Indexing Range Maximum-Sum Segment Queries with Offsets

Seungbum Jo
Chungnam National University, Daejeon, Republic of Korea

Dominik Köppl
University of Yamanashi, Kofu, Japan

Abstract

Given an array of n real numbers, the maximum segment sum (MSS) problem is to find a contiguous subarray that has the largest sum. While the MSS problem can be solved optimally with Kadane's algorithm in $O(n)$ time, the study of its indexing version spawned new extensions such as (a) retrieving the MSS after subtracting a query offset parameter for all array entries or (b) retrieving the MSS for arbitrary query ranges. We here study the combination of both problems (a) and (b), which requires retrieving the MSS for arbitrary query ranges after subtracting a query offset parameter for all array entries. For that, we present an index whose query time is only slower than the best known for (a) by a factor of $O(\log n)$. In detail, our index uses $O(n \log n)$ space, supports queries in $O(\log^2 n)$ time, and can be constructed in $O(n \log^3 n)$ time. As side results, we study our combined problem in the context of run-length compressed input, and also deduce a solution for (a) that works in run-length compressed space and time. Finally we give supportive lower bounds for our query problem, showing that there is only a polylogarithmic gap of improvement left.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases maximum segment sum, data structure, range query

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.23

Funding *Seungbum Jo*: Seungbum Jo was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-202523963814).
Dominik Köppl: This work was supported by JSPS KAKENHI Grant Number 25K21150.

1 Introduction

Given an array X of real numbers, the maximum-sum segment (MSS) problem is to find an interval $[i, j]$ within X that maximizes the sum of its elements, which we call an MSS of the sequence. This problem found applications, among others, in image processing [11], pattern recognition [19], and biological sequence analysis [26].

The textbook algorithm for solving the MSS problem is Kadane's algorithm, cf. [10, 4.4 Maximum Subarray Sum]. Since the various applications often require adaptations to specific problem variations, it is not surprising that the MSS problem has been studied in various settings, cf. [6] for an encyclopedic survey of MSS and its variations. Chen and Chao [7] considered an indexing problem, where it is allowed to preprocess X and then answer queries on intervals of X. For that task, they devised a linear-time constructible data structure that supports constant-time queries of an MSS for any query interval. They also showed that this problem is linearly equivalent to the range-minimum query (RMQ) problem, meaning that a data structure with construction time $t_c(n)$ and query time $t_q(n)$ solving the MSS problem can be transformed into a data structure solving RMQs with the time complexities $t_c(cn)$ and $t_q(cn)$ for some constant c, respectively. Gawrychowski and Nicholson [12] gave a $\Theta(n)$-bits data structure that also achieves constant query times in the setting that the answer is only the interval and not the maximum sum itself. They also show that $1.89113n - \Theta(\log n)$ bits are necessary for supporting this query type.

A maximal local MSS is a local MSS that is not a segment of any local MSS other than it, where a local MSS is a segment that has itself as its only MSS. Ruzzo and Tompa [21] gave an algorithm that finds all distinct maximal local MSSs in $O(n)$ time, and Sakai [22] designed a linear-time constructible data structure supporting constant-time queries of the maximal local MSS of any given segment that contains any given position.

Bengtsson and Chen [3] showed that an arbitrarily given number of non-overlapping segments that maximize the sum of all their elements can be found in linear time. Yu et al. [28] considered the MSS problem where each element of the input sequence is uncertain within a specific interval and proposed a linear-time algorithm for this problem. The density of a segment is defined as the mean of all elements in the segment. Cheng et al. [8] considered the MSS problem with the condition that the density of the segment to be found is between given lower and upper bounds. Bae [1] and Liu and Chao [17] considering returning k non-overlapping segments that maximize the sum of all their elements.

Another variant is the maximum-density segment query [13, 15, 16] that asks to find a segment of length between a given lower and upper bound that maximizes the density. Chung and Lu [9] showed that this query is solvable in linear time.

Another natural extension of the one-dimensional maximum segment sum problem is the maximum subarray problem, where the input is a two-dimensional array with length n in each dimension, and the goal is to find a contiguous rectangular subarray that has the largest sum. This problem has been addressed by Bentley [4], who gave an $O(n^3)$-time algorithm for the problem. Subsequently, the time complexity has been improved to $O(n^3\sqrt{\log\log n/\log n})$ by Takaoka [24] and further to $O(n^3 \log\log n/\log n)$ by Tamaki and Tokuyama [25]. They showed that the problem can be reducible to $(\min, +)$-matrix multiplication. Using this result, the current fastest algorithm runs in $n^3/2^{\Omega(\log n)^{1/2}}$ time by Chan and Williams [5], which also applies to higher-dimensional arrays. In addition, Fukuda et al. [11] introduced the maximum convex sum problem, which is a variant of the maximum subarray problem where the retrieved subarray must be convex (i.e., the intersection of the subarray with any row or column is connected). Here, Bae and Takaoka [2] gave an $O(n)$-time parallel algorithm solving the maximum convex sum problem for n^2 processors.

Our Contribution. Among the aforementioned variations, we focus on two indexing versions of the MSS problem, where we are allowed to preprocess the input array and then answer queries (a) on intervals of the input array, or (b) after subtracting a query offset parameter from all array entries. In particular, we are interested in the combinations of both problems (a) and (b), where we want to answer queries on intervals of the input array after subtracting a query offset parameter from all array entries. We present an indexing data structure solving this combined problem in Theorem 8. The main tool is to answer this problem for the special case that the output interval has to be a prefix or a suffix of the input array, and to reuse known results for the problem (b) to solve the general case on $O(\log n)$ segments. The results are summarized in Table 1. Subsequently, we focus on the special case of run-length compressed input, where we present a data structure in Corollary 15 for the combined problem that uses space linear in the size of the compressed input and answers queries in time polylogarithmic in the size of the compressed input. We can improve the query time for the plain array or in the run-length compressed setting if we allow only integer offsets (Corollary 11 and Remark 16). Finally, we conclude with lower bounds for our combined problem, which show that our data structures are optimal up to polylogarithmic factors (Corollary 19).

Table 1 Summary of best known complexities for the MSS query and its variants.

Query type	Space	Query time	Reference
MSS	$O(1)$	$O(n)$	Kadane's algorithm
OMSS	$O(n)$	$O(\log n)$	Sakai [23]
R-OMSS without offset	$O(n)$	$O(1)$	Chen et al. [7]
Suffix R-OMSS	$O(n)$	$O(\log n)$	Theorem 5
R-OMSS	$O(n \log n)$	$O(\log^2 n)$	Theorem 8

	1	2	3	4	5	6	7	8	9	10	11	12
X_0	4	-9	15	-22	7	-31	18	-5	27	-14	33	-26
C_0	4	-5	10	-12	-5	-36	-18	-23	4	-10	23	-3

Query type	Query Range	Offset	Answer Range	Answer
MSS	-	-	$[7, 11]$	59
SMSS	-	-	$[7, 12]$	33
OMSS	-	10	$[11, 11]$	23
Suffix R-OMSS	$[1, 5]$	1	$[5, 5]$	6
R-OMSS	$[1, 5]$	1	$[3, 3]$	14

Figure 1 Examples for the MSS query and its variants.

