Encoding Two-Dimensional Range Top-k Queries Revisited

Seungbum Jo[1]
University of Siegen, Germany
Seungbum.Jo@uni-siegen.de
https://orcid.org/0000-0002-8644-3691

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

Abstract

We consider the problem of encoding two-dimensional arrays, whose elements come from a total order, for answering Top-k queries. The aim is to obtain encodings that use space close to the information-theoretic lower bound, which can be constructed efficiently. For $2 \times n$ arrays, we first give upper and lower bounds on space for answering sorted and unsorted 3-sided Top-k queries. For $m \times n$ arrays, with $m \leq n$ and $k \leq mn$, we obtain $(m \lg \binom{(k+1)n}{n} + 4nm(m-1) + o(n))$-bit encoding for answering sorted 4-sided Top-k queries. This improves the $\min(O(mn \lg n), m^2 \lg \binom{(k+1)n}{n} + m \lg m + o(n))$-bit encoding of Jo et al. [CPM, 2016] when $m = o(\lg n)$. This is a consequence of a new encoding that encodes a $2 \times n$ array to support sorted 4-sided Top-k queries on it using an additional $4n$ bits, in addition to the encodings to support the Top-k queries on individual rows. This new encoding is a non-trivial generalization of the encoding of Jo et al. [CPM, 2016] that supports sorted 4-sided Top-2 queries on it using an additional $3n$ bits. We also give almost optimal space encodings for 3-sided Top-k queries, and show lower bounds on encodings for 3-sided and 4-sided Top-k queries.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Encoding model, top-k query, range minimum query

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.69

Related Version An extended version of the paper is available at [12], `https://arxiv.org/abs/1809.07067`.

Acknowledgements We would like to thank Moses Ganardi for helpful discussions

1 Introduction

Given a one-dimensional (1D) array $A[1 \dots n]$ of n elements from a total order, the *range Top-k query on A* (Top-$k(i, j, A), 1 \leq i, j \leq n$) returns the positions of k largest values in $A[i \dots j]$. In this paper, we refer to these queries as 2-sided Top-k queries; and the special case where the query range is $[1 \dots i]$, for $1 \leq i \leq n$, as the 1-sided Top-k queries. We can extend the definition to the two-dimensional (2D) case – given an $m \times n$ 2D array $A[1 \dots m][1 \dots n]$ of mn elements from a total order and a $k \in \{1, \dots, mn\}$, the *range Top-k query on A* (Top-$k(i, j, a, b, A), 1 \leq i, j \leq m, 1 \leq a, b \leq n$) returns the positions of k largest values in

[1] The author of this paper is supported by the DFG research project LO748/11-1.

29th International Symposium on Algorithms and Computation (ISAAC 2018).
Editors: Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao; Article No. 69; pp. 69:1–69:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$A[i \dots j][a \dots b]$. Without loss of generality, we assume that all elements in A are distinct (by ordering equal elements based on the lexicographic order of their positions). Also, we assume that $m \leq n$. In this paper, we consider the following types of Top-k queries.

- Based on the order in which the answers are reported
  - Sorted query: the k positions are reported in sorted order of their corresponding values.
  - Unsorted query: the k positions are reported in an arbitrary order.
- Based on the query range
  - 3-sided query: the query range is $A[i \dots j][1 \dots b]$, for $i, j \in \{1, m\}$, and $b \in \{1, n\}$.
  - 4-sided query: the query range is $A[i \dots j][a \dots b]$, for $i, j \in \{1, m\}$, and $a, b \in \{1, n\}$.

We consider how to support these range Top-k queries on A in the *encoding model*. In this model, one needs to construct a data structure (an encoding) so that queries can be answered without accessing the original input array A. The minimum size of an encoding is also referred to as the *effective entropy* of the input data [8]. Our aim is to obtain encodings that use space close to the effective entropy, which can be constructed efficiently. In the rest of the paper, we use Top-$k(i, j, a, b)$ to denote Top-$k(i, j, a, b, A)$ if A is clear from the context. Also, unless otherwise mentioned, we assume that all Top-k queries are sorted 4-sided Top-k queries. Finally, we assume the standard word-RAM model [14] with word size $\Theta(\lg n)$.

