct data structure using*

$$n + \lceil \frac{n}{w^{\Omega(t)}} \rceil$$

bits of memory in the cell-probe model with word-size $w \geq 7 \log n$, such that every rank query can be answered in $O(t)$ time.

Proof. The top-level of the data structure is a sequence of range trees, each with branching factor $B = w^{1/3}$ and depth t. In the proof, we assume for simplicity that w is an even perfect cube, and n is a multiple of $B^t w$. General n and w can be handled via similar approaches. Each range tree takes care of a subarray of length $B^t w = w^{t/3+1}$, using $B^t w + 1$ bits of memory, i.e., one bit of redundancy. We also store the number of ones in the first i subarrays for every i using $\lceil \log(n+1) \rceil$ bits. Hence, for every $B^t w$ bits of input, there will be $O(\log n)$ bits of redundancy, which will give us the claimed space bound.

More specifically, consider a subarray of $B^t w$ bits, and a range tree built from it with branching factor B and depth t. Each leaf corresponds to a subarray of length w. Every node at level $t - i$ is the root of a subtree of size $B^i w$ and depth i (assuming root has level 0). We are going to inductively construct data structures for all subtrees. The inductive hypothesis is stated below in the claim.

Claim 1. *For each subtree of size $B^i w$ for $i \geq 0$, one can construct a data structure with spillover, using space $[K_i] \times \{0,1\}^{m_i}$, supporting rank queries in $O(i)$ time, where*

$$m_i = B^i w - w$$

and

$$K_i = 2^w + (34B)^i \cdot n 2^{w/2}.$$

Moreover, the sum of all $B^i w$ bits in the subtree can be computed by only reading the spillover $k_i \in [K_i]$.

Before proving the claim, let us first show that it implies the theorem. When $i = t$,

$$K_t = 2^w + (34B)^t n 2^{w/2} \leq 2^w + n^{2+o(1)} 2^{w/2} \leq 2^{w+1}.$$

Hence, K_t takes $w + 1$ bits to store. The space usage will be $B^t w + 1$ bits.

Finally, we divide the input into $n' = n/(B^t w)$ subarrays $A_1, A_2, \ldots, A_{n'}$ of length $B^t w$. For each subarray, we construct a data structure using Claim 1. We also store the total number of ones in $A_1 \cup A_2 \cup \cdots \cup A_i$ for all $i \in \{1, \ldots, n' - 1\}$, each taking $\lceil \log(n+1) \rceil$ bits. Note that the sum of all n' subarrays is not necessary to store. When $n' = 1$, the redundacy is one bit, otherwise, it is at most $O(n' \log n)$. Hence, the total space usage is at most

$$\frac{n}{B^t w} \cdot (B^t w + 1) + \left(\frac{n}{B^t w} - 1 \right) \lceil \log(n+1) \rceil \leq n + \lceil \frac{n}{w^{\Omega(t)}} \rceil$$

as claimed.

To answer a rank query $\texttt{rank}(u)$, we first compute the subarray A_i that u is in. By retrieving the number of ones in first $i - 1$ subarrays and querying the rank of u within A_i, we obtain the answer in $O(t)$ time. Hence, it remains to prove the claim (by induction).

Base case. The statement is trivial when $i = 0$: store the entire subtree, which has only w bits, in the spillover.

Induction step. First construct a data structure for each child of the root, which corresponds to a subtree of size $l = B^{i-1}w$, using space $[K_{i-1}] \times \{0,1\}^{m_{i-1}}$ each. The key technical part of the induction step is the following lemma that combines B spillovers into one data structure, and allows one to decode each spillover and the sum of first i subtrees in constant time.

Lemma 2. *Given B such spillovers $k_1, k_2, \ldots, k_B \in [2^w+\sigma]$ for $2^{w/2}n \le \sigma \le 2^w/n$, let $\mathrm{SUM} : [2^w+\sigma] \to [0, l]$ be the function that decodes the sum from a spillover. For $i = 0, \ldots, B$, denote by T_i the sum of first i subtrees, i.e., $T_i := \sum_{j \le i} \mathrm{SUM}(k_j)$. One can construct a data structure using space $[K] \times \{0,1\}^m$ for $K = [2^w + 34B\sigma]$ and $m = (B-1)w$, such that for $i = 1, \ldots, B$, decoding each k_i and T_i takes constant time, and the spillover determines the sum of the entire subtree T_B.*

Its proof is deferred to the next subsection. By induction hypothesis, we have

$$n2^{w/2} \le K_{i-1} - 2^w \le (34B)^t n 2^{w/2} \le n^{2+o(1)} 2^{w/2} < 2^w/n,$$

i.e., $n2^{w/2} < \sigma < 2^w/n$. Lemma 2 lets us combine the B spillovers, and obtain a $[K_i] \times \{0,1\}^{(B-1)w}$-space data structure for

$$K_i = 2^w + (34B)(K_{i-1} - 2^w) = 2^w + (34B)^i \cdot n2^{w/2}.$$

Hence, in total the data structure uses $Bm_{i-1} + (B-1)w = B^i w - w$ bits and a spillover of size $2^w + (34B)^i \cdot n2^{w/2}$, and the spillover determines the sum.

To answer a rank query $\texttt{rank}(x)$, we first compute i, the index of the subtree that x is in. Then we retrieve T_{i-1} and k_i in constant time by Lemma 2. Given the spillover k_i, we may recursively query the rank of x inside the i-th subtree. The query output can be computed by adding its rank inside the i-th subtree to T_{i-1}. The total query time is proportional to the depth of the tree, which is $O(i)$. This proves the theorem. □

4.1 Combining the spillovers

The goal of this subsection is to prove Lemma 2. We first observe that if there is a data structure (with spillover) that allows one to decode the sum of the entire subarray (or subtree) in constant time, then one may assume without loss of generality that the sum is encoded in the spillover, which we state in the following lemma.

Lemma 3. *Given input data Z, suppose there is a data structure D using space $[K] \times \{0,1\}^m$, which allows one to answer each query $f_i(Z)$ for $i \ge 0$ in time t_q, assuming the word-size is w. Then D can be turned into another data structure using space $[K + r] \times \{0,1\}^m$, which allows one to answer each query in time $2t_q$, moreover, $f_0(Z)$ can be answered by reading only the spillover, where r is the number of different values that $f_0(Z)$ can take.*

Proof. To construct a data structure that stores $f_0(Z)$ in the spillover, we first simulate D on Z, which generates m bits of memory $s \in \{0,1\}^m$ and a spillover $k \in [K]$. Then we simulate the query algorithm for query $f_0(Z)$, which reads t_q words (or $t_q w$ bits) of s. We move those $t_q w$ bits to the spillover, by increasing the spillover size to $K2^{t_q w}$ and removing them from s. The relative order of all other bits in s are unchanged. This generates a new spillover $k' \in [K2^{t_q w}]$ and a memory s' of $m - t_q w$ bits. Note that the query algorithm can be adaptive, hence the bits removed from s could vary for different inputs.

Now k' does encode $f_0(Z)$, but its size is much larger than claimed. To decrease the spillover size back to approximately K, observe that we are free to choose any bijection between its domain $[K2^{t_q w}]$ and the pair of original spillover k and the $t_q w$ bits. Hence, we will pick a representation of k', such that all its values that encode the same value of $f_0(Z)$ are consecutive in $[K2^{t_q w}]$. For example, we may use a representation such that $f_0(Z)$ is monotone. Intuitively, if each value of $f_0(Z)$ corresponds to an interval in $[K2^{t_q w}]$, reading the "top bits" of k' should likely tell us $f_0(Z)$. However, if k' lies close to the boundary between two consecutive $f_0(Z)$ values, reading the entire k' may still be required to distinguish between the two.

This issue can be resolved by rounding up the boundaries to integer multiples of $2^{t_q w}$. That is, we adjust the representation, so that each interval corresponding to a value of $f_0(Z)$ always starts at a multiple of $2^{t_q w}$. Thus, $f_0(Z)$ can be computed without reading the lowest $t_q w$ bits of k'. Since $f_0(Z)$ can take r different values, this could only increase the spillover set size to at most $(K + r)2^{t_q w}$, which can be viewed as $[K + r] \times \{0, 1\}^{t_q w}$ By moving these $t_q w$ bits back to (the beginning of) the memory, we obtain a data structure using space $[K + r] \times \{0, 1\}^m$, such that $f_0(Z)$ can be answered by reading only the spillover.

To answer a generic query $f_i(Z)$ for $i \geq 1$, one first reads the spillover and first t_q words of the memory. This determines k', and hence the initial spillover k generated from D as well as all words read by the query algorithm of D when $f_0(Z)$ is queried, which are the words removed from s. In particular, this determines the mapping between words in s' and s. Thus, $f_i(Z)$ can be computed by simulating the query algorithm of D. The total query time is $2t_q$. □

In particular, when Z is a subarray of size Bl, and $f_0(Z)$ is the number of ones in it, we have $r = Bl + 1 \leq n + 1$. Note that this lemma is applied once at each level of the recursion, thus the total query time would at most increase by a factor of two. To prove Lemma 2, we will use different constructions based on the value of l, the length of each subarray. The two cases are stated below in Lemma 4 and Lemma 5 respectively. Since $n + 1 \leq B\sigma$, Lemma 2 is an immediate corollary of Lemma 3, 4 and 5.

Lemma 4. *If $B \log(l + 1) \leq w/2$, given $k_1, \ldots, k_B \in [2^w + \sigma]$ for $n2^{w/2} \leq \sigma \leq 2^w/n$, one can construct a data structure using space $[K] \times \{0, 1\}^m$ for $m = (B - 1)w$ and $K \leq 2^w + 2B\sigma$, such that each k_i and T_i can be decoded in constant time.*

Lemma 5. *If $B \log(l + 1) > w/2$, given $k_1, \ldots, k_B \in [2^w + \sigma]$ for $n2^{w/2} \leq \sigma \leq 2^w/n$, one can construct a data structure using space $[K] \times \{0, 1\}^m$ for $m = (B - 1)w$ and $K \leq 2^w + 33B\sigma$, such that each k_i and T_i can be decoded in constant time.*

Recall that each spillover $k_i \in [2^w + \sigma]$ together with $l - w$ additional bits encodes a subarray of length l, and k_i encodes the number of ones in this subarray, which is decoded by the function SUM. Since the space usage is close to the information theoretical limit, if we sample a random k_i, the distribution of $\mathrm{SUM}(k_i)$ should be close to the distribution of the sum of the subarray, i.e., the binomial distribution $B(l, 1/2)$. In particular, we have the following facts by counting.