2 Preliminaries and Query Definitions

Our computation model is the RAM model. Let $\mathbb{R}$ denote the set of all real numbers. We write $[i, j] = \{i, i+1, i+2, \ldots, j\}$ for two integers $i \leq j$, and use the same notation $[i, j]$ for intervals of real numbers in case that $i, j \in \mathbb{R}$. We assume that all real numbers can be stored in a single word and that basic arithmetic operations on these numbers can be performed in constant time. (We also assume that there are no errors in computational precision due to rounding or overflow.)

Let $X_0[1..n]$ be the input array of real numbers. X_0 is subject to the following queries, for which we present examples in Figure 1. While the queries are initially stated to return the value of a specific subarray of X_0, it is beneficial to also return the interval of the subarray that gives the answer, which we call the *answer range*. Since we here mostly focus on data structures using $\Omega(n)$ space, it suffices to focus on finding answer ranges – we will shortly later see that the actual value can be obtained from an answer range with an $O(n)$-space array of prefix sums. We state our base problem as follows.

Maximum-Segment-Sum (MSS)
Input: array of real numbers $X_0[1..n]$.
Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[1,n]} \sum_{i=\ell}^{r} X_0[i]$.

We also study the special case that the answer range of the MSS problem is constrained to be a suffix of X_0.

Suffix Maximum-Segment-Sum (SMSS)
Input: array of real numbers $X_0[1..n]$
Output: value and integer ℓ achieving $\max_{\ell\in[1,n]} \sum_{i=\ell}^{n} X_0[i]$.

Sakai [23] introduced the following indexing problem that generalizes the MSS problem by allowing a query offset parameter α to be subtracted from all array entries before answering the query.

Offset Maximum-Segment-Sum (OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$.
Query Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[1,n]} \sum_{i=\ell}^{r}(X_0[i] - \alpha)$.

Defining the array $X_\alpha[1..n]$ with $X_\alpha[i] := X_0[i] - \alpha$, the OMSS query on X_0 with offset α is the MSS query on X_α. Hence, OMSS is a generalization of MSS, since the answer to an MSS query on X_0 is the same as the answer to an OMSS query on X_α with offset $\alpha = 0$. Sakai [23] presented an approach answering an OMSS query as follows.

▶ **Lemma 1** ([23]). *Given an array $X_0[1..n]$ of n real numbers, we can compute a list L of at most n (offset, interval)-pairs in $O(n \log^2 n)$ time using $O(n)$ space such that the solution to an OMSS query for a given offset α is the interval of the predecessor offset of α in L. By sorting L descendingly by the offsets, a binary search for the predecessor of α in L can return an answer range of an OMSS query in $O(\log n)$ time.*

The *predecessor* of α in L is the largest offset in L that is smaller than or equal to α. Three observations are in order regarding Lemma 1.

First, while the list of Lemma 1 does not explicitly return the MSS for a query offset α, we can compute the sum $\sum_{i=a}^{b}(X_0[i] - \alpha)$ for an entry $(\alpha, [a..b])$ in L in constant time if we have the array of prefix lengths $C_0[1..n]$ of X_0, where $C_0[i] := \sum_{k=1}^{i} X_0[k]$ for each $i \in [1, n]$. To see that, define the prefix sum of X_α for each $i \in [1, n]$ as $C_\alpha[i] := \sum_{k=1}^{i} X_\alpha[k] = C_0[i] - \alpha k$, and the sum of the subarray $X_\alpha[i..j]$ as $S_\alpha(i, j) = \sum_{k=i}^{j} X_\alpha[k] = C_0[j] - C_0[i-1] - \alpha(j-i+1)$ for any i and j with $1 \leq i \leq j \leq n$. Then $\sum_{i=a}^{b}(X_0[i] - \alpha) = S_\alpha(a, b)$.

Second, the upper bound $|L| \leq n$ is tight since L can reach the size n for the input $X_0 = [1, 2, 3, \ldots, n]$, where the OMSS query for $\alpha = k + \epsilon$ gives the range $[k + 1, n]$ for every $\epsilon \in (0, 1)$ and $k \in [0, n]$.

Finally, there can be multiple answer ranges for a query, in particular if the interval covers trailing zeros (after applying the offset). While implicit, the intervals of Lemma 1 do not contain any zero-sum suffix nor prefix.

Now we present our main problem of interest, which combines the OMSS and MSS queries on subarrays of X_0.

Range Offset Maximum-Segment-Sum (R-OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$ and a range $[a, b] \subseteq [1, n]$.
Query Output: value and answer range $[\ell, r]$ achieving $\max_{[\ell,r]\subseteq[a,b]} \sum_{i=\ell}^{r}(X_0[i] - \alpha)$.

Using the notation of $S_\alpha(i, j)$, the answer to an R-OMSS query on X_0 with query range $[a, b]$ and offset α can be computed as $\max_{[\ell,r]\subseteq[a,b]} S_\alpha(\ell, r)$. The R-OMSS is a generalization of the OMSS problem (with the query range $[1, n]$) and the range MSS problem introduced by Chen et al. [7], which asks for the MSS of a query range $[a, b]$ without the offset parameter (i.e., $\alpha = 0$). Like for MSS and its restriction SMSS, we can also define the suffix and prefix versions of R-OMSS queries, where the answer range is constrained to be a suffix or a prefix of the query range, respectively.

Suffix Range Offset Maximum-Segment-Sum (Suffix R-OMSS)
Construction Input: array of real numbers $X_0[1..n]$
Query Input: real-value offset $\alpha \geq 0$ and a range $[a, b] \subseteq [1, n]$.
Query Output: value and integer ℓ achieving $\max_{\ell\subseteq[a,b]} \sum_{i=\ell}^{b}(X_0[i] - \alpha)$.

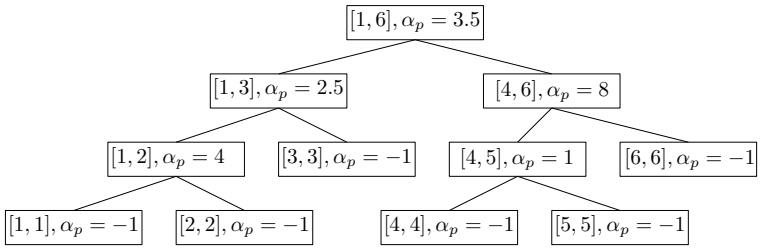

Figure 2 SBST on $X_0 = [4, 1, 6, 1, 8, 0]$. Each node is labeled with its corresponding canonical range and threshold offset.

A symmetric definition can be given for *prefix R-OMSS* queries, where the answer range is constrained to be a prefix of the query range.

3 Data structure for R-OMSS Queries

Our main goal of this paper is to devise data structures for answering R-OMSS queries on X_0, where a non-negative offset $\alpha \geq 0$ and a query range $[a, b]$ are given at query time. There are two naive approaches prioritizing either (1) space or (2) time.

(1) We can apply Kadane's algorithm on $X_\alpha[a..b]$. Since any value in X_α can be computed in $O(1)$ time from X_0, Kadane's algorithm answers R-OMSS in $O(n)$ time using $O(1)$ space additional to X_0.

(2) Alternatively, we can maintain the data structure of Lemma 1 for all $\Theta(n^2)$ possible query ranges. This results in an $O(n^3)$-space data structure with $O(\log n)$ query time.

In what follows, we present two non-trivial data structures for R-OMSS queries that finds its complexities between the solutions (1) and (2). We start with an $O(n)$-space data structure answering suffix R-OMSS queries on X_0 in $O(\log^2 n)$ time. By symmetry, the same data structure results can be obtained for prefix R-OMSS queries. Next, in Section 3.2, we consider a data structure for (general) R-OMSS queries. By combining the data structure of Lemma 1 with the suffix R-OMSS data structures, we obtain an $O(n \log n)$-space data structure that can answer an R-OMSS query in $O(\log^2 n)$ time.