1.1 Previous work

The problem of encoding 1D and 2D arrays to support Top-k queries has been widely studied in the recent years. Especially, the case when $k = 1$, which is commonly known as the *Range maximum query* (RMQ) problem, has been studied extensively, and has a wide range of applications [1]. Optimal encodings for answering RMQ queries on 1D and 2D arrays are well-studied. Fischer and Heun [5] proposed a $2n + o(n)$-bit data structure which answers RMQ queries on 1D array of size n in constant time. For a 2D array A of size $m \times n$, a trivial way to encode A for answering RMQ queries is to store the rank of all elements in A, using $O(nm \lg n)$ bits. Golin et al. [8] show that when $m = 2$ and RMQ encodings on each row are given, one can support RMQ queries on A using $n - O(\lg n)$ extra bits by encoding *joint Cartesian tree* on both rows. By extending the above encoding, they obtained $nm(m + 3)/2$-bit encoding for answering RMQ queries on A, which takes less space than the trivial $O(nm \lg n)$-bit encoding when $m = o(\lg n)$. Brodal et al. [3] proposed an $O(\min(nm \lg n, m^2 n))$-bit data structure which supports RMQ queries on A in constant time. Finally, Brodal et al. [2] obtained an optimal $O(nm \lg m)$-bit encoding for answering RMQ queries on A (although the queries are not supported efficiently).

For the case when $k = 2$, Davoodi et al. [4] proposed a $3.272n + o(n)$-bit data structure to encode a 1D array of size n, which supports Top-2 queries in constant time. The space was later improved by Gawrychowski and Nicholson [7] to the optimal $2.755n + o(n)$ bits, although it does not support queries efficiently. For Top-2 queries on $2 \times n$ array A, Jo et al. [11] showed that $3n + o(n)$-bit extra space is enough for answering 4-sided Top-2 queries on A, when encodings of 2-sided Top-2 queries for each row are given.

For general k, on a 1D array of size n, Grossi et al. [10] proposed an $O(n \lg k)$-bit[2] encoding which supports sorted Top-k queries in $O(k)$ time, and showed that at least $n \lg k - O(n)$ bits are necessary for answering 1-sided Top-k queries; Gawrychowski and Nicholson [7] proposed a $(k + 1)nH(1/(k + 1)) + o(n)$-bit[3] encoding for Top-k queries (although the queries are not

[2] We use $\lg n$ to denote $\log_2 n$.
[3] $H(x) = x \lg(1/x) + (1 - x) \lg(1/(1 - x))$, i.e., an entropy of the binary string whose density of zero is x

Table 1 Summary of the results of upper and lower bounds for Top-k encodings on 2D arrays. The lower bound results marked (*) (of Theorem 12 and 13) are for the additional space (in bits) necessary, assuming that encodings of Top-k queries for both rows are given.

Dimension	Query type	Space (in bits)	Reference
Upper bounds			
$2 \times n$	3-sided, unsorted	$2\lg\binom{(k+1)n}{n} + \lceil (n - \lfloor k/2 \rfloor)\lg 3 \rceil + o(n)$	Theorem 2
	3-sided, sorted	$2\lg\binom{3n}{n} + 2n + o(n)$	Theorem 3, $k = 2$
		$2\lg\binom{(k+1)n}{n} + \lceil 2n\lg 3 \rceil + o(n)$	Theorem 4
	4-sided, sorted	$5n - O(\lg n)$	[8], $k = 1$
		$2\lg\binom{3n}{n} + 3n + o(n)$	[11], $k = 2$
		$2\lg\binom{(k+1)n}{n} + 4n + o(n)$	Theorem 8
$m \times n$	4-sided, sorted	$O(\min(nm\lg n, m^2 n))$	[8], $k = 1$
		$O(nm\lg m)$	[2], $k = 1$
		$m^2\lg\binom{(k+1)n}{n} + m\lg m + o(n)$	[11]
		$m\lg\binom{(k+1)n}{n} + 2nm(m-1) + o(n)$	Theorem 9
Lower bounds			
$2 \times n$	4-sided	$5n - O(\lg n)$	[8], $k = 1$
	3-sided, unsorted	$1.27(n - k/2) - o(n)$	(*) Theorem 12
	3 or 4-sided, sorted	$2n - O(\lg n)$	(*) Theorem 13
$m \times n$	4-sided, sorted	$\Omega(nm\lg(\max(m,k)))$	[3, 10]