Fact 1. *For every $x \in [0, l]$, we have*

$$|\mathrm{SUM}^{-1}(x)| \geq \binom{l}{x} \cdot 2^{-l+w}.$$

For any subset $X \subseteq [0, l]$,

$$|\mathrm{SUM}^{-1}(X)| \leq \sigma + \sum_{x \in X} \binom{l}{x} \cdot 2^{-l+w}.$$

Proof. The encoding supports $\texttt{rank}(u)$ operations for all $0 \le u \le l$, and the answers to all queries recover the entire subarray. Hence, all 2^l different subarrays of length l must have different encodings. On the other hand, for every $x \in [0, l]$, the number of subarrays with sum equal to x is $\binom{l}{x}$, and each k_i corresponds to only 2^{l-w} different encodings. Therefore, at least $\binom{l}{x} \cdot 2^{-l+w}$ different k_i should encode arrays with the sum equal to x, i.e.,

$$|\mathrm{SUM}^{-1}(x)| \ge \binom{l}{x} \cdot 2^{-l+w}.$$

By subtracting the complement of X from the universe, we have

$$\begin{aligned}
|\mathrm{SUM}^{-1}(X)| &= 2^w + \sigma - \sum_{x \notin X} |\mathrm{SUM}^{-1}(x)| \\
&\le 2^w + \sigma - \sum_{x \notin X} \binom{l}{x} \cdot 2^{-l+w} \\
&= \sigma + \sum_{x \in X} \binom{l}{x} \cdot 2^{-l+w}.
\end{aligned}$$

□

One crucial subroutine used in several parts of our construction is a succinct data structure storing "uniform and independent" elements with nearly no redundancy from [DPT10], which we state in the following lemma.

Lemma 6. *Suppose we are given a B-tuple $(x_1, x_2, \ldots, x_B) \in [M_1] \times [M_2] \times \cdots \times [M_B]$, such that $B \ll 2^{w/2}$ and $M_i \le 2^w$ for every $i \in [B]$. Then for every integer $m \ge \sum_{i=1}^{B} \log M_i - w$, there is a data structure that uses space $[K] \times \{0,1\}^m$ for $K = \left\lceil 2^{-m} \cdot \prod_{i=1}^{B} M_i \right\rceil + 1$, and allows one to decode each x_i in constant time.*

The original theorem in [DPT10] is stated only for numbers from the same domain, i.e., $M_1 = \cdots = M_B$. However, the same idea also applies when the domains are different. The proof of Lemma 6 can be found in Appendix A.

We first present the construction for small l, which is similar to [Păt08].

Proof of Lemma 4. When $B \log(l+1) \le w/2$, the sums of B subarrays can all fit in one spillover. We use the spillover to store the sums, then encode the subarrays *conditioned on* the sums.

More specifically, let $m = (B-1)w$. For every B-tuple of sums $\mathbf{s} = (s_1, \ldots, s_B) \in [0, l]^B$, by Fact 1, we have

$$\begin{aligned}
\sum_{i=1}^{B} \log \left|\mathrm{SUM}^{-1}(s_i)\right| - w &\le \sum_{i=1}^{B} \log \left(\sigma + \binom{l}{s_i} \cdot 2^{-l+w} \right) - w \\
&\le \sum_{i=1}^{B} \log \left(2^{w-1} + 2^w/\sqrt{l} \right) - w \\
&\le Bw - w \\
&\le m.
\end{aligned}$$

By Lemma 6, for every $\mathbf{s}$, there is a data structure encoding a tuple $(k_1, \ldots, k_B)$ such that $\mathrm{SUM}(k_i) = s_i$, using space $[K_{\mathbf{s}}] \times \{0,1\}^m$, where

$$K_{\mathbf{s}} \le 2^{-Bw+w} \cdot \prod_{i=1}^{B} |\mathrm{SUM}^{-1}(s_i)| + 2.$$

We then "glue together" these $(l+1)^B$ data structures for different tuples of sums by taking the union of the spillover sets. That is, let $K = \sum_{\mathbf{s}} K_{\mathbf{s}}$, we can view $[K]$ as the set of pairs $\{(\mathbf{s}, k) : \mathbf{s} \in [0,l]^B, k \in [K_{\mathbf{s}}]\}$ (via a fixed bijection hard-wired in the data structure). Given an input $(k_1, \ldots, k_B)$, we first compute the sums $\mathbf{s} = (s_1, \ldots, s_B)$, and encode the input using the above data structure for $\mathbf{s}$, which generates m bits and a spillover $k \in [K_{\mathbf{s}}]$. The final data structure will consist of these m bits and the spillover $(\mathbf{s}, k)$, encoded in $[K]$. The size of the spillover set is at most

$$\begin{aligned}
K &= \sum_{\mathbf{s} \in [0,l]^B} K_{\mathbf{s}} \\
&\le \sum_{\mathbf{s} \in [0,l]^B} (2^{-Bw+w} \cdot \prod_{i=1}^{B} |\mathrm{SUM}^{-1}(s_i)| + 2) \\
&= 2^{-Bw+w} \cdot \prod_{i=1}^{B} \sum_{s_i=0}^{l} |\mathrm{SUM}^{-1}(s_i)| + 2(l+1)^B \\
&\le 2^{-Bw+w} \cdot (2^w + \sigma)^B + 2^{w/2+1} \\
&= 2^w \cdot (1 + \sigma 2^{-w})^B + 2^{w/2+1},
\end{aligned}$$

which by the bounds on σ that $n2^{w/2} < \sigma < 2^w/n$, is at most

$$\le 2^w + 2B\sigma.$$

Decoding T_i or k_i can be done in constant time by a straightforward algorithm: First decode the pair $(\mathbf{s}, k)$, which already determines the value of $T_i (= s_1 + \cdots + s_i)$, k_i can then be decoded using the decoding algorithm for tuple $\mathbf{s}$ from Lemma 6. □

When l is large, the key step is to find a random variable η such that all T_i are uniform and independent conditioned on (most values of) η. This allows us to first encode η, then encode the prefix sums $\{T_i\}$ nearly optimally using Lemma 6. Finally, we encode the spillovers $\{k_i\}$ conditioned on the prefix sums.

In the following lemma, we first present a "not-so-efficient" solution, which will be used as a subroutine in our final construction.

Lemma 7. *For any $B' \le B$, given a sequence $(k_1, \ldots, k_{B'})$, one can construct a data structure using $B'(w + \log w)$ bits of space, such that each k_i and T_i can be retrieved in constant time.*

Proof. We first partition $[2^w + \sigma]$, the domain of each k_i, into two sets based on the sum it encodes: $\mathcal{K}_{\text{high}} := \mathrm{SUM}^{-1}(l/2 \pm \sqrt{lw})$ and $\mathcal{K}_{\text{low}} := \mathrm{SUM}^{-1}([0,l] \setminus (l/2 \pm \sqrt{lw}))$. The idea is that if several consecutive spillovers are in $\mathcal{K}_{\text{high}}$, then it takes few bits to encode their sum; if a spillover is in $\mathcal{K}_{\text{low}}$, then it takes few bits to encode the spillover itself. By Fact 1,

$$|\mathcal{K}_{\text{low}}| = |\mathrm{SUM}^{-1}([0,l] \setminus (l/2 \pm \sqrt{lw}))|$$

$$\begin{aligned}
&\le \sigma + \sum_{x \notin l/2 \pm \sqrt{lw}} \binom{l}{x} 2^{-l+w} \\
&\le \sigma + 2^{w+1-2w} \\
&\le \sigma + 1.
\end{aligned}$$

The first B' bits encode for each $i \in [B']$, if k_i is in $\mathcal{K}_{\text{high}}$ or in $\mathcal{K}_{\text{low}}$. Then we allocate $w + \lfloor \log w \rfloor - 1$ consecutive bits to each k_i, where we store extra information about each k_i as follows.

If $k_i \in \mathcal{K}_{\text{low}}$, we spend $\lceil \log(Bl+1) \rceil$ bits to write down T_i, the sum of first i blocks, and $\lceil \log(\sigma+1) \rceil$ bits to encode k_i within $\mathcal{K}_{\text{low}}$. The space usage is at most

$$\log Bl + \log \sigma + 2 \le w + 2 < w + \lfloor \log w \rfloor - 1$$

bits, since $\sigma < 2^w/n$ and $Bl \le n$.

If $k_i \in \mathcal{K}_{\text{high}}$, denote by i_{pred} the closest block preceding i that is not in $\mathcal{K}_{\text{high}}$, i.e.,

$$i_{\text{pred}} := \max\{j : j < i, k_j \in \mathcal{K}_{\text{low}} \text{ or } j = 0\}.$$

We first spend $\lceil \log(2B\lfloor \sqrt{lw} \rfloor + 1) \rceil$ bits to encode $T_i - T_{i_{\text{pred}}}$. This is possible since all subarrays in-between have their sums in a consecutive range of length $2\lfloor \sqrt{lw} \rfloor$. Then we spend another $\lceil \log(\sigma + 2^w/\sqrt{l}) \rceil$ bits to encode k_i conditioned on $\text{SUM}(k_i)$. Again such encoding is possible, since for any x, by Fact 1,

$$\begin{aligned}
\left|\text{SUM}^{-1}(x)\right| &\le \sigma + \binom{l}{x} \cdot 2^{-l+w} \\
&\le \sigma + \binom{l}{\lfloor l/2 \rfloor} \cdot 2^{-l+w} \\
&\le \sigma + 2^w/\sqrt{l}.
\end{aligned}$$

In this case, the space usage is at most

$$\begin{aligned}
\log(2B\sqrt{lw}) + \log(\sigma + 2^w/\sqrt{l}) + 2 &= \log(2B\sqrt{lw}\sigma + 2^{w+1}B\sqrt{w}) + 2 \\
&\le \log(2^{w+1}\sqrt{w/l} + 2^{w+1}B\sqrt{w}) + 2 \\
&= w + 1 + \log B + \frac{1}{2}\log w + \log((B\sqrt{l})^{-1} + 1) + 2 \\
&\le w + \log w - 1.
\end{aligned}$$

where the first inequality uses $\sigma < 2^w/n$ and $Bl \le n$, and the last inequality uses $B = w^{1/3}$.