3.1 Data Structure for Suffix R-OMSS Queries

We start with devising a data structure for range suffix OMSS queries. The main idea is to maintain the canonical ranges (defined below) of X_0 using a *suffix binary search tree (SBST)*. Each node in the tree stores a *threshold offset*. By comparing α with the offsets stored along a root-to-leaf path in the tree, as in a standard binary search tree, we can track the endpoint of the range corresponding to the query answer when the query range is canonical. Finally, we partition the query range $[a, b]$ into $O(\log n)$ canonical ranges and answer the query using a dynamic programming approach.

The canonical ranges of X_0 are the elements of a recursive binary partition of $[1, n]$ into $O(n)$ ranges, which can be represented by a binary tree. The *canonical ranges* of X_0 are defined as follows:

(1) $[1, n]$ is a canonical range, and

(2) for every canonical range $[g, h]$ with $g \leq h$, both $[g, g + \lfloor (h - g)/2 \rfloor]$ and $[g + \lfloor (h - g)/2 \rfloor + 1, h]$ are also canonical ranges.

Under this definition, there are $O(n)$ canonical ranges in total. We then define the SBST on $X_0[g, h]$ as follows:

- The root of the SBST on $X_0[g, h]$ is the canonical range $[g, h]$.
- If $g < h$, then the SBSTs on $X_0[g, g + \lfloor (h - g)/2 \rfloor]$ and $X_0[g + \lfloor (h - g)/2 \rfloor + 1, h]$ are the left and right child subtrees of the root, respectively. Otherwise, g=h; a leaf is added with canonical range [g, g].

See Figure 2 for an example. (The reader familiar with segment trees may notice that the SBST is a specialization of a segment tree indexing all integers in $[1, n]$.)

Let T denote the SBST on $X_0[1, n]$. From the definition, each canonical range of X_0 is a node in T, and vice versa. The leaves of T are all canonical ranges of size 1, i.e., $[i, i] = \{i\}$ for each $i \in [1, n]$. By construction, T is a binary tree with $O(n)$ nodes and height $O(\log n)$. Finally, we augment each node p of T with a *threshold offset* α_p. Before defining the threshold offsets, we introduce the following key property for answering SMSS queries.

▶ **Proposition 2.** *For any positive α and β with $\alpha < \beta$, the answer range of the SMSS query on X_β is a subrange of the answer range of the SMSS query on X_α.*

Proof. Let $[x, n]$ and $[x', n]$ be the answer ranges of the SMSS query on X_α and X_β, respectively. Suppose that $x' < x$. Since $S_\beta(x', n) > S_\beta(x, n)$, it follows that the sum $S_\beta(x', x - 1) > 0$, which implies $S_\alpha(x', x - 1) = S_\beta(x', x - 1) + (\beta - \alpha)(x - x') > 0$. This leads to a contradiction since $S_\alpha(x', n) = S_\alpha(x, n) + S_\alpha(x', x - 1) > S_\alpha(x, n)$. ◀

Let $[x_p, y_p]$ be the range corresponding to the node p, and let $[x_\alpha, y_p] \subseteq [x_p, y_p]$ be the answer range of the SMSS problem on $X_\alpha[x_p, y_p]$ for some variable α. We then define the threshold offset α_p at node p as the maximum value of α such that $x_\alpha \in [x_p, x_p + \lfloor (y_p - x_p)/2 \rfloor]$, that is, x_α lies in the range corresponding to the left child of p. If no such α exists, or if p is a leaf node, we define $\alpha_p = -1$. By Proposition 2, the value α_p is well defined as a single constant.

The data structure for suffix R-OMSS queries is composed of (1) the array C_0, and (2) the SBST T on X_0. Each node of the SBST stores its threshold offset and its canonical range. (1) and (2) together use $O(n)$ space in total.

We now describe how to answer a query using this data structure. We first consider the case where the query range is a canonical range $[x_p, y_p]$ corresponding to a node p in T. In this case, it suffices to find the position x such that $[x, y_p]$ is the answer range. To find x, we start from the node p and compare α with α_p. As in a standard binary search tree, we recursively perform the same comparison: we proceed to the left child if $\alpha \leq \alpha_p$, and to the right child otherwise, until we reach a leaf node corresponding to the canonical range $[x', x']$. Then $x' = x$ from the following proposition.

▶ **Proposition 3.** *Let p be an internal node in T with the corresponding range $[x_p, y_p]$, and let $z_p = x_p + \lfloor (y_p - x_p)/2 \rfloor$. If $\alpha \leq \alpha_p$ (resp. $\alpha > \alpha_p$), then the leftmost endpoints of the answer ranges of the SMSS queries on $X_\alpha[x_p, y_p]$ and $X_\alpha[x_p, z_p]$ (resp. on $X_\alpha[x_p, y_p]$ and $X_\alpha[z_p + 1, y_p]$) are the same.*

Proof. Here we consider only the case that $\alpha \leq \alpha_p$; the case $\alpha > \alpha_p$ can be proved analogously. Let x and x' be the leftmost endpoints of the answer ranges of the SMSS queries on $X_\alpha[x_p, y_p]$ and $X_\alpha[x_p, z_p]$. We now claim that $x = x'$. To prove the claim, suppose that $x < x'$ (the case $x > x'$ can be handled analogously). Since $S_\alpha(x, y_p) > S_\alpha(x', y_p)$, it follows that $S_\alpha(x, x' - 1) > 0$. Consequently, $S_\alpha(x, z_p) = S_\alpha(x', z_p) + S_\alpha(x, x' - 1) > S_\alpha(x', z_p)$, which contradicts the definition of x'. ◀

Since the height of T is $O(\log n)$, we can answer a suffix R-OMSS query on X_0 in $O(\log n)$ time when the query range is canonical.

Finally, we consider how to answer a suffix R-OMSS query on X_0 for a general query range $[a, b]$ using the data structure. We first partition $[a, b]$ into $c = O(\log n)$ distinct canonical ranges $[a_1, b_1], \ldots, [a_c, b_c]$, ordered from left to right (i.e., $a_1 = a$, $b_c = b$, and $b_k + 1 = a_{k+1}$ for all $k \in [c-1]$). This partition can be computed by starting from the root of T and traversing the tree downwards in two directions for finding the leaves of the canonical ranges $[a, a]$ and $[b, b]$, respectively. We then collect $O(\log n)$ canonical ranges corresponding to children on the two root-to-leaf paths. At each level of T, there are at most two nodes whose canonical ranges $[x, y]$ satisfy $[x, y] \cap [a, b] \neq \emptyset$ but $[x, y] \not\subseteq [a, b]$; only these nodes require recursion to the next level, while any other intersecting node is fully contained in $[a, b]$ and can be reported as a canonical range immediately. Therefore, we can compute these canonical ranges in $O(\log n)$ time.

If $c = 1$ (i.e., $[a, b]$ is a canonical range), we can answer the R-OMSS query in $O(\log n)$ time using T. Otherwise, we apply a dynamic programming approach as follows. Define an array $A[1..c]$ of size c, where for each $k \in [1, c]$, $A[k]$ stores the answer range of the SMSS query on $X_\alpha[a_1, b_k]$. Then $A[c]$ stores the answer of the suffix R-OMSS query.