supported efficiently), and showed that at least $(k+1)nH(1/(k+1))(1-o(1))$ bits are required to encode Top-k queries. They also proposed a $(k+1.5)nH(1.5/(k+1.5)) + o(n\lg k)$-bit data structure for answering Top-k queries in $O(k^6\lg^2 nf(n))$ time, for any strictly increasing function f. For a 2D array A of size $m \times n$, one can answer Top-k queries using $O(nm\lg n)$ bits, by storing the rank of all elements in A. Jo et al. [11] recently developed the first non-trivial Top-k encodings on 2D arrays. They proposed an $(m^2\lg\binom{(k+1)n}{n} + m\lg m + o(n))$-bit encoding for sorted 4-sided Top-k queries, which takes less space than trivial $O(nm\lg n)$-bit encoding when $n = \Omega(k^m)$. They also proposed an $O(nm\lg n)$-bit data structure which supports Top-k queries in $O(k)$ time.

1.2 Our results

For any $2 \times n$ array A, we first show, in Section 2, that given the sorted 1-sided Top-k encodings of the two individual rows, we can support the 3-sided sorted (resp., unsorted) Top-k queries on A using an additional $\lceil (n - \lfloor k/2 \rfloor)\lg 3 \rceil + o(n)$ (resp., $\lceil 2n\lg 3 \rceil + o(n)$) bits. For unsorted queries, our encoding can answer the queries in $2T(n,k) + O(1)$ time, when one can answer the 1-sided sorted Top-k queries for each row in $T(n,k)$ time.

For 4-sided Top-k queries on A, we show that $4n$ bits are sufficient for answering sorted 4-sided Top-k queries on $2 \times n$ array, when encodings for answering sorted 2-sided Top-k queries for each row are given. This encoding is obtained by extending a DAG for answering Top-2 queries on $2 \times n$ array which is proposed by Jo et al. [11], but we use a different approach from their encoding to encode the DAG. Our result generalizes the $(5n - O(\lg n))$-bit encoding of RMQ query on $2 \times n$ array proposed by Golin et al. [8] to general k, and shows that we can encode a joint Cartesian tree for general k (which corresponds to the DAG in our paper) using $4n$ bits. Note that the additional space is independent of k. We also obtain a data structure for answering Top-k queries in $O(k^2 + kT(n,k))$ time using $2S(n,k) + (4k+7)n + ko(n)$ bits, if there exists an $S(n,k)$-bit encoding to answer sorted 2-sided Top-k queries on a 1D array of size n in $T(n,k)$ time. Comparing to the $2S(n,k)+4n+o(n)$-bit encoding, this data structure uses more space but supports Top-k queries efficiently (the $2S(n,k)+4n+o(n)$-bit encoding takes $O(k^2n^2+nkT(n,k))$ time for answering Top-k queries).

By extending the $2S(n,k)+4n+o(n)$-bit encoding on $2\times n$ array, we obtain $(m\lg\binom{(k+1)n}{n}+2nm(m-1)+o(n))$-bit encoding for answering 4-sided Top-k queries on $m\times n$ arrays. This improves upon the trivial $O(mn\lg n)$-bit encoding when $m=o(\lg n)$, and also generalizes the $nm(m+3)/2$-bit encoding [8] for answering RMQ queries. Comparing with Jo et al.'s [11] $(m^2\lg\binom{(k+1)n}{n}+m\lg m+o(n))$-bit encoding, our encoding takes less space in all cases (for $k>1$) since $m^2\lg\binom{(k+1)n}{n}=m\lg\binom{(k+1)n}{n}+m(m-1)\lg\binom{(k+1)n}{n}$. The trivial encoding of the input array takes $O(nm\lg n)$ bits, whereas one can easily show a lower bound of $\Omega(nm\lg(\max(m,k)))$ bits for any encoding of an $m\times n$ array that supports Top-k queries since at least $O(nm\lg m)$ bits are necessary for answering RMQ queries [3], and at least $n\lg k$ bits are necessary for answering Top-k queries for each row [10]. Thus, there is only a small range of parameters where a strict improvement over the trivial encoding is possible. Our result closes this gap partially, achieving a strict improvement when $m=o(\lg n)$.