The total space usage is at most $B'(w + \log w)$. To decode T_i, one reads the first B' bits in constant time (as $B' < w$) to retrieve for *every* j, whether $k_j \in \mathcal{K}_{\text{high}}$ or $\mathcal{K}_{\text{low}}$. If $k_i \in \mathcal{K}_{\text{low}}$, we have explicitly stored the value of T_i. Retrieving its value thus takes constant time. If $k_i \in \mathcal{K}_{\text{high}}$, one first computes i_{pred} using the B' bits retrieved earlier, and reads $T_i - T_{i_{\text{pred}}}$. It reduces the problem to decoding $T_{i_{\text{pred}}}$. If $i_{\text{pred}} = 0$, the problem is solved. Otherwise, $k_{i_{\text{pred}}} \in \mathcal{K}_{\text{low}}$, and one may apply the above query algorithm. In all cases, T_i can be decoded in constant time.

To decode k_i, if $k_i \in \mathcal{K}_{\text{low}}$, k_i is also explicitly encoded. Otherwise, $k_i \in \mathcal{K}_{\text{high}}$, and one applies the above query algorithm to retrieve both T_i and T_{i-1}, thus determines the value of $\text{SUM}(k_i)$ by taking their difference. The value of k_i conditioned on $\text{SUM}(k_i)$ is encoded in the data structure, and can therefore be retrieved in constant time.

See Figure 3 for the construction pictorially. □

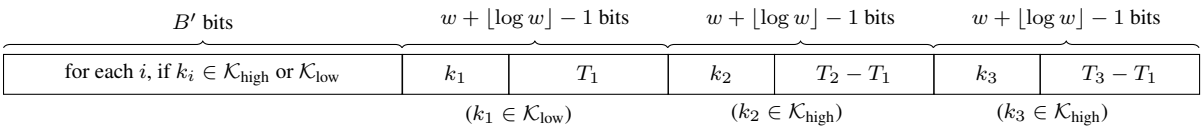

Figure 3: memory content for $B' = 3$

To prove Lemma 5, we will need the following lemma for approximating binomial coefficients, whose proof is presented in the next subsection.

Lemma 8. *For any large even integer l, positive numbers M_x, M_y and ϵ, such that $l > 8M_x$, $l > 8M_y$ and $\epsilon > 2^{-C\sqrt{l}/2+8}$, we have*

$$\binom{l}{l/2+x+y} \cdot 2^{-l+w} = E(x,y) + \sum_{i=1}^{r} 2^{e_i} \mathbf{1}_{X_i}(x) \mathbf{1}_{Y_i}(y)$$

for all integers $x \in [-M_x, M_x]$ and $y \in [-M_y, M_y]$, such that

a) $E(x,y) \geq 0$ and for every $x \in [-M_x, M_x]$, $\sum_{y=-M_y}^{M_y} E(x,y) \leq \epsilon 2^w + 4rM_y 2^{w/2}$;

b) for every $i \in [r]$, $e_i \geq 0$ is an integer, $X_i \subseteq [-M_x, M_x]$, $Y_i \subseteq [-M_y, M_y]$ are sets of integers;

c) $r \leq O((M_x M_y/l) w^2 \log^4(1/\epsilon))$.

Using the above two lemmas, we are ready to prove Lemma 5.

Proof of Lemma 5. By Fact 1, for each $i \in [B]$, $\mathrm{SUM}(k_i) = T_{i+1} - T_i$ is distributed approximately according to the binomial distribution $B(l, 1/2)$ for a random k_i. We first apply Lemma 8 to approximate the probability masses of $B(l, 1/2)$, which are binomial coefficients, hence approximating $|\mathrm{SUM}^{-1}(x)|$ for $x \in [0, n]$.

For each $i \in [B]$, let $\epsilon = \sigma \cdot 2^{-w-2}$, $M_x = (i-1)\sqrt{l \log 1/\epsilon}$, $M_y = i\sqrt{l \log 1/\epsilon}$. Since $B = w^{1/3}$ and $B \log(l+1) \geq w/2$, we have

$$l/M_y \geq \sqrt{l/(B^2 \log 1/\epsilon)} \geq \sqrt{l}/w > 8,$$

similarly $l > 8M_x$, and

$$\epsilon = 2^{-w-2+\log \sigma} \geq 2^{-w-2} \geq 2^{-O(\log^2 l)} \geq 2^{-o(\sqrt{l})}.$$

Hence, by Lemma 8 (setting $x = (i-1)l/2 - T_{i-1}$ and $y = T_i - il/2$), there exists E_i, $X_{i,j}$ and $Y_{i,j}$ such that for all $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$ and $T_i \in i(l/2 \pm \sqrt{l \log 1/\epsilon})$,

$$\binom{l}{T_i - T_{i-1}} \cdot 2^{-l+w} = E_i(T_{i-1}, T_i) + \sum_{j=1}^{r_i} 2^{e_{i,j}} \mathbf{1}_{X_{i,j}}(T_{i-1}) \mathbf{1}_{Y_{i,j}}(T_i) \tag{3}$$

for integers $e_{i,j} \geq 0$ and

$$r_i \leq O((M_x M_y/l) w^2 \log^4(1/\epsilon)) \leq O(B^2 w^2 \log^5 1/\epsilon) \leq w^{O(1)}.$$

Since $\mathrm{SUM}(k_i)$ approximately follows a binomial distribution, we can partition its domain according to Equation (3), as follows.

Claim 2. *For every $i \in [B]$ and $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$, there exists a partition of $[2^w + \sigma]$ (domain of k_i) into $\{\mathcal{K}_{i,j}^{(T_{i-1})}\}_{0 \le j \le r_i}$, such that*

(a) $|\mathcal{K}_{i,0}^{(T_{i-1})}| \le 2\sigma$ and for all $k_i \in [2^w + \sigma]$ such that $\mathrm{SUM}(k_i) \notin l/2 \pm \sqrt{l \log 1/\epsilon}$, we have $k_i \in \mathcal{K}_{i,0}^{(T_{i-1})}$;

(b) for $j = 1, \ldots, r_i$ and $T_i \in T_{i-1} + l/2 \pm \sqrt{l \log 1/\epsilon}$,

$$\left|\mathcal{K}_{i,j}^{(T_{i-1})} \cap \mathrm{SUM}^{-1}(T_i - T_{i-1})\right| = 2^{e_{i,j}} \mathbf{1}_{X_{i,j}}(T_{i-1}) \mathbf{1}_{Y_{i,j}}(T_i).$$

To focus on the construction of our data structure, we deferred its proof to the end of the subsection. Now let us fix one such partition $\{\mathcal{K}_{i,j}^{(T_{i-1})}\}_{0 \le j \le r_i}$ for every $i \in [B]$ and $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$.

Definition 1. *For $B' \le B$, a sequence of B' spillovers $(k_1, \ldots, k_{B'})$ is* good*, if for every $i \in [B']$, $k_i \notin \mathcal{K}_{i,0}^{(T_{i-1})}$, where $T_i = \sum_{j \le i} \mathrm{SUM}(k_j)$.*

Note that $\mathcal{K}_{i,0}^{(T_{i-1})}$ is only defined when $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$. However, if $k_{i-1} \notin \mathcal{K}_{i-1,0}^{(T_{i-2})}$, then by Item (a) above, $\mathrm{SUM}(k_{i-1}) \in l/2 \pm \sqrt{l \log 1/\epsilon}$. Hence, if $T_{i-2} \in (i-2)(l/2 \pm \sqrt{l \log 1/\epsilon})$, we must also have $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$, and thus "good sequences" are well-defined.

Now we are ready to describe our construction for large l. We first handle good sequences.

Input sequence $(k_1, \ldots, k_B)$ is good. Given a good sequence $(k_1, \ldots, k_B)$, one can compute $(T_0, \ldots, T_B)$, and for each i, the index of set j_i which k_i is in according to the partition, i.e., $j_i \in \{1, \ldots, r_i\}$ such that $k_i \in \mathcal{K}_{i,j_i}^{(T_{i-1})}$. We first construct a data structure *given* the sequence of indices $\mathbf{j} = (j_1, \ldots, j_B)$.

Claim 3. *For every $\mathbf{j} = (j_1, \ldots, j_B)$ such that $j_i \in \{1, \ldots, r_i\}$ for all $i \in [B]$, given a sequence of spillovers $(k_1, \ldots, k_B)$ such that $k_i \in \mathcal{K}_{i,j_i}^{(T_{i-1})}$, one can construct a data structure using space $[K_{\mathbf{j}}] \times \{0,1\}^{(B-1)w}$ for*

$$K_{\mathbf{j}} = \left\lceil 2^{-(B-1)w} \cdot \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \left|X_{i+1,j_{i+1}} \cap Y_{i,j_i}\right|\right) \right\rceil + 1,$$

which allows one to decode each k_i and T_i in constant time.[4]

To construct such a data structure, we are going to encode each k_i within $\mathcal{K}_{i,j_i}^{(T_{i-1})} \cap \mathrm{SUM}^{-1}(T_i - T_{i-1})$, and encode each T_i within $X_{i+1,j_{i+1}} \cap Y_{i,j_i}$ using Lemma 6.

More specifically, for every i, we know that $k_i \in \mathcal{K}_{i,j_i}^{(T_{i-1})}$ and $\mathrm{SUM}(k_i) = T_i - T_{i-1}$. One can spend e_{i,j_i} bits to encode the index of k_i within $\mathcal{K}_{i,j_i}^{(T_{i-1})} \cap \mathrm{SUM}^{-1}(T_i - T_{i-1})$, which has size at most $2^{e_{i,j_i}}$ by Item (b) in Claim 2 (in fact, it will be exactly $2^{e_{i,j_i}}$). Note that the encoding length of this part does not depend on the input.