To compute $A[c]$, we first handle the base case by computing $A[1]$ in $O(\log n)$ time using T. Next, for $k > 1$, we compute $A[k]$ from $A[k-1]$ as follows. To compute $A[k]$, it suffices to determine the answer range $[x, b_k]$ of the SMSS query on $X_\alpha[a_1, b_k]$. Now we introduce the following lemma, which is used to compute $A[k]$.

▶ **Lemma 4.** *Let $[x, b_k]$ and $[x', b_{k-1}]$ be the answer ranges of the SMSS queries on $X_\alpha[a_1, b_k]$ and $X_\alpha[a_1, b_{k-1}]$, respectively. Then x is either (1) x' or (2) z, where $[z, b_k]$ is the answer range of the SMSS query on $X_\alpha[a_k, b_k]$.*

Proof. If $x < x'$ or $x' < x \leq b_{k-1}$, then $S_\alpha(x, b_k) = S_\alpha(x, b_{k-1}) + S_\alpha(a_k, b_k) > S_\alpha(x', b_{k-1}) + S_\alpha(a_k, b_k)$, which contradicts the fact that $[x', b_{k-1}]$ is the answer range of $X_\alpha[a_1, b_{k-1}]$.

Similarly, if $a_k \leq x < z$ or $x > z$, then $S_\alpha(x, b_k) > S_\alpha(z, b_k)$, which contradicts the fact that $[z, b_k]$ is the answer range of $X_\alpha[a_k, b_k]$. ◀

The answer ranges of the SMSS queries on $X_\alpha[a_1, b_{k-1}]$ and $X_\alpha[a_k, b_k]$ can be obtained from $A[k-1]$ and from T, respectively, in $O(\log n)$ time. Thus, by Lemma 4, given $A[k-1]$, the value $A[k]$ can be computed in $O(\log n)$ time, which implies that $A[c]$ can be computed in $O(\log^2 n)$ time.

Construction time. As C_0 can be constructed in $O(n)$ time, it suffices to analyze the construction time of T. To construct T, for each node p whose corresponding range is $[x_p, y_p]$ with $\ell_p = y_p - x_p + 1$, we compute its threshold offset α_p as follows. Consider the point set $P = \{(t, S_0(y_p - t + 1, y_p)) \mid 1 \leq t \leq \ell_p\}$, that is, each point corresponds to a suffix of $[x_p, y_p]$, where the x-coordinate t is the suffix length and the y-coordinate is its sum. Among these points, we keep only those that can be optimal for some $\alpha > 0$. Equivalently, we keep the vertices of the upper convex hull of P (since we maximize $y - \alpha x$). The convex hull can be constructed in $O(\ell_p)$ time, since the points in P are already sorted by their first coordinates [14].

For any suffix represented by a point $(x, y) \in P$, we have $S_\alpha(y_p - x + 1, y_p) = y - \alpha x$. Thus, given an offset α, answering the SMSS query on $X_0[x_p, y_p]$ reduces to finding the point $(x, y) \in P$ that maximizes $y - \alpha x$. By point–line duality, each point (x, y) corresponds to the line $L_{(x,y)}(w) = y - wx$, and $\max_{(x,y)\in P}(y - \alpha x)$ equals the value of the upper envelope at $w = \alpha$, which can be computed in $O(\ell_p)$ time from the convex hull of P. The upper

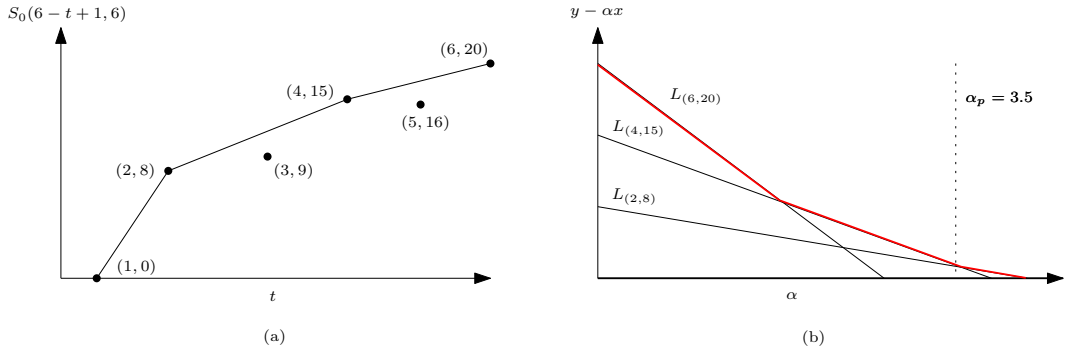

Figure 3 Example illustrating the computation of the threshold offset at the root of T in Figure 2. (a) The point set P with its convex hull. (b) The dual lines corresponding to the convex hull of P. The red segments represent the upper envelope of these lines.

envelope is piecewise linear and changes only at intersection points of consecutive envelope lines. Therefore, α_p is the breakpoint on the envelope where the optimal suffix switches from a suffix whose starting position lies in the left half of $[x_p, y_p]$ to one whose starting position lies in the right half. Since the envelope breakpoints are sorted by slope, α_p can be found by binary search over the hull in $O(\log \ell_p)$ time (using Proposition 2). See Figure 3 for an example.

Thus, the nodes at each level of T can be constructed in $O(n)$ time, implying that the entire tree T can be constructed in $O(n \log n)$ time overall.

We summarize the results of this section in the following theorem.

▶ **Theorem 5.** *Given an array X_0 of size n, there exists an $O(n)$-space data structure that can answer a prefix/suffix R-OMSS query on X_0 in $O(\log^2 n)$ time for any positive offset α. When the query range is a canonical range of X_0, the data structure can answer a prefix/suffix R-OMSS query in $O(\log n)$ time. The data structure can be constructed in $O(n \log n)$ time.*

3.2 Data Structure for General R-OMSS Queries

Building on Section 3.1, we now generalize our findings and devise a data structure for answering (general) R-OMSS queries on X_0 for any positive offset $\alpha > 0$ with the query range $[a, b]$. The data structure is composed of the following three parts:

(1) the array C_0,

(2) the data structures of Theorem 5 on X_0, which can answer a prefix and suffix R-OMSS query on X_0 in $O(\log n)$ time when the query range is canonical, and

(3) an $O(|h - g|)$-space data structure of Lemma 1 constructed for each canonical range $[g, h]$, which enables answering the OMSS query on $X_0[g, h]$ in $O(\log n)$ time.

Since the total length of all canonical ranges of X_0 is $O(n \log n)$, (3) uses $O(n \log n)$ space in total. As both (1) and (2) use $O(n)$ space, the overall data structure requires $O(n \log n)$ space.

To answer an R-OMSS query on X_0 with query range $[a, b]$, we first partition $[a, b]$ into $c = O(\log n)$ distinct canonical ranges $[a_1, b_1], \ldots, [a_c, b_c]$, ordered from left to right, in $O(\log n)$ time. If $c = 1$, we answer the query directly in $O(\log n)$ time using the corresponding OMSS data structure of Lemma 1. Otherwise, we apply a dynamic programming approach similar to that in Section 3.1, as follows. We define an array $A[1..c]$, where for each $k \in [1, c]$, $A[k]$ stores the answer range of the R-OMSS query on X_0 with query range $[a_1, b_k]$, under the condition that the rightmost endpoint of the answer range is in $[a_k, b_k]$.