Finally in Section 4, given a $2\times n$ array A, we consider the lower bound on additional space required to answer unsorted (or sorted) Top-k on A when encodings of Top-k query for each row are given. We show that at least $1.27(n-k/2)-o(n)$ (or $2n-O(\lg n)$) additional bits are necessary for answering unsorted (or sorted) 3-sided Top-k queries on A, when encodings of unsorted (or sorted) 1-sided Top-k query for each row are given. We also show that $2n-O(\lg n)$ additional bits are necessary for answering sorted 4-sided Top-k queries on A, when encodings of unsorted (or sorted) 2-sided Top-k query for each row are given. These lower bound results imply that our encodings in Sections 2 and 3 are close to optimal (i.e., within $O(n)$ bits of the lower bound), since any Top-k encoding for the array A also needs to support the Top-k queries on the individual rows. All these results are summarized in Table 1.

2 Encoding 3-sided range Top-k queries on $2\times n$ array

In this section, we consider the upper bounds on space for encoding unsorted and sorted 3-sided Top-k queries on $2\times n$ array $A[1,2][1\dots n]$, given the encodings of Top-k on the two individual rows. For the case of $k=1$ (i.e., the RMQ problem), there exists an optimal $(5n-O(\lg n))$-bit encoding of a $2\times n$ array, which stores two Cartesian trees for the individual rows, and encodes the additional information (to answer the queries involving both rows) using a *joint Cartesian tree* [8]. In the rest of this section, we assume that $k>1$. We first consider answering unsorted and sorted 3-sided Top-k queries. If sorted 1-sided Top-k queries on each row can be answered using $S(n,k)$ space[4], we can support unsorted and sorted 3-sided Top-k queries on A using $(2S(n,k)+\lceil(n-\lfloor k/2\rfloor)\lg 3\rceil)$ and $(2S(n,k)+\lceil 2n\lg 3\rceil+o(n))$ bits respectively. For $i\in\{1,2\}$, let $A_i=[a_{i1},\dots,a_{in}]$ be an array of size n constituting the i-th row of A and let (i,j) denote the position in the i-th row and j-th column in A. We first introduce a lemma from Grossi et al. [9], to support queries efficiently.

[4] here and in the rest of the paper, we assume that $S(n,k)=S(n,n)$ for $k>n$

▶ **Lemma 1** ([9]). *Let A be an array of size n over an alphabet of size 3. Then one can encode A using at most $nH_0(A) + o(n) \leq \lceil n \lg 3 \rceil + o(n)$ bits, while supporting the following queries in $O(1)$ time ($H_0(A)$ denotes the zeroth-order entropy of A).*

- $\mathsf{rank}_A(x, i)$: *returns the number of occurrence of the symbol x in $A[1 \dots i]$*
- $\mathsf{select}_A(x, i)$: *returns the position of the i-th occurrence of the symbol x in A.*

Also, we define $\mathsf{rank}_A(x, 0) = \mathsf{select}_A(x, 0) = 0$

Encoding unsorted 3-sided Top-k queries on $2 \times n$ array. We now show how to support (unsorted and sorted) 3-sided Top-k queries on a $2 \times n$ array A, given the sorted 1-sided Top-k encodings on the two rows A_1 and A_2. (Note that in 1D, the space used by the sorted and unsorted 1-sided Top-k encodings differ by $O(k \lg k)$ bits.) For $1 \leq i \leq n$, let f_i and $s_i = k - f_i$ be the number of answers to the (sorted or unsorted) Top-$k(1, 2, 1, i)$ query that belong to the first row and the second row, respectively. We first consider the unsorted case. Since the encodings for answering unsorted 1-sided Top-k queries on A_1 and A_2 are given, it is enough to show how to answer 1-sided Top-k queries on A (to support all possible unsorted 3-sided Top-k queries).

▶ **Theorem 2.** $(*)$[5] *Let A be a $2 \times n$ array. For $1 < k \leq 2n$, if we have $S(n, k)$-bit encoding which can answer the sorted 1-sided Top-k queries for each row in $T(n, k)$ time, then we can answer unsorted 3-sided Top-k queries on A using $(2S(n, k) + \lceil (n - \lfloor k/2 \rfloor) \lg 3 \rceil + o(n))$ bits with $2T(n, k) + O(1)$ query time.*

Encoding sorted 3-sided Top-k queries on $2 \times n$ array. We now consider the encoding for answering sorted 3-sided Top-k queries on $2 \times n$ array A, when sorted 1-sided Top-k encodings for the two rows A_1 and A_2 are given. Similar to the unsorted case, it is enough to show how to support the sorted 1-sided Top-k queries on A. We first give an encoding that uses less space for small values of k, and later give another encoding that is space-efficient for large values of k