We also know that by Item (b), no input sequence will have $T_{i-1} \notin X_{i,j_i}$ or $T_i \notin Y_{i,j_i}$ for any $i \in [B]$. That is, we must have $T_i \in X_{i+1,j_{i+1}} \cap Y_{i,j_i}$. One can thus apply Lemma 6 to encode each T_i within the set $X_{i+1,j_{i+1}} \cap Y_{i,j_i}$, for $m = (B-1)w - \sum_{i=1}^{B} e_{i,j_i}$. The premise of the lemma is satisfied, since

$$m - (\sum_{i=1}^{B} \log \left|X_{i+1,j_{i+1}} \cap Y_{i,j_i}\right| - w) = Bw - \sum_{i=1}^{B} \left(e_{i,j_i} + \log \left|X_{i+1,j_{i+1}} \cap Y_{i,j_i}\right|\right)$$

[4] $X_{B+1,j_{B+1}}$ is assumed to be the entire domain $[2^w + \sigma]$.

$$\geq \sum_{i=1}^{B} (w - e_{i,j_i} - \log |Y_{i,j_i}|)$$
$$\geq 0.$$

The last inequality is due to Equation (3). Hence, Lemma 6 constructs a data structure with spillover size

$$K_{\mathbf{j}} = \left\lceil 2^{-m} \cdot \prod_{i=1}^{B} \left| X_{i+1,j_{i+1}} \cap Y_{i,j_i} \right| \right\rceil + 1$$
$$= \left\lceil 2^{-(B-1)w} \cdot \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \left| X_{i+1,j_{i+1}} \cap Y_{i,j_i} \right| \right) \right\rceil + 1.$$

The total space usage is $(B-1)w$ bits with a spillover of size $K_{\mathbf{j}}$.

To decode a T_i, one can simply invoke the decoding algorithm from Lemma 6, since $\mathbf{j}$ is given and all sets $X_{i+1,j_{i+1}} \cap Y_{i,j_i}$ are known. To decode a k_i, one first decodes T_{i-1} and T_i, after which both sets $\mathcal{K}_{i,j_i}^{(T_{i-1})}$ and $\mathrm{SUM}^{-1}(T_i - T_{i-1})$ are known. Then k_i can be decoded by retrieving its index within $\mathcal{K}_{i,j_i}^{(T_{i-1})} \cap \mathrm{SUM}^{-1}(T_i - T_{i-1})$.

To obtain a data structure for all good sequences, we "glue" the above data structures for all $\mathbf{j}$ in a similar way to Lemma 4. Let $K_{\text{good}} = \sum_{\mathbf{j}} K_{\mathbf{j}}$. We may view the set $[K_{\text{good}}]$ as $\{(\mathbf{j}, k) : k \in [K_{\mathbf{j}}]\}$ (via a fixed bijection hard-wired in the data structure). Given a good sequence $k_1, \ldots, k_B$, one first computes $\mathbf{j} = (j_1, \ldots, j_B)$ and constructs a data structure using Claim 3, which generates $(B-1)w$ bits and a spillover $k \in [K_{\mathbf{j}}]$. The data structure will consist of these $(B-1)w$ bits and a final spillover of pair $(\mathbf{j}, k)$, encoded in $[K_{\text{good}}]$. To decode T_i or k_i, it suffices to decode the pair $(\mathbf{j}, k)$, and then invoke the decoding algorithm from Claim 3.

The spillover size is

$$K_{\text{good}} = \sum_{\mathbf{j} \in [r_1] \times \cdots \times [r_B]} K_{\mathbf{j}}$$
$$\leq \sum_{\mathbf{j} \in [r_1] \times \cdots \times [r_B]} \left(2^{-(B-1)w} \cdot \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \left| X_{i+1,j_{i+1}} \cap Y_{i,j_i} \right| \right) + 2 \right)$$
$$= 2^{-(B-1)w} \cdot \sum_{\mathbf{j} \in [r_1] \times \cdots \times [r_B]} \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \cdot \sum_{T_i \in il/2 \pm i\sqrt{l \log 1/\epsilon}} \mathbf{1}_{X_{i+1,j_{i+1}}}(T_i) \mathbf{1}_{Y_{i,j_i}}(T_i) \right) + w^{O(B)}$$
$$= 2^{-(B-1)w} \cdot \sum_{\mathbf{j} \in [r_1] \times \cdots \times [r_B]} \sum_{\substack{T_0, T_1, \ldots, T_B: \\ T_i \in il/2 \pm i\sqrt{l \log 1/\epsilon}}} \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \cdot \mathbf{1}_{X_{i+1,j_{i+1}}}(T_i) \mathbf{1}_{Y_{i,j_i}}(T_i) \right) + w^{O(B)},$$

which by the fact that $\mathbf{1}_{X_{B+1,i_{B+1}}}(T_B) = \mathbf{1}_{X_{1,i_1}}(T_0) = 1$, is equal to

$$= 2^{-(B-1)w} \cdot \sum_{\substack{T_0, T_1, \ldots, T_B: \\ T_i \in il/2 \pm i\sqrt{l \log 1/\epsilon}}} \sum_{\mathbf{j} \in [r_1] \times \cdots \times [r_B]} \prod_{i=1}^{B} \left(2^{e_{i,j_i}} \cdot \mathbf{1}_{X_{i,j_i}}(T_{i-1}) \mathbf{1}_{Y_{i,j_i}}(T_i) \right) + w^{O(B)}$$

$$= 2^{-(B-1)w} \cdot \sum_{\substack{T_0,T_1,\ldots,T_B: \\ T_i \in il/2 \pm i\sqrt{l \log 1/\epsilon}}} \prod_{i=1}^{B} \sum_{j_i=1}^{r_i} 2^{e_{i,j_i}} \cdot \mathbf{1}_{X_{i,j_i}}(T_{i-1}) \mathbf{1}_{Y_{i,j_i}}(T_i) + w^{O(B)},$$

which by Equation (3), is at most

$$\leq 2^{-(B-1)w} \cdot \sum_{T_0,T_1,\ldots,T_B: T_0=0} \prod_{i=1}^{B} \left(\binom{l}{T_i - T_{i-1}} \cdot 2^{-l+w} \right) + w^{O(B)}$$

$$= 2^{w} \cdot \sum_{T_0,T_1,\ldots,T_B: T_0=0} \prod_{i=1}^{B} \left(\binom{l}{T_i - T_{i-1}} \cdot 2^{-l} \right) + w^{O(B)}$$

$$= 2^{w} + w^{O(B)}.$$

Hence, one can construct a data structure for good sequence using space $[K_{\text{good}}] \times \{0,1\}^{(B-1)w}$ for

$$K_{\text{good}} = 2^w + w^{O(B)}, \tag{4}$$

such that each k_i and T_i can be decoded in constant time. In particular, one may also choose to use $Bw + 1$ bits in total, by rounding up the spillover to $w + 1$ bits. One can verify that the above construction also applies to any shorter good sequence of length $B' \leq B$, using $B'w + 1$ bits of space, which will be used as a separate subroutine below.

When the input sequence is not good, there is a smallest i^* such that $k_{i^*} \in \mathcal{K}_{i^*,0}^{(T_{i^*-1})}$. We are going to use different constructions based on whether the suffix $(k_{i^*+1}, \ldots, k_B)$ is good, i.e., whether we have for every $i = i^* + 1, \ldots, B$, $k_i \notin \mathcal{K}_{i-i^*,0}^{(T_{i-1}-T_{i^*})}$. If $(k_{i^*+1}, \ldots, k_B)$ is good, we apply the above construction for good sequences to both prefix and suffix. The details are presented below.

i^* breaks the input into two good subsequences. Suppose the input has one $i^* \in [B]$ such that

- $k_{i^*} \in \mathcal{K}_{i^*,0}^{(T_{i^*-1})}$;
- $(k_1, \ldots, k_{i^*-1})$ is good;
- $(k_{i^*+1}, \ldots, k_B)$ is good.

Since $|\mathcal{K}_{i^*,0}^{(T_{i^*-1})}| \leq 2\sigma$, one can spend $\lceil \log B \rceil + \lceil \log 2\sigma \rceil$ bits to encode i^* and the index of k_{i^*} within $\mathcal{K}_{i^*,0}^{(T_{i^*-1})}$. Then by the above construction for good sequences, one can construct a data structure for $(k_1, \ldots, k_{i^*-1})$ using $(i^* - 1)w + 1$ bits, and a data structure for $(k_{i^*+1}, \ldots, k_B)$ using $(B - i^*)w + 1$ bits. Hence, the total space is at most

$$\lceil \log B \rceil + \lceil \log 2\sigma \rceil + (i^* - 1)w + 1 + (B - i^*)w + 1 \leq (B-1)w + \lfloor \log B\sigma \rfloor + 5$$

bits. By converting the extra (at most) $\lfloor \log B\sigma \rfloor + 5$ bits to the spillover, one obtains a data structure using space $[K_{\text{bad}}] \times \{0,1\}^{(B-1)w}$ for

$$K_{\text{bad}} = 32B\sigma. \tag{5}$$

To decode T_i, one first retrieves i^*. If $i < i^*$, T_i can be decoded from the data structure for $(k_1, \ldots, k_{i^*-1})$ in constant time. If $i = i^*$, we have $T_{i^*} = T_{i^*-1} + \mathrm{SUM}(k_{i^*})$. The former term can be decoded in constant time from the data structure for the prefix, the latter term can be computed from k_{i^*}, which is explicitly stored in memory once T_{i^*-1} is computed. If $i > i^*$, one can first compute T_{i^*}, then compute $T_i - T_{i^*}$ by querying the data structure for the suffix.

To decode k_i, if $i < i^*$ (or $i > i^*$), k_i can be decoded from the data structure for the prefix (or the data structure for the suffix). If $i = i^*$, one computes T_{i^*-1} using the above algorithm, and decodes the index of k_{i^*} within $\mathcal{K}_{i^*,0}^{(T_{i^*-1})}$, which determines k_{i^*}. This completes the construction when i^* breaks the sequence into two good subsequences.

If the suffix $(k_{i^*+1}, \ldots, k_B)$ is not good either, then there exists another index $i_2^* > i^*$ such that $k_{i_2^*} \in \mathcal{K}_{i_2^*-i^*,0}^{(T_{i_2^*-1}-T_{i^*})}$, which also has size at most 2σ. For a random input sequence, such case happens sufficiently rarely, so that the "not-so-efficient" solution of Lemma 7 becomes affordable. We present the details below.