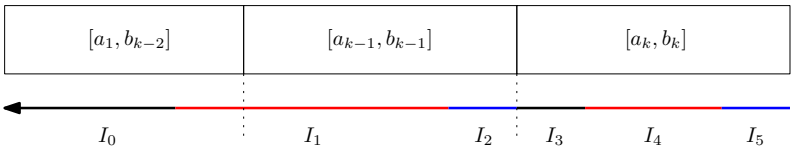

Figure 4 Subranges that partition the range $[a_1, b_k]$.

Then the answer to the R-OMSS query on X_0 with range $[a, b]$ is the maximum value in A, since the rightmost endpoint of the answer range may lie in any of the canonical ranges.

The base case $k = 1$ can be computed in $O(\log n)$ time using the corresponding OMSS data structure of Lemma 1. For $k > 1$, we compute $A[k]$ from $A[k-1]$, we partition the range $[a_1, b_k]$ into five subranges as follows (see Figure 4 for a schematic sketch):

- I_0: The range $[a_1, b_{k-1}] \setminus A[k-1]$
- I_1: The answer range of the R-OMSS query on X_0 with query range $[a_1, b_{k-1}]$ under the condition that the rightmost endpoint of the answer range is in $[a_{k-1}, b_{k-1}]$, i.e., the answer range corresponding to $A[k-1]$.
- I_2: The range $[a_{k-1}, b_{k-1}] \setminus (I_0 \cup I_1)$.
- I_3: The maximal prefix of $X_0[a_k, b_k]$ that does not intersect with I_4.
- I_4: The answer range of the R-OMSS query on X_0 with query range $[a_k, b_k]$.
- I_5: The range $[a_k, b_k] \setminus (I_3 \cup I_4)$.

I_1 (and hence I_0 and I_2) can be obtained from $A[k-1]$ and I_4 (and hence I_3 and I_5) can be computed in $O(\log n)$ time using the data structure of Lemma 1 on the canonical range $[a_k, b_k]$. Thus, all the subranges above can be identified in $O(\log n)$ time. Now we introduce the following lemma, which we use to compute $A[k]$.

▶ **Lemma 6.** *Let I be the answer range of the R-OMSS query on X_0 with query range $[a_1, b_k]$, under the condition that the rightmost endpoint of the answer range falls within $[a_k, b_k]$. Then I must be one of the following five ranges:*

1. I_4,
2. $I_1 \cup I_2 \cup I'_3$,
3. $I_1 \cup I_2 \cup I_3 \cup I_4$,
4. $I'_2 \cup I'_3$, *or*
5. $I'_2 \cup I_3 \cup I_4$.

Here, I'_2 denotes the answer range of the suffix R-OMSS query on X_0 with offset α and query range $[a_{k-1}, b_{k-1}]$, and I'_3 denotes the answer range of the prefix R-OMSS query on X_0 with offset α and query range $[a_k, b_k]$.

Proof. Observe that the sum of elements in X_α over any suffix of I_0 is negative; otherwise, I_1 would include such a suffix of I_0. Similarly, the sum of elements in X_α over any prefix of I_5 is negative. Hence, I does not contain any position from either I_0 or I_5.

Also, if $I'_2 \cap I_1 \neq \emptyset$, then $I_1 \cup I_2$ is the answer range of the SMSS problem on $X_\alpha[a_1, b_{k-1}]$. Otherwise, suppose that $I' \neq I_1 \cup I_2$ is the answer range of the SMSS problem on $X_\alpha[a_1, b_{k-1}]$. By the definition of I'_2, we have $I_2 \subseteq I'_2 \subseteq I'$. Then the sum of X_α over the range $I' \setminus I_2$, which is neither I_1 nor empty (since $I'_2 \cap I_1 \neq \emptyset$), is greater than the sum of X_α over I_1. This contradicts the definition of I_1 as the answer range on $[a_1, b_{k-1}]$. Therefore, when I'_2 contains positions in $[a_1, b_{k-1}]$, I'_2 must be either $I_1 \cup I_2$ or satisfy $I'_2 \subseteq I_2$. By a similar argument for I'_3, we obtain that $I'_3 = I_3 \cup I_4$ or $I'_3 \subseteq I_3$.

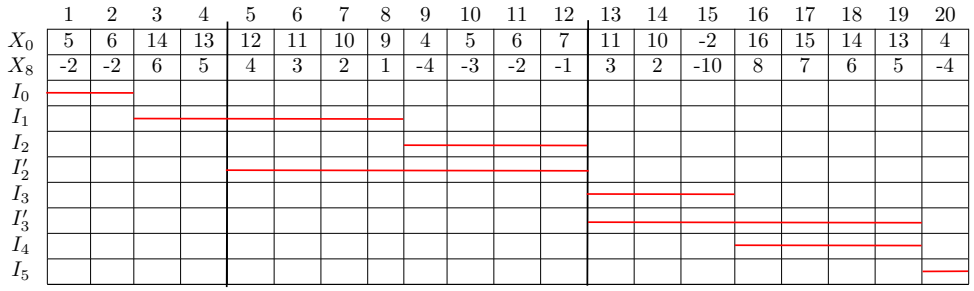

Figure 5 Supplement figure for Example 7. Canonical ranges are represented by thick vertical bars, and the intervals I_j by red horizontal lines in their respective rows.

From the above observations, we consider the following three cases:

(A) $I \subseteq [a_k, b_k]$,

(B) $I \cap I_1 \neq \emptyset$, and

(C) $I \cap I_1 = \emptyset$ but $I \cap I_2 \neq \emptyset$.

For Case (A), we have $I = I_4$ by the definition of I_4. For Case (B), I_1 must be completely contained in I by the observation above. Thus, in this case, I is either $I_1 \cup I_2 \cup I'_3$ or $I_1 \cup I_2 \cup I_3 \cup I_4$ (the latter corresponds to the case where $I'_3 = I_3 \cup I_4$). For Case (C), since $I'_2 \subseteq I_2$ by the observation above, I is either $I'_2 \cup I'_3$ or $I'_2 \cup I_3 \cup I_4$ (again, the latter corresponds to the case where $I'_3 = I_3 \cup I_4$). ◀

▶ **Example 7.** Figure 5 illustrates how Lemma 6 operates. In this example, the range $[1, 20]$ is partitioned into three canonical ranges $[1, 4]$, $[5, 12]$, and $[13, 20]$ (here the notion of canonical ranges is defined differently from that in the main text for convenience).

To answer the query with offset $\alpha = 8$ on the query range $[1, 20]$, consider the step of computing $A[3]$ from $A[2]$. In this case, both $I'_2 \cap I_1$ and $I'_3 \cap I_4$ are non-empty. Therefore, $A[3]$ is the maximum between the sum of X_8 over I_4 and the sum over $I_1 \cup I_2 \cup I_3 \cup I_4$, which are 26 and 32, respectively. Hence, $A[3] = 32$.

Query time. I'_2 can be computed in $O(\log n)$ time using the suffix R-OMSS data structure on X_0 on the canonical query range $[a_{k-1}, b_{k-1}]$ with I_1. Similarly, I'_3 can be computed in $O(\log n)$ time using the prefix R-OMSS data structure on X_0 on the canonical query range $[a_k, b_k]$. Therefore, by Lemma 6, we can compute $A[k]$ from $A[k-1]$ in $O(\log n)$ time, which implies that all values in A can be computed in $O(\log^2 n)$ time. Finally, we can find the maximum value in A in $c = O(\log n)$ time by a linear scan.

Construction time. The construction time of the data structure in Lemma 1 is $O(\ell \log^2 \ell)$, where ℓ denotes the input size. Since the total length of all canonical ranges of X_0 is $O(n \log n)$, this yields a total construction time of $O(n \log^3 n)$ [23]. In addition, constructing the data structure of Theorem 5 requires $O(n \log n)$ time. Hence, the overall construction time of the data structure is $O(n \log^3 n)$.

We summarize the results of this section in the following theorem.

▶ **Theorem 8.** *Given an array X_0 of size n, there exists an $O(n \log n)$-space data structure that can answer an R-OMSS query on X_0 in $O(\log^2 n)$ time for any positive offset α. The construction time of the data structure is $O(n \log^3 n)$.*

4 Special Types of Inputs

Based on the results of the previous section, we analyze and devise specializations of our data structures for R-OMSS queries on X_0 under special input types. Specifically, we consider the following three scenarios: (1) the offset α is restricted to positive integers, (2) X_0 is an array over an alphabet of size σ, and (3) X_0 is given in run-length encoded (RLE) form.

Integer Offsets. When the offset α is restricted to positive integers, the data structure of Theorem 8 can be improved by the following modifications. First, we introduce a lemma that states the data structure of Lemma 1 can be improved under this restriction.

▶ **Lemma 9.** *When the offset α is restricted to positive integers, there exists an $O(n)$-space data structure that can answer an OMSS query on X_0 in $O(\log\log n)$ time.*

Proof. We modify the data structure of Lemma 1 as follows. For each (offset, interval) pair (α, I) in L (the list defined in the statement of Lemma 1), we replace it with the integer pair $(\lceil\alpha\rceil, I)$. If there exist two distinct pairs (α_1, I_1) and (α_2, I_2) in L such that $\alpha_1 < \alpha_2$ and $\lceil\alpha_1\rceil = \lceil\alpha_2\rceil$, we discard the pair (α_1, I_1) and maintain only $(\lceil\alpha_2\rceil, I_2)$ in L. We then maintain L using a y-fast trie [27], which supports predecessor searches on L in $O(\log\log n)$ time using $O(n)$ space. ◀

Next, for each canonical range $[g, h]$ of X_0, let $T_{[g,h]}$ be the rooted subtree of the SBST T on X_0 (defined in Section 3.1) whose root corresponds to $[g, h]$. We define a list $L_{[g,h]}$ of (offset, interval) pairs as follows: $(\beta, I) \in L_{[g,h]}$ if and only if β is a threshold offset stored in the node of $T_{[g,h]}$, and I is the answer range of the suffix R-OMSS query on X_0 with query range $[g, h]$ and offset β. The list $L_{[g,h]}$ has size $O(|h - g|)$, since the number of distinct offsets equals the number of nodes in $T_{[g,h]}$. Moreover, by the algorithm for answering suffix R-OMSS queries on X_0 using T, for any offset α, the answer range of the suffix R-OMSS query on X_0 with the query range $[g, h]$ is exactly the interval I such that $(\beta, I) \in L_{[g,h]}$, where β is the predecessor of α in $L_{[g,h]}$.

Therefore, $L_{[g,h]}$ plays the same role as the data structure of Lemma 1 for the suffix R-OMSS query on X_0 when the query range is restricted to $[g, h]$. By maintaining such lists for all canonical ranges of X_0, together with the modifications used in the proof of Lemma 9, we obtain the following lemma, since the total length of all canonical ranges of X_0 is $O(n \log n)$.

▶ **Lemma 10.** *When the offset α is restricted to positive integers and the query range is restricted to a canonical range of X_0, there exists an $O(n \log n)$-space data structure that can answer an suffix R-OMSS query on X_0 in $O(\log\log n)$ time.*

Finally, by combining the data structures of Lemma 9 and Lemma 10, together with C, we obtain the following corollary.

▶ **Corollary 11.** *When the offset α is restricted to positive integers, there exists an $O(n \log n)$-space data structure that can answer an R-OMSS query on X_0 in $O(\log n \log\log n)$ time.*

Bounded Alphabet Size. When X_0 is an array over an alphabet $\Sigma \subset \mathbb{R}$ of size σ, we show that the data structure in Lemma 1 requires $\Omega(\sqrt{n})$ space, even when X_0 is a binary array. Recall that this data structure partitions the domain of offsets into the ranges $[0, \alpha_1)$, $[\alpha_1, \alpha_2)$, $[\alpha_2, \alpha_3), \ldots, [\alpha_{k-1}, \alpha_k), [\alpha_k, \infty)$ and assigns each such range $[\alpha_i, \alpha_{i+1})$ an answer range $[a_i, b_i]$ such that $[a_i, b_i]$ is the answer range for each offset $\alpha \in [\alpha_i, \alpha_{i+1})$ (defining

the border cases $\alpha_0 = 0$ and $\alpha_{k+1} = \infty$). Therefore, the question arises how large k must be. We say that two answer ranges $[a, b]$ and $[a', b']$ for the OMSS queries on X_0 with two different offsets α and α', respectively, are *non-compatible* if $S_\alpha(a, b) > S_\alpha(a', b')$ or $S_{\alpha'}(a, b) < S_{\alpha'}(a, b)$. Distinct answer ranges are compatible if and only if their respective subarrays in X_0 are permutations of each other. For the analysis, we say that two offsets α and α' are *non-compatible* if the offsets α and α' have no answer range in common, i.e., there is no answer range of α that is compatible with any answer range of α' (and vice versa).

▶ **Theorem 12.** *There exists a binary array X_0 of size n with $\Theta(\sqrt{n})$ non-compatible offsets.*

Proof. Consider the input $X_0 = 110^1 10^2 10^3 10^4 \cdots 0^k$ for $k = \Theta(\sqrt{n})$. Then, for each $k' \in [k-1]$, when the offset α lies in the interval $(1/(k'+1), 1/k')$, the answer range of the OMSS query on X_0 is the minimal prefix range that covers $1 + \lfloor 1/k' \rfloor$ ones. Since for each such k' there is exactly one answer range, and all these answer ranges are non-compatible, the claim follows. ◀

For the bounded-alphabet setting, improving the $\Omega(\sqrt{n})$ space lower bound for the data structure of Lemma 1, or designing an indexing data structure for OMSS queries that uses $o(n)$ space, remains open. In Section 5, we will discuss the space usage of the data structure of Theorem 5 and Theorem 8 in the bounded-alphabet case.

Run-Length Encoded (RLE) Input. If the input array $X_0 = X_0[j_1]^{k_1} X_0[j_2]^{k_2} \cdots X_0[j_z]^{k_z}$ for $j_1 = 1 < j_2 < \cdots < j_z = n$ and $k_1, \ldots, k_z \geq 1$ of size n has z distinct runs, we can reduce the space usage of the data structures in Lemma 1, Theorem 5, and Theorem 8 to $O(z)$ space by indexing the RLE form of X_0. The *RLE form* of X_0 is a compact representation of X_0 that encodes each run $X_0[j_i]^{k_i}$ of X_0 by a pair of value $X_0[j_i]$ and the length of the run k_i. Hence, X_0 can be represented in $O(z)$ space with this RLE form.

We make use of the RLE form of X_0 as follows. As common components of our data structure, we maintain the following arrays:

1. $X' := [X_0[j_1], X_0[j_2], \ldots, X_0[j_z]]$ (i.e., X_0 in RLE form),
2. D of size z, where $D[i] = \sum_{t=1}^{i} k_t$, and
3. C' of size z, where $C'[i] = \sum_{t=1}^{i} k_t \cdot X'[t]$ for all $i \in [1, z]$.