There exist $i_1^* < i_2^*$ which are both in sets of size at most 2σ. Suppose there exist i_1^* and i_2^* such that

- $k_{i_1^*} \in \mathcal{K}_{i_1^*,0}^{(T_{i_1^*-1})}$;
- $k_{i_2^*} \in \mathcal{K}_{i_2^*-i_1^*,0}^{(T_{i_2^*-1}-T_{i_1^*})}$.

Note that both $\mathcal{K}_{i_1^*,0}^{(T_{i_1^*-1})}$ and $\mathcal{K}_{i_2^*-i_1^*,0}^{(T_{i_2^*-1}-T_{i_1^*})}$ have size at most 2σ.

We first spend $2\lceil \log B \rceil$ bits to encode i_1^* and i_2^*, and another $2\lceil \log 2\sigma \rceil$ bits to encode $k_{i_1^*}$ within $\mathcal{K}_{i_1^*,0}^{(T_{i_1^*-1})}$ and k_{i_2*} within $\mathcal{K}_{i_2^*-i_1^*,0}^{(T_{i_2^*-1}-T_{i_1^*})}$. i_1^* and i_2^* break the input sequence into three consecutive subsequences. Next we apply Lemma 7 to separately encode each subsequence: $(k_1, \ldots, k_{i_1^*-1})$, $(k_{i_1^*+1}, \ldots, k_{i_2^*-1})$ and $(k_{i_2^*+1}, \ldots, k_B)$. Hence, the total space usage is at most

$$\begin{aligned} & 2\log B + 2\log 2\sigma + 6 + (B-2)(w + \log w) \\ \leq\ & (B-1)w + 2\log B\sigma + 8 + B\log w - w \end{aligned}$$

bits. By converting the extra (at most) $2\log B\sigma + 8 + B\log w - w$ bits to the spillover, one obtains a data structure using space $[K_{\text{lowprob}}] \times \{0,1\}^{(B-1)w}$ for

$$K_{\text{lowprob}} = (B\sigma)^2 w^B 2^{-w+8}. \tag{6}$$

To decode a T_i, one first retrieves i_1^* and i_2^*. If $i < i_1^*$, T_i can be decoded from the data structure for the first subsequence $(k_1, \ldots, k_{i_1^*-1})$. If $i = i_1^*$, one first decodes $T_{i_1^*-1}$, which determines the set $\mathcal{K}_{i_1^*,0}^{(T_{i_1^*-1})}$. $k_{i_1^*}$ becomes retrievable in constant time, and $T_{i_1^*}$ can be computed as $T_{i_1^*-1} + \mathrm{SUM}(k_{i_1^*})$. When $i_1^* < i < i_2^*$, T_i can be computed from $T_{i_1^*}$ and the data structure for $(k_{i_1^*+1}, \ldots, k_{i_2^*-1})$. Decoding T_i in the cases when $i = i_2^*$ or $i > i_2^*$ is similar. Likewise, one can also decode each k_i in constant time.

The final data structure. To handle a general input sequence $(k_1, \ldots, k_B)$, we again glue the above three data structures together. By setting $K = K_{\text{good}} + K_{\text{bad}} + K_{\text{lowprob}}$, we obtain a data structure using space $[K] \times \{0,1\}^{(B-1)w}$ which allows one to decode each k_i and each T_i in constant time. Here, by Equation (4), (5) and (6), we have

$$\begin{aligned} K &\le 2^w + w^{O(B)} + 32B\sigma + (B\sigma)^2 w^B 2^{-w+8} \\ &\le 2^w + 32B\sigma + (B\sigma)^2 w^{O(B)} 2^{-w}, \end{aligned}$$

which by the fact that $\sigma < 2^w/n$, is at most

$$\begin{aligned} &\le 2^w + B\sigma(32 + Bw^{O(B)}/n) \\ &\le 2^w + 33B\sigma, \end{aligned}$$

since $n \ge l \ge 2^{w/2B} - 1$ and $B = w^{1/3}$. This proves the lemma. □

Proof of Claim 2. By Fact 1 and Equation (3), we have

$$|\mathrm{SUM}^{-1}(T_i - T_{i-1})| \ge \binom{l}{T_i - T_{i-1}} \cdot 2^{-l+w} \ge \sum_{j=1}^{r_i} 2^{e_{i,j}} \mathbf{1}_{X_{i,j}}(T_{i-1}) \mathbf{1}_{Y_{i,j}}(T_i).$$

Hence Item (b) can be satisfied, e.g., by setting

$$\begin{aligned} \mathcal{K}_{i,j}^{(T_{i-1})} := \Big\{ &t\text{-th element in } \mathrm{SUM}^{-1}(T_i - T_{i-1}) : T_i \in T_{i-1} + l/2 \pm \sqrt{l \log 1/\epsilon}, \\ &t \in \left[\sum_{a=1}^{j-1} 2^{e_{i,a}} \mathbf{1}_{X_{i,a}}(T_{i-1}) \mathbf{1}_{Y_{i,a}}(T_i) + 1, \sum_{a=1}^{j} 2^{e_{i,a}} \mathbf{1}_{X_{i,a}}(T_{i-1}) \mathbf{1}_{Y_{i,a}}(T_i) \right] \Big\} \end{aligned}$$

for $1 \le j \le r_i$.

Now let $\mathcal{K}_{i,0}^{(T_{i-1})} = [2^w + \sigma] \setminus \bigcup_{j=1}^{r_i} \mathcal{K}_{i,j}^{(T_{i-1})}$. By definition, for all k_i such that $\mathrm{SUM}(k_i) \notin l/2 \pm \sqrt{l \log 1/\epsilon}$, we have $k_i \in \mathcal{K}_{i,0}^{(T_{i-1})}$ (since $\mathrm{SUM}(k_i) = T_i - T_{i-1}$).

To bound its size, we first observe that since $\sigma > n2^{w/2}$ and $n > l \ge 2^{w^{2/3}/2} - 1$, by Lemma 8, for $T_{i-1} \in (i-1)(l/2 \pm \sqrt{l \log 1/\epsilon})$, we have

$$\sum_{T_i} E(T_{i-1}, T_i) \le \sigma/4 + w^{O(1)} 2^{w/2} \le \sigma/2.$$

Hence,

$$\begin{aligned} |\mathcal{K}_{i,0}^{(T_{i-1})}| &= 2^w + \sigma - \sum_{j=1}^{r_i} |\mathcal{K}_{i,j}^{(T_{i-1})}| \\ &= 2^w + \sigma - \sum_{j=1}^{r_i} \sum_{T_i \in T_{i-1} + l/2 \pm \sqrt{l \log 1/\epsilon}} 2^{e_{i,j}} \mathbf{1}_{X_{i,j}}(T_{i-1}) \mathbf{1}_{Y_{i,j}}(T_i), \end{aligned}$$

which by Equation (3), is equal to

$$= 2^w + \sigma - \sum_{T_i - T_{i-1} \in l/2 \pm \sqrt{l \log 1/\epsilon}} \left(\binom{l}{T_i - T_{i-1}} \cdot 2^{-l+w} - E_i(T_{i-1}, T_i) \right)$$

$$
\begin{aligned}
&\le 2^w + \sigma - \sum_{s_i \in l/2 \pm \sqrt{l \log 1/\epsilon}} \binom{l}{s_i} \cdot 2^{-l+w} + \sigma/2 \\
&= 3\sigma/2 + \sum_{s_i \notin l/2 \pm \sqrt{l \log 1/\epsilon}} \binom{l}{s_i} \cdot 2^{-l+w} \\
&\le 3\sigma/2 + 2^{w+1}\epsilon \\
&= 2\sigma.
\end{aligned}
$$

□

4.2 Approximating binomial coefficients

In this subsection, we prove Lemma 8. We begin by approximating binomial coefficients in a small range.

Lemma 9. *For any large integers l and d, $0 < \alpha \le \frac{1}{2}$, such that $d \le C\sqrt{\alpha l}$, there is a polynomial P of degree d^2, such that*

$$
\binom{l}{\alpha l + x + y} \cdot (1-\epsilon) \le \binom{l}{\alpha l} \cdot \left(\frac{1-\alpha}{\alpha}\right)^{x+y} \cdot P(x+y) \le \binom{l}{\alpha l + x + y}
$$

for all integers $x, y \in [0, C \cdot \sqrt{\alpha l}]$, a (small) universal constant $C > 0$, and $\epsilon = 2^{-d+8}$.

Moreover, $P(x+y)$ can be written as a sum of $d^4 + d^2 + 1$ nonnegative products:

$$
P(x+y) = \sum_{i=1}^{d^4+d^2+1} Q_i(x) R_i(y),
$$

where $Q_i(x), R_i(y) \ge 0$ for $x, y \in [0, C \cdot \sqrt{\alpha l}]$.

Proof. First observe that for $t \in [0, 2C \cdot \sqrt{\alpha l}]$, we have

$$
\begin{aligned}
\binom{l}{\alpha l + t} &= \binom{l}{\alpha l} \cdot \prod_{i=1}^{t} \frac{(1-\alpha)l - (i-1)}{\alpha l + i} \\
&= \binom{l}{\alpha l} \cdot \left(\frac{1-\alpha}{\alpha}\right)^t \cdot \prod_{i=1}^{t} \frac{1 - \frac{i-1}{(1-\alpha)l}}{1 + \frac{i}{\alpha l}}. \qquad (7)
\end{aligned}
$$

It suffices to approximate the last factor $f(t) := \prod_{i=1}^{t} \frac{1 - \frac{i-1}{(1-\alpha)l}}{1 + \frac{i}{\alpha l}}$.

For $|z| < 1$, $1 - z = e^{-\sum_{j \ge 1} j^{-1} \cdot z^j}$. Taking only the first $d-1$ terms, $e^{-\sum_{j=1}^{d-1} j^{-1} \cdot z^j}$, introduces a multiplicative error of

$$
e^{|\sum_{j \ge d} j^{-1} z^j|} \le e^{\frac{1}{d(1-|z|)}|z|^d}.
$$

Applying the above approximation to $1 - \frac{i-1}{(1-\alpha)l}$ and $1 + \frac{i}{\alpha l}$ in (7), we have

$$
f(t) \approx \exp\left(-\sum_{i=1}^{t} \sum_{j=1}^{d-1} \frac{1}{j} \cdot \left(\frac{i-1}{(1-\alpha)l}\right)^j + \sum_{i=1}^{t} \sum_{j=1}^{d-1} \frac{1}{j} \cdot \left(-\frac{i}{\alpha l}\right)^j \right)
$$

$$
\begin{aligned}
&= \exp\left(\sum_{j=1}^{d-1} \frac{1}{j} \cdot \sum_{i=1}^{t} \left(-(i-1)^j \cdot ((1-\alpha)l)^{-j} + i^j \cdot (-\alpha l)^{-j}\right)\right) \\
&= \exp\left(\sum_{j=1}^{d-1} \frac{1}{j} \cdot \left(-S_j(t-1) \cdot ((1-\alpha)l)^{-j} + S_j(t) \cdot (-\alpha l)^{-j}\right)\right) \qquad (8) \\
&=: \exp\left(P_1(t)\right),
\end{aligned}
$$

where $S_j(t) = 1^j + 2^j + \cdots + t^j$ is a degree-$(j+1)$ polynomial of t. The approximation above has a total multiplicative error of at most

$$
\exp\left(\frac{2}{d}\sum_{i=1}^{t}\left(\left(\frac{i-1}{(1-\alpha)l}\right)^d + \left(\frac{i}{\alpha l}\right)^d\right)\right) \le \exp\left(\frac{4t}{d} \cdot \left(\frac{t}{\alpha l}\right)^d\right),
$$

since $\alpha \le 1/2$. When $t \le 2C \cdot \sqrt{\alpha l} \le \sqrt{\alpha l}$, this is at most

$$
\exp\left(\frac{4}{d(\alpha l)^{(d-1)/2}}\right) \le 1 + (\alpha l)^{-(d-1)/2}
$$

for large d.

The exponent in (8) is a degree-d polynomial of t, which we denote by $P_1(t)$. We then apply $e^z = 1 + \sum_{i \ge 1} z^i/i!$ to (8). Note that taking only the first d terms in the sum introduces an *additive* error of

$$
\left|\sum_{i \ge d+1} z^i/i!\right| \le \frac{2|z|^{d+1}}{(d+1)!} \le 2^{-d}
$$

as long as $|z| \le d/6$. The exponent in (8), $P_1(t)$, is bounded by constants: Since $S_j(t) \le t^{j+1}$, and

$$
\begin{aligned}
|P_1(t)| &= \left|\sum_{j=1}^{d} \frac{1}{j} \cdot \left(-S_j(t-1) \cdot ((1-\alpha)l)^{-j} + S_j(t) \cdot (-\alpha l)^{-j}\right)\right| \\
&\le \sum_{j=1}^{d} t^{j+1} \cdot ((1-\alpha)l)^{-j} + (\alpha l)^{-j}) \\
&\le 2t \cdot \sum_{j \ge 1} (t/\alpha l)^j \\
&\le 4t^2/\alpha l \\
&\le 4. \qquad (9)
\end{aligned}
$$

Hence for large d, this approximation produces a degree-d^2 polynomial $P_2(t)$, such that

$$
f(t) \cdot (1 - (\alpha l)^{-(d-1)/2}) - 2^{-d} \le P_2(t) \le f(t) \cdot (1 + (\alpha l)^{-(d-1)/2}) + 2^{-d}.
$$

By a similar argument to (9), we have $e^{-4} \le f(t) \le 1$. The additive errors can be translated into multiplicative errors,

$$
f(t) \cdot (1 - (\alpha l)^{-(d-1)/2} - e^4 \cdot 2^{-d}) \le P_2(t) \le f(t) \cdot (1 + (\alpha l)^{-(d-1)/2} + e^4 \cdot 2^{-d}).
$$

By setting $P(t) := (1 - \epsilon/2)P_2(t)$, we prove the first half of the lemma.

To prove the second half, the following equation is applied to transform a negative monomial $-x^a y^b$ (plus a large positive constant) to a sum of two nonnegative terms,

$$M^{a+b} - x^a y^b \equiv \frac{1}{2}(M^a - x^a)(M^b + y^b) + \frac{1}{2}(M^a + x^a)(M^b - y^b). \tag{10}$$

Both terms are nonnegative when $x, y \in [0, M]$. It suffices to prove that the constant term of P (or equivalently P_2) is large enough to accomplish this transformation simultaneously for all negative monomials. In particular, by expanding $P_2(x + y) = \sum_{a,b\geq 0} \beta_{a,b} x^a y^b$, it suffices to show

$$\beta_{0,0} \geq \sum_{a,b\geq 0, a+b\neq 0} |\beta_{a,b}|(C\sqrt{\alpha l})^{a+b}. \tag{11}$$

Since if (11) does hold, we will be able to rewrite

$$P_2(x+y) = \left(\beta_{0,0} - \sum_{a,b\geq 0,a+b\neq 0} |\beta_{a,b}|(C\sqrt{\alpha l})^{a+b}\right) + \sum_{a,b\geq 0,a+b\neq 0} |\beta_{a,b}| \cdot \left((C\sqrt{\alpha l})^{a+b} + \mathrm{sgn}(\beta_{a,b}) x^a y^b\right)$$

and apply (10) to the later terms with $\beta_{a,b} < 0$. The number of terms will be at most $d^4 + d^2 + 1$, which will prove the second half of the lemma.

In order to prove (11), let us first bound the coefficients of $S_j(t)$. By Faulhaber's formula (or Bernoulli's formula),

$$1^j + 2^j + \cdots + t^j = S_j(t) = \frac{1}{j+1} \sum_{k=0}^{j} \binom{j+1}{k} \cdot B_k \cdot t^{j-k+1},$$

where B_k are the Bernoulli numbers: $B_0 = 1$; $B_1 = 1/2$; $B_k = 0$ for all odd integers $k > 1$; and for all even integers $k \geq 2$,

$$B_k = (-1)^{k/2+1} \frac{2 \cdot k!}{(2\pi)^k} \zeta(k),$$

where $\zeta(z) = \sum_{m=1}^{\infty} m^{-z}$ is the Riemann zeta function [AWH05].

The constant term in P_1 is 0, since both $S_j(0) = 0$ and $S_j(-1) = (-1)^{j+1}(S_j(1) - 1) = 0$. Hence, by definition,

$$\beta_{0,0} = P_2(0) = 1. \tag{12}$$

To upper bound the RHS of (11), consider the following operator: For polynomial $Q(x) = \sum_i \beta_i x^i$,

$$F(Q, z) := \sum_i |\beta_i| z^i.$$

Then for $z > 0$,

$$\begin{aligned} F(S_j, z) &= \frac{1}{j+1} \sum_{k=0}^{j} \binom{j+1}{k} \cdot |B_k| \cdot z^{j-k+1} \\ &\leq \sum_{k=0}^{j} \frac{2(j+1)^{k-1}}{(2\pi)^k} \cdot |\zeta(k)| \cdot z^{j-k+1}. \end{aligned}$$

Since $\lim_{k\to+\infty}\zeta(k)=1$, it is at most

$$F(S_j,z)\le\frac{z^{j+1}}{64C}\sum_{k=0}^{j}\frac{(j+1)^{k-1}}{(2\pi z)^k}.$$

for sufficiently small constant $C>0$. When $j+1\le 2\pi z$, it is at most $z^{j+1}/64C$. Therefore, by the fact that $j+1\le d\le C\sqrt{\alpha l}$,

$$\begin{aligned}
F(P_1,2C\sqrt{\alpha l}) &\le \sum_{j=1}^{d-1}\frac{1}{j}\cdot\left(F(S_j,2C\sqrt{\alpha l}+1)\cdot((1-\alpha)l)^{-j}+F(S_j,2C\sqrt{\alpha l})\cdot(\alpha l)^{-j}\right)\\
&\le \sum_{j=1}^{d-1}\frac{1}{64Cj}\left((2C\sqrt{\alpha l}+1)^{j+1}\cdot((1-\alpha)l)^{-j}+(\alpha l)^{-j}\right)\\
&\le \sum_{j=1}^{d-1}\frac{1}{32C}\left((4C\sqrt{\alpha l})^{j+1}\cdot(\alpha l)^{-j}\right)\\
&\le \frac{\sqrt{\alpha l}}{8}\sum_{j\ge 1}(4C/\sqrt{\alpha l})^{j}\\
&\le C.
\end{aligned}$$

Thus, the RHS of (11) is at most

$$\sum_{i=1}^{d}F(P_1,2C\sqrt{\alpha l})^i/i!\le e^{F(P_1,2C\sqrt{\alpha l})}-1\le e^C-1.$$

Together with (12), Equation (11) must hold for sufficiently small $C>0$. It completes the proof of the lemma. □

The above approximation can be extended to much larger ranges. Let C be the constant in Lemma 9, we have the following.

Lemma 10. *For any large even integer l, positive numbers M_x, M_y and ϵ, such that $l>8M_x, l>8M_y$ and $\epsilon>2^{-C\sqrt{l}/2+8}$, we can write $\binom{l}{l/2+x+y}\cdot 2^{-l}$ as*

$$\binom{l}{l/2+x+y}\cdot 2^{-l}=E(x,y)+\sum_{i=1}^{r}\tilde{Q}_i(x)\tilde{R}_i(y)$$

for all integers $x\in[-M_x,M_x]$ and $y\in[-M_y,M_y]$, such that

a) $E(x,y)\ge 0$ and for every integer $x\in[-M_x,M_x]$, $\sum_{y=-M_y}^{M_y}E(x,y)\le\epsilon$;

b) for all $i\in[r]$ and $x\in[-M_x,M_x], y\in[-M_y,M_y]$, $0\le\tilde{Q}_i(x),\tilde{R}_i(y)\le 1$;

c) $r\le O((M_xM_y/l)\log^4 1/\epsilon)$.

Proof. The idea is to first partition the domain $[-M_x, M_x] \times [-M_y, M_y]$ into rectangles, so that in each rectangle $[a_x, b_x] \times [a_y, b_y]$, we have

$$\binom{l}{l/2+x+y} = \binom{l}{(l/2+a_x+a_y)+(x-a_x)+(y-a_y)}$$

and

$$\binom{l}{l/2+x+y} = \binom{l}{(l/2-b_x-b_y)+(b_x-x)+(b_y-y)},$$

which allows us to apply Lemma 9 as long as both sides of the rectangle are not too large. Finally, we put all error terms from these rectangles into $E(x, y)$. In the following, we present this construction in detail.