These arrays use $O(z)$ space in total. We stipulate that $D[0] = 0$.

Given a position $i \in [1, n]$ of X_0 with $i \in [D[i'-1]+1, D[i']]$ for some $i' \in [1, z]$, i.e., i is in the i'-th run X_0, we can compute i' from i in $O(\log z)$ time using a binary search on D. Then, we have $X_0[i] = X'[i']$ and $C_0[i] = C'[i'-1] + (i - D[i'-1]) \cdot X'[i']$. Conversely, given a position $i' \in [1, z]$ of X', i' corresponds to the i'-th run that starts at position $D[i'-1]+1$ and ends at position $D[i']$ in X_0. We now present a lemma showing that the data structure of Lemma 1 can be improved under this representation.

▶ **Lemma 13.** *There exists a data structure of size $O(z)$ that supports OMSS queries on X_0 in $O(\log z)$ time.*

Proof. We construct the data structure of Lemma 1 on X' using $O(z)$ space, which supports OMSS queries on X' in $O(\log z)$ time. To answer an OMSS query on X with offset α, we first compute the OMSS on X' with the same offset in $O(\log z)$ time. Let $[i', j']$ be the answer range of the OMSS on X'. We claim that $[i, j]$ is the answer range on X, where i (resp. j) is the leftmost (resp. rightmost) position in X corresponding to i' (resp. j').

To prove the claim, it suffices to show that the endpoints of the answer range on X_0 must lie on run boundaries. Assume the contrary, i.e., one side j of the answer range partially covers a run. By definition, this run covering position $X_0[j]$ corresponds to a positive value

in $X_\alpha[j]$, otherwise the range could be shortened to exclude all values of this run and thus increase the sum. However, since $X_\alpha[j]$ is positive, extending the range to include the entire run further increases the sum, and thus contradicts the assumption.

Since the claim holds and the corresponding positions i and j can be computed in $O(\log z)$ time, we can answer an OMSS query on X_0 in $O(\log z)$ time overall. ◀

From the claim used in the proof of Lemma 13, we can also obtain data structures in Theorem 5 and Theorem 8 parameterized by z, as summarized in the following corollaries.

▶ **Corollary 14.** *There exists a data structure of size $O(z)$ that supports suffix R-OMSS queries on X_0 in $O(\log^2 z)$ time.*

Proof. We construct the data structure of Theorem 5 on X' using $O(z)$ space, which supports OMSS queries on X' in $O(\log^2 z)$ time. To answer a suffix R-OMSS query on X_0 with query range $[a, b]$ and offset α, we first compute the OMSS on X' with the same offset and query range $[a', b']$ in $O(\log^2 z)$ time, where a' and b' are the positions in X' corresponding to the runs of X_0 containing the positions a and b, respectively.

Let $[i', j']$ be the answer range of the OMSS on X', and i and j be the leftmost and the rightmost positions in X_0 corresponding to i' and j', respectively. Then, by the claim in the proof of Lemma 13, the answer range of the query on X_0 is $[\max(a, i), b]$. ◀

▶ **Corollary 15.** *There exists a data structure of size $O(z \lg z)$ that supports R-OMSS queries on X_0 in $O(\log^2 z)$ time.*

Proof. The proof is analogous to that of Corollary 14, exchanging the data structure of Theorem 5 with that of Theorem 8. The only difference is that we return $[\max(a, i), \min(b, j)]$ as the answer range, where j is the rightmost position in X_0 corresponding to j' (all notations are the same as in the proof of Corollary 14). ◀

▶ Remark 16. As all values in D are integers, we can locate any position in X and its corresponding position in X' (and vice versa) in $O(\log \log z)$ time by maintaining a y-fast trie on D. Consequently, by applying the results of Lemma 9, Lemma 10, and Corollary 11, the query times of the data structures in Lemma 13, Corollary 14, and Corollary 15 can be improved to $O(\log \log z)$, $O(\log z \log \log z)$, and $O(\log z \log \log z)$ time, respectively, when the offset α is restricted to positive integers.

5 Lower Bounds

Finally, we present lower bound results for data structures that answer R-OMSS queries on X_0. We first show a reduction from the predecessor query to the OMSS query, and subsequently show that any encoding data structure that answers suffix R-OMSS queries requires at least $\Omega(n \log n)$ bits.

Time-Space Trade-Offs Lower Bound. In the following theorem, we show there exists a reduction from the predecessor query to OMSS query.

▶ **Theorem 17.** *For any set S of n values from a universe $U \subset \mathbb{R}$, given a value $\alpha \in U$, the predecessor of α in S can be determined by a OMSS query and an access on an array of size n.*

Proof. Suppose we want to answer a predecessor query for a value $\alpha \in U$ in S. For that, we have constructed, in a precomputation step, an OMSS data structure on X_0, where X_0 stores the values of S in ascendingly sorted order, and compute the OMSS query with offset α. If the answer range for α is $[n, n]$, then the predecessor of α is either $X_0[n]$ (if $X_0[n] \leq \alpha$) or $X_0[n-1]$ (if $X_0[n] > \alpha$). If the answer range is $[1, n]$, then there is no predecessor of α in S. Otherwise, the answer range is $[i, n]$ with $i \in [2, n-1]$. Then the predecessor of α is $X_0[i-1]$ since (a) $X_\alpha[k] < 0$ for all $k \leq i-2$, (b) $X_\alpha[i-1] \leq 0$, and (c) $X_\alpha[k] > 0$ for all $k \geq i$. We here assume that the answer range of the OMSS query is the shortest one (omitting trailing zero entries), which can be ensured by a tie-breaking rule in the definition of the OMSS query. ◀

From the predecessor lower bound of Pătraşcu and Thorup [18], the data structure of Lemma 1 achieves an optimal tradeoff between space and query time. Moreover, the data structures of Theorem 5 and Theorem 8 also provide optimal space–query time tradeoffs up to a $\log n$ factor in the query time, since they can answer OMSS queries.

Space Lower Bounds in the Encoding Model. One may also study the R-OMSS query in the variant where only the answer range in X_0 is returned. In this setting, the problem is considered under the *encoding model*: after preprocessing X_0, we maintain only the information necessary to answer queries, without storing X_0 explicitly. For many range queries, encoding data structures use less space than explicitly storing X_0 (see [20] for details). For example, there exists an $O(n)$-bit data structure for answering range MSS queries under a word-RAM model with a word size of $\Theta(\log n)$ bits [12], which uses asymptotically less space than the $\Omega(n \log n)$ bits required to store X_0 under the same model. We now prove the following theorem, which shows that for the suffix R-OMSS queries (and hence for R-OMSS queries), any encoding data structure must use asymptotically as much space as storing the input. This implies that the data structure of Theorem 5 uses asymptotically optimal space.

▶ **Theorem 18.** *Any permutation X_0 of $[1, n]$ can be reconstructed from an encoding that supports suffix R-OMSS queries. Consequently, any encoding data structure that answers suffix R-OMSS queries requires at least $\Omega(n \log n)$ bits.*

Proof. We first identify the position of 1 as follows. Let $\alpha_1 = 0$ and $\alpha_2 = 1 + \epsilon$, where $0 < \epsilon < 1$ is a fixed constant. For each $a \in [1, n-1]$, we perform R-OMSS queries on the range $[a, a+1]$ with offsets α_1 and α_2. From the definition of the offsets, $X_0[i] = 1$ if and only if the answer ranges for the query range $[i, i+1]$ are $[i, i+1]$ under α_1 and $[i+1, i+1]$ under α_2. Since X_0 is a permutation, there exists exactly one such position i unless $X_0[n] = 1$. To check whether $X_0[n] = 1$, we perform two queries with query range $[n-1, n]$: one query with offset α_1 and expected answer range $[n-1, n]$, and another query with offset α_2 and expected answer range $[n-1, n-1]$.