Approximation in subrectangles. We partition $[-M_x, M_x] \times [-M_y, M_y]$ into $O(M_x M_y/l)$ rectangles of size at most $C\sqrt{l}/2 \times C\sqrt{l}/2$. For each rectangle $\mathcal{R}_k = [a_x^{(k)}, b_x^{(k)}] \times [a_y^{(k)}, b_y^{(k)}]$, if $a_x^{(k)} + a_y^{(k)} \le 0$, we apply Lemma 9 to approximate the binomial coefficient

$$\binom{l}{(l/2+a_x^{(k)}+a_y^{(k)})+(x-a_x^{(k)})+(y-a_y^{(k)})}$$

with $d = \log(2^8/\epsilon)$. The premises of the lemma are satisfied:

- $\alpha = 1/2 + a_x^{(k)}/l + a_y^{(k)}/l \le 1/2$;
- since $l/2 + a_x + a_y \ge l/2 - M_x - M_y \ge l/4$, we have $\alpha \ge 1/4$ and $C\sqrt{l}/2 \le C\sqrt{\alpha l}$, i.e., $(x - a_x^{(k)}), (y - a_y^{(k)}) \in [0, C\sqrt{\alpha l}]$;
- $d = \log(2^8/\epsilon) \le C\sqrt{l}/2 \le C\sqrt{\alpha l}$.

Otherwise, $b_x^{(k)} + b_y^{(k)}$ must be greater than zero, and we apply Lemma 9 to approximate

$$\binom{l}{(l/2-b_x^{(k)}-b_y^{(k)})+(b_x^{(k)}-x)+(b_y^{(k)}-y)},$$

which is equal to $\binom{l}{l/2-x-y} = \binom{l}{l/2+x+y}$. Similarly, one can verify in this case, all premises of the lemma are also satisfied.

Therefore, we obtain an approximation of $\binom{l}{l/2+x+y}$ in rectangle $\mathcal{R}_k$,

$$\sum_{i=1}^{r_k} \left(\binom{l}{\alpha l} \cdot \left(\frac{1-\alpha}{\alpha}\right)^x Q_i^{(k)}(x) \right) \cdot \left(\left(\frac{1-\alpha}{\alpha}\right)^y R_i^{(k)}(y) \right),$$

where $\alpha l = l/2 + a_x^{(k)} + a_y^{(k)}$ (or $l/2 - b_x^{(k)} - b_y^{(k)}$, depending on their values), and $r_k = O(d^4) = O(\log^4 1/\epsilon)$. Next we make both Q and R bounded by 1 and vanish outside $\mathcal{R}_k$, by scaling them by the following factor and multiplying by the indicator functions of the two sides correspondingly. Let

$$C_i^{(k)} = \max_{y \in [a_y^{(k)}, b_y^{(k)}]} \left(\frac{1-\alpha}{\alpha}\right)^y R_i^{(k)}(y),$$

and let

$$\tilde{Q}_i^{(k)}(x) := C_i^{(k)} \cdot 2^{-l} \binom{l}{\alpha l} \left(\frac{1-\alpha}{\alpha}\right)^x Q_i^{(k)}(x) \cdot \mathbf{1}_{[a_x^{(k)}, b_x^{(k)}]}(x)$$

and

$$\tilde{R}_i^{(k)}(y) := \frac{1}{C_i^{(k)}} \left(\frac{1-\alpha}{\alpha}\right)^y R_i^{(k)}(y) \cdot \mathbf{1}_{[a_y^{(k)}, b_y^{(k)}]}(y),$$

where $\mathbf{1}_S(z)$ is equal to 1 if $z \in S$, and 0 if $z \notin S$.

Now consider $W^{(k)}(x,y) := \sum_{i=1}^{r_k} \tilde{Q}_i^{(k)}(x) \cdot \tilde{R}_i^{(k)}(y)$, we have

i) for $(x,y) \in \mathcal{R}_k$, $2^{-l}\binom{l}{l/2+x+y}(1-\epsilon) \le W^{(k)}(x,y) \le 2^{-l}\binom{l}{l/2+x+y}$;

ii) for $(x,y) \notin \mathcal{R}_k$, $W^{(k)}(x,y) = 0$;

iii) $\tilde{Q}_i^{(k)}(x), \tilde{R}_i^{(k)}(y) \ge 0$;

iv) by definition, $\max_{y \in [a_y^{(k)}, b_y^{(k)}]} \tilde{R}_i^{(k)}(y) = 1$, and

$$\begin{aligned}
\tilde{Q}_i^{(k)}(x) &\le \min_{y \in [a_y^{(k)}, b_y^{(k)}]} \left\{ W^{(k)}(x,y) / \tilde{R}_i^{(k)}(y) \right\} \\
&\le \left(\max_{y \in [a_y^{(k)}, b_y^{(k)}]} \tilde{R}_i^{(k)}(y) \right)^{-1} \\
&= 1.
\end{aligned}$$

Merging the subrectangles. Finally, let

$$W(x,y) := \sum_{k \le O(M_x M_y/l)} W^{(k)}(x,y) = \sum_{k \le O(M_x M_y/l)} \sum_{i=1}^{r_k} \tilde{Q}_i^{(k)}(x) \tilde{R}_i^{(k)}(y)$$

and

$$E(x,y) := \binom{l}{l/2+x+y} \cdot 2^{-l} - W(x,y).$$

In the following, we show this construction indeed has the claimed properties:

a) by Item i) and ii) above and the fact that $\{\mathcal{R}_k\}$ is a partitioning of $[-M_x, M_x] \times [-M_y, M_y]$, we have $E(x,y) \ge 0$; for every integer $x \in [-M_x, M_x]$, we have

$$\sum_{y=-M_y}^{M_y} E(x,y) \le \sum_{y=-M_y}^{M_y} \epsilon 2^{-l} \binom{l}{l/2+x+y} \le \epsilon;$$

b) by Item iii) and iv) above, we have $0 \le \tilde{Q}_i^{(k)}(x), \tilde{R}_i^{(k)}(x) \le 1$;

c) $r = \sum_{k \le O(M_x M_y/l)} r_k = O((M_x M_y/l) \log^4 1/\epsilon)$.

This proves the lemma.

□

Finally, we are ready to prove Lemma 8.

Lemma 8 (restated). *For any large even integer l, positive numbers M_x, M_y and ϵ, such that $l > 8M_x$, $l > 8M_y$ and $\epsilon > 2^{-C\sqrt{l}/2+8}$, we have*

$$\binom{l}{l/2+x+y} \cdot 2^{-l+w} = E(x,y) + \sum_{i=1}^{r} 2^{e_i} \mathbf{1}_{X_i}(x)\mathbf{1}_{Y_i}(y)$$

for all integers $x \in [-M_x, M_x]$ and $y \in [-M_y, M_y]$, such that

a) $E(x,y) \geq 0$ and for every $x \in [-M_x, M_x]$, $\sum_{y=-M_y}^{M_y} E(x,y) \leq \epsilon 2^w + 4rM_y 2^{w/2}$;

b) for every $i \in [r]$, $e_i \geq 0$ is an integer, $X_i \subseteq [-M_x, M_x]$, $Y_i \subseteq [-M_y, M_y]$ are sets of integers;

c) $r \leq O((M_x M_y/l)w^2 \log^4(1/\epsilon))$.

Proof. By Lemma 10 and multiplying both sides by 2^w, we have

$$\begin{aligned}\binom{l}{l/2+x+y} \cdot 2^{-l+w} &= E_0(x,y)2^w + \sum_{i=1}^{r_0} (2^{w/2}\tilde{Q}_i(x)) \cdot (2^{w/2}\tilde{R}_i(y)) \\ &= E(x,y) + \sum_{i=1}^{r_0} \lfloor 2^{w/2}\tilde{Q}_i(x) \rfloor \cdot \lfloor 2^{w/2}\tilde{R}_i(y) \rfloor, \end{aligned} \tag{13}$$

for $E(x,y) \geq 0$ and

$$E(x,y) \leq E_0(x,y)2^w + \sum_{i=1}^{r_0} \left(2^{w/2}\tilde{Q}_i(x) + 2^{w/2}\tilde{R}_i(y)\right) \leq E_0(x,y)2^w + 2r_0 2^{w/2}.$$

Therefore, we have $\sum_{y=-M_y}^{M_y} E(x,y) \leq \epsilon 2^w + 4r_0 M_y 2^{w/2}$ for every $x \in [-M_x, M_x]$. It proves Item a).

Next, we write each $\lfloor 2^{w/2}\tilde{Q}_i(x) \rfloor$ and $\lfloor 2^{w/2}\tilde{R}_i(y) \rfloor$ in binary representation. Let

$$\lfloor 2^{w/2}\tilde{Q}_i(x) \rfloor = \sum_{j=0}^{w/2-1} 2^j \cdot \mathbf{1}_{X_{i,j}}(x)$$

and

$$\lfloor 2^{w/2}\tilde{R}_i(y) \rfloor = \sum_{j=0}^{w/2-1} 2^j \cdot \mathbf{1}_{Y_{i,j}}(y),$$

where $X_{i,j} \ni x$ (resp. $Y_{i,j} \ni y$) if and only $\lfloor 2^{w/2}\tilde{Q}_i(x) \rfloor$ (resp. $\lfloor 2^{w/2}\tilde{R}_i(y) \rfloor$) has a "1" in the j-th bit in its binary representation.

By expanding Equation (13), we have

$$\binom{l}{l/2+x+y} \cdot 2^{-l+w} = E(x,y) + \sum_{i=1}^{r_0} \sum_{j_1=0}^{w/2-1} \sum_{j_2=0}^{w/2-1} 2^{j_1+j_2} \mathbf{1}_{X_{i,j_1}}(x)\mathbf{1}_{Y_{i,j_2}}(y).$$

Hence, $r = O(r_0 w^2) \leq O((M_x M_y/l)w^2 \log^4 1/\epsilon)$.

□

4.3 Using standard word operations

The above data structure assumes that the computational model allows one to compute arbitrary functions on $O(w)$-bit input (that are hard-wired in the solution) in constant time. To only use standard word operations, one nature idea is to precompute all such functions needed at preprocessing time, and store a lookup table in memory.

By examining the data structure, one may verify that the only part that uses non-standard word operations is Lemma 2, combining the B spillovers. This subroutine is applied at t different levels, for $\sigma = (34B)^i \cdot n2^{w/2}$ ($i = 0, \ldots, t-1$). In each application, there are $2B$ different possible queries (decoding one of $k_1, \ldots, k_B$ or $T_1, \ldots, T_B$). The query time is bounded by a universal constant c_q. Hence, the whole query algorithm, which is a decision tree, can be encoded by a lookup table of size

$$t \cdot 2B \cdot \sum_{k=0}^{c_q} 2^{kw} \cdot O(w) = O(tB2^{c_q w} \cdot w)$$

bits.

Since w is required to be at least $7 \log n$, storing the entire lookup table is unaffordable. However, one may use a standard trick to decrease w for self-reducible problems. Given an input of n bits, we evenly partition the input into blocks of size $n' = n^{1/8c_q}$. For each block, we apply Theorem 1 with $w' = \frac{7 \log n}{8c_q}$, and construct a data structure using

$$n' + \lceil \frac{n'}{w'^{\Omega(t)}} \rceil \leq n' + \frac{n'}{(\log n)^{\Omega(t)}} + 1$$

bits. In addition, we also store the lookup table for the whole query algorithm using

$$O(tB2^{c_q w'} \cdot w') = \tilde{O}(n^{7/8})$$

bits. Note that this lookup table is *shared* among all blocks, and hence only one copy needs to be stored. Finally, the total space usage is

$$n + \frac{n}{(\log n)^{\Omega(t)}} + O(n^{1-1/8c_q}).$$

This proves Theorem 2.

Theorem 2 (restated). *Given a 0-1 array of length n for sufficiently large n, for any $t \geq 1$, one can construct a succinct data structure using*

$$n + \frac{n}{(\log n)^{\Omega(t)}} + n^{1-c}$$

bits of memory supporting rank queries in $O(t)$ time, in a word RAM with word-size $w = \Theta(\log n)$, for some universal constant $c > 0$.

5 Discussion and Open Questions

The sibling of the rank query is *select*, which asks "where is the k-th one in the input array?" Most previous succinct rank data structures support both rank and select at the same time, and the lower bounds [PV10] also apply to select queries. It is a natural question to ask whether our data structure also generalizes to both rank and select.

Several previous solutions also apply to sparse inputs (e.g., [Pag01], [GRR08], [Păt08], etc). That is, we are given an array of n bits with m ones, and would like to design a data structure with space close to $\log \binom{n}{m}$. However, generalizing our data structure to such instances seems more complicated, may require massive technical manipulation.

Most cell-probe lower bound proof techniques also apply to the *information cell-probe model*, defined in Section 3. However, some data structure problems are trivial in the information cell-probe model, while their complexities in the cell-probe model are unclear (e.g., succinct dictionary). If one believes such a problem does not admit trivial cell-probe data structures, then proving any lower bound requires distinguishing the two models, and likely needs a new technique. One might hope to apply the breakthrough in information and communication [GKR14] to achieve such separation. Despite the close connections between data structures and communication complexity, separating cell-probe and information cell-probe is not entirely equivalent to separating information and communication complexity. One reason is that only the memory side changes the measure from worst-case communication to information, while the data structure side still sends a message of fixed length. It is unclear to us whether similar separations can be established in communication complexity.

Acknowledgments The author would like to thank Jiantao Jiao and Tengyu Ma for helpful discussions.

References

[AGKM16] Sanjeev Arora, Rong Ge, Ravi Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization - provably. *SIAM J. Comput.*, 45(4):1582–1611, 2016.

[AWH05] G.B. Arfken, H.J. Weber, and F.E. Harris. *Mathematical Methods for Physicists*. Elsevier, 2005.

[Cla97] David R. Clark. *Compact PAT trees*. PhD thesis, 1997.

[CM96] David R. Clark and J. Ian Munro. Efficient suffix trees on secondary storage (extended abstract). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA.*, pages 383–391, 1996.

[DPT10] Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, pages 593–602, 2010.

[GGG+07] Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 371–382, 2007.

[GKR14] Anat Ganor, Gillat Kol, and Ran Raz. Exponential separation of information and communication. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 176–185, 2014.

[GM07] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. *Theor. Comput. Sci.*, 379(3):405–417, 2007.

[GMR06] Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 368–373, 2006.

[Gol07a] Alexander Golynski. Optimal lower bounds for rank and select indexes. *Theor. Comput. Sci.*, 387(3):348–359, 2007.

[Gol07b] Alexander Golynski. *Upper and Lower Bounds for Text Indexing Data Structures*. PhD thesis, University of Waterloo, Ontario, Canada, 2007.

[GRR08] Alexander Golynski, Rajeev Raman, and S. Srinivasa Rao. On the redundancy of succinct data structures. In *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings*, pages 148–159, 2008.

[Jac89] Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554, 1989.

[Mil05] Peter Bro Miltersen. Lower bounds on the size of selection and rank indexes. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 11–12, 2005.

[Moi16] Ankur Moitra. An almost optimal algorithm for computing nonnegative rank. *SIAM J. Comput.*, 45(1):156–173, 2016.

[MRR98] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. In *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings*, pages 186–196, 1998.

[MRRR03] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, pages 345–356, 2003.

[Mun96] J. Ian Munro. Tables. In *Foundations of Software Technology and Theoretical Computer Science, 16th Conference, Hyderabad, India, December 18-20, 1996, Proceedings*, pages 37–42, 1996.

[Pag01] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31(2):353–363, 2001.

[Păt08] Mihai Pătraşcu. Succincter. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 305–313, 2008.

[PV10] Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 117–122, 2010.

[RRR02] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 233–242, 2002.

[Yao78] Andrew Chi-Chih Yao. Should tables be sorted? (extended abstract). In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 22–27, 1978.

A Proof of Lemma 6

Observe that given two elements $x_i \in [M_i]$ and $x_j \in [M_j]$, we may treat the pair as a number from $[M_i \cdot M_j]$ by mapping $(x_i, x_j) \mapsto x_i M_j + x_j$. Since initially each $M_i \leq 2^w$, by repeatedly merging elements in this way, we may assume the universe size of each element is between 2^{3w} and 2^{6w}, except for the last element, which may be from a set of size smaller than 2^{3w}.

Now the task becomes to encode a sequence $(y_1, \ldots, y_{B'}) \in [N_1] \times \cdots \times [N_{B'}]$ such that

- for $i = 1, \ldots, B' - 1$, $2^{3w} < N_i \leq 2^{6w}$, and
- $N_{B'} \leq 2^{6w}$.

We first inductively calculate and hard-wire the following constants:

$$V_0 = 1$$

and for $i \geq 1$,

$$w_i = \lfloor \log(V_{i-1}) + 1.5w \rfloor \tag{14}$$

$$U_i = \lfloor \frac{2^{w_i}}{V_{i-1}} \rfloor \tag{15}$$

$$V_i = \lceil \frac{N_i}{U_i} \rceil \tag{16}$$

Then we break each element $y_i \in [N_i]$ into a pair (u_i, v_i) such that $u_i \in [U_i]$ and $v_i \in [V_i]$ due to (16), e.g., $u_i := \lfloor y_i / V_i \rfloor$ and $v_i = y_i \bmod V_i$. Then for $1 \leq i \leq B' - 1$, we combine (v_{i-1}, u_i) into an element $z_i \in [2^{w_i}]$ due to (15), e.g., $z_i = v_{i-1} \cdot U_i + u_i$.

The data structure will explicitly store $z_1, \ldots, z_{B'-1}$ using $w_1 + \cdots + w_{B'-1}$ bits. For the last two elements $(v_{B'-1}, y_{B'})$, we combine them into an element $z_{B'} \in [V_{B'-1} \cdot N_{B'}]$, and store $z_{B'}$ using $m - (w_1 + \cdots + w_{B'-1})$ bits and a spillover of size

$$K = \lceil V_{B'-1} \cdot N_{B'} \cdot 2^{(w_1 + \cdots + w_{B'-1}) - m} \rceil.$$

To decode a y_i, it suffices to retrieve z_i and z_{i+1}, which determines u_i and v_i respectively. Since $w_i = O(w)$, it takes constant time.

The data structure uses m bits of memory. It remains to show that K is at most $\left\lceil 2^{-m} \cdot \prod_{i=1}^{B} M_i \right\rceil + 1$. By Equation (14), (15) and (16), we have

$$U_i = \Theta(2^{w_i}/V_i) = \Theta(2^{1.5w}),$$

and

$$V_i = \Theta(N_i/U_i) \geq \Omega(2^{1.5w}).$$

Thus, by (15) again, we have

$$2^{w_i} \leq (U_i + 1)V_{i-1} \leq U_i V_{i-1} \cdot (1 + O(2^{-1.5w})),$$

and by (16), we have

$$U_i V_i \leq N_i \cdot (1 + O(2^{-1.5w})).$$

Finally, by definition, we have

$$\begin{aligned}
K &= \lceil 2^{-m} \cdot V_{B'-1} N_{B'} \prod_{i=1}^{B'-1} 2^{w_i} \rceil \\
&\leq \lceil 2^{-m} \cdot V_{B'-1} N_{B'} \prod_{i=1}^{B'-1} U_i V_{i-1} \cdot (1 + O(2^{-1.5w})) \rceil \\
&\leq \lceil 2^{-m} \cdot N_{B'} \cdot (1 + O(B' \cdot 2^{-1.5w})) \prod_{i=1}^{B'-1} U_i V_i \rceil \\
&\leq \lceil 2^{-m} \cdot (1 + O(B' \cdot 2^{-1.5w})) \prod_{i=1}^{B'} N_i \cdot (1 + O(2^{-1.5w})) \rceil \\
&\leq \lceil 2^{-m} \cdot (1 + O(B \cdot 2^{-1.5w})) \prod_{i=1}^{B} M_i \rceil.
\end{aligned}$$

The last inequality is because $\prod_{i=1}^{B} M_i = \prod_{i=1}^{B'} N_i$. Since $m \geq \sum_{i=1}^{B} \log M_i - w$ and $B = o(2^{w/2})$, we have

$$K \leq \lceil 2^{-m} \prod_{i=1}^{B} M_i + 1 \rceil = \lceil 2^{-m} \prod_{i=1}^{B} M_i \rceil + 1.$$

This proves the lemma.