After locating the position i of 1, we split the permutation into the two subarrays $X_0[1, i-1]$ and $X_0[i+1, n]$. We then find the position of 2 in the same manner, which must lie in exactly one of these two subarrays. Repeating this procedure recursively determines the position of every value up to n, and hence reconstructs X_0. ◀

As a corollary of Theorem 18 we can also obtain the space lower bound for suffix R-OMSS queries (and hence for R-OMSS queries) in the encoding model with bounded alphabet size.

▶ **Corollary 19.** *Any encoding data structure that supports suffix R-OMSS queries on an array of size n over an alphabet of size σ requires at least $\Omega(n \log \sigma)$ bits.*

Proof. Let $\mathcal{X}$ be the set of all arrays of size n obtained by concatenating n/σ permutations of $[1, \sigma]$ (we assume that n is a multiple of σ, which does not affect the asymptotic lower bound). Then $|\mathcal{X}| = (\sigma!)^{n/\sigma}$, and hence $\log|\mathcal{X}| = \Omega(n \log \sigma)$. Moreover, any array in $\mathcal{X}$ can be reconstructed from suffix R-OMSS queries, since each permutation block can be reconstructed by Theorem 18. This proves the corollary. ◀

Corollary 19 implies that even in the bounded-alphabet setting, any encoding data structure for suffix R-OMSS queries (and hence for R-OMSS queries) must use asymptotically as much space as required to store the input.

6 Open Problems

There are many gaps left open regarding time and space complexities. It is puzzling to us whether we can construct the data structure of Lemma 1 in optimal $O(n)$ time. This would improve the construction time of our R-OMSS data structures in Theorem 8. Here, we additionally want to improve the space to $O(n)$, which would be optimal by Theorem 18. Improving the query time to $O(\log n)$ is also an interesting open problem, which would be optimal by Theorem 17 for general offsets. In case that the number of distinct values in X_0 is bounded, we wonder whether we can improve lower and upper bounds on the number of non-compatible offsets.

For future directions, we want to study the variations of the MSS problem described in the introduction as query problems with offset and/or query range parameters.

References

1 Sung Eun Bae. *Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem*. PhD thesis, University of Canterbury, 2007.

2 Sung Eun Bae, Tong-Wook Shinn, and Tadao Takaoka. Efficient algorithms for the maximum sum problems. *Algorithms*, 10(1), 2017. doi:10.3390/a10010005.

3 Fredrik Bengtsson and Jingsen Chen. Computing Maximum-Scoring segments optimally. Research Report 2007:03, Luleå University of Technology, Department of Computer Science and Electrical Engineering, 2007.

4 Jon Louis Bentley. Perspective on performance. *Commun. ACM*, 27(11):1087–1092, 1984. doi:10.1145/1968.381154.

5 Timothy M. Chan and R. Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

6 Kun-Mao Chao. Maximum-sum segments. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1242–1244. Springer New York, New York, NY, 2016. doi:10.1007/978-1-4939-2864-4_226.

7 Kuan-Yu Chen and Kun-Mao Chao. On the range maximum-sum segment query problem. *Discrete Applied Mathematics*, 155(16):2043–2052, 2007. doi:10.1016/J.DAM.2007.05.018.

8 Chih-Huai Cheng, Hsiao-Fei Liu, and Kun-Mao Chao. Optimal algorithms for the average-constrained maximum-sum segment problem. *Inf. Process. Lett.*, 109(3):171–174, 2009. doi:10.1016/J.IPL.2008.09.024.

9 Kai-Min Chung and Hsueh-I Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2004. doi:10.1137/S0097539704440430.

10 Christoph Dürr and Jill-Jênn Vie. *Competitive Programming in Python: 128 Algorithms to Develop your Coding Skills*. Cambridge University Press, 2020. URL: https://books.google.de/books?id=JA0FEAAAQBAJ.

11 Takeshi Fukuda, Yasuhiko Morimoto, Shinichi Morishita, and Takeshi Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.*, 26(2):179–213, 2001. doi:10.1145/383891.383893.

12 Paweł Gawrychowski and Patrick K Nicholson. Encodings of range maximum-sum segment queries and applications. In *Annual Symposium on Combinatorial Pattern Matching*, pages 196–206. Springer, 2015.

13 Michael H. Goldwasser, Ming-Yang Kao, and Hsueh-I Lu. Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In *Proc. WABI*, volume 2452 of *LNCS*, pages 157–171, 2002. doi:10.1007/3-540-45784-4_12.

14 Ronald L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

15 Sung Kwon Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *Information Processing Letters*, 86(6):339–342, 2003. doi:10.1016/S0020-0190(03)00225-4.

16 Yaw-Ling Lin, Tao Jiang, and Kun-Mao Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.*, 65(3):570–586, 2002. doi:10.1016/S0022-0000(02)00010-7.

17 Hsiao-Fei Liu and Kun-Mao Chao. Algorithms for finding the weight-constrained k longest paths in a tree and the length-constrained k maximum-sum segments of a sequence. *Theor. Comput. Sci.*, 407(1-3):349–358, 2008. doi:10.1016/J.TCS.2008.06.052.

18 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 232–240, 2006.

19 Kalyan S. Perumalla and Narsingh Deo. Parallel algorithms for maximum subsequence and maximum subarray. *Parallel Process. Lett.*, 5:367–373, 1995. doi:10.1142/S0129626495000345.

20 Rajeev Raman. Encoding data structures. In *International Workshop on Algorithms and Computation*, pages 1–7. Springer, 2015. doi:10.1007/978-3-319-15612-5_1.

21 Walter L. Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *Proc. ISMB*, pages 234–241, 1999. URL: http://www.aaai.org/Library/ISMB/1999/ismb99-027.php.

22 Yoshifumi Sakai. A maximal local maximum-sum segment data structure. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101-A(9):1541–1542, 2018. doi:10.1587/TRANSFUN.E101.A.1541.

23 Yoshifumi Sakai. A Data Structure for the Maximum-Sum Segment Problem with Offsets. In *35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024)*, volume 296 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CPM.2024.26.

24 Tadao Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science*, 61:191–200, 2002. CATS'02, Computing: the Australasian Theory Symposium. doi:10.1016/S1571-0661(04)00313-5.

25 Hisao Tamaki and Takeshi Tokuyama. Algorithms for the maximum subarray problem based on matrix multiplication. *Interdisciplinary Information Sciences*, 6(2):99–104, 2000. doi:10.4036/iis.2000.99.

26 Lusheng Wang and Ying Xu. SEGID: identifying interesting segments in (multiple) sequence alignments. *Bioinform.*, 19(2):297–298, 2003. doi:10.1093/BIOINFORMATICS/19.2.297.

27 Dan E Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983.

28 Hung-I Yu, Tien-Ching Lin, and D. T. Lee. Finding maximum sum segments in sequences with uncertainty. *Theor. Comput. Sci.*, 850:221–235, 2021. doi:10.1016/J.TCS.2020.11.005.