▶ **Theorem 3.** $(*)$ *Let A be a $2 \times n$ array. For $1 < k \leq n$, if we have $S(n, k)$-bit encoding which can answer the sorted 1-sided Top-k queries for each row in $T(n, k)$ time, then we can encode A using $2S(n, k) + kn$ bits to support sorted 3-sided Top-k queries in $2T(n, k)$ time.*

The additional space used in Theorem 3 is close to the optimal for $k = 1, 2$ or 3, but increases with k. Using similar ideas, one can obtain another encoding that uses $2n \lg(k + 1)$ bits, in addition to the individual row encodings. In the following, we give an alternative encoding whose additional space is independent of k.

▶ **Theorem 4.** $(*)$ *Let A be a $2 \times n$ array. For $1 < k \leq 2n$, suppose we have $S(n, k)$-bit encoding which can answer the sorted 1-sided Top-k queries. Then we can answer sorted 3-sided Top-k queries on A using $(2S(n, k) + \lceil 2n \lg 3 \rceil + o(n))$ bits.*

If we use the $(n \lg k + O(n))$-bit Top-k encoding of a 1D array by Grossi et al. [10] that can answer sorted 1-sided Top-k query on 1D array of size n in $O(k \lg k)$ time, then we obtain 3-sided unsorted (or sorted) Top-k encodings on A using $2n \lg k + O(n)$ bits. Furthermore for unsorted queries, we can answer the query in $O(k \lg k)$ time. Also if one can construct an encoding for answering 1-sided unsorted (or sorted) Top-k queries on individual rows

[5] Proofs of the results marked with $(*)$ is omitted due to space limitation, and can be found in the extended version [12].

in $C(n,k)$ time, we can construct an encoding for answering 3-sided unsorted (or sorted) Top-k queries in $O(C(n,k) + n \lg k)$ time as follows. Since it is enough to know the answers of unsorted (or sorted) Top-$k(1,2,1,i)$ queries for $1 \le i \le n$ to construct, we maintain a min-heap and insert the first k values (in column-major order) to the heap and sort them using $O(k \lg k)$ time. After that, whenever we insert a next value in A in column-major order, we delete the smallest value in the heap, using $O((n-k)\lg k)$ time in total. The data structure of Lemma 1 can be constructed in $O(n)$ time [9].

3 Encoding 4-sided Top-k queries on $2 \times n$ array

In this section, we describe the encoding of sorted 4-sided Top-k on $2 \times n$ array A, assuming that sorted 2-sided Top-k encodings on A_1 and A_2 are given. We show that we can encode sorted 4-sided Top-k queries on A using at most $2S(n,k) + 4n$ bits if sorted 2-sided Top-k queries on each row can be answered in $T(n,k)$ time using $S(n,k)$ bits. By extending this encoding to an $n \times m$ array, we obtain an $mS(n,k) + 2nm(m-1)$-bit encoding for answering the 4-sided Top-k query on $m \times n$ array. Note that if we use Gawrychowski and Nicholson's $(\lg \binom{(k+1)n}{n} + o(n))$-bit optimal encoding for sorted 2-sided Top-k queries on a 1D array [7], we obtain an encoding that takes $(m \lg \binom{(k+1)n}{n} + 2nm(m-1) + o(n))$ bits for answering 4-sided Top-k queries. This improves upon the trivial $O(mn \lg n)$-bit encoding when $m = o(\lg n)$, and comparing with Jo et al.'s [11] $(m^2 \lg \binom{(k+1)n}{n} + m \lg m + o(n))$-bit encoding, our encoding takes less space than in all cases when $k > 1$. Finally for $2 \times n$ array, we describe a data structure for answering Top-k queries in $O(k^2 + kT(n,k))$ time using $2S(n,k) + (4k+7)n + ko(n)$ bits, which supports Top-k queries in efficient time, and for small constant k ($2 \le k < 160$), this data structure takes less space than constructing a data structure of Grossi et al. [10] on the array of size $2n$ which stores the values in A in column-major order.

We first define a binary DAG D_A^k on A, which generalizes the binary DAG defined by Jo et al. to answer Top-2 queries on A [11]. Then we show how to encode D_A^k using $4n$ bits, to answer the sorted 4-sided Top-k queries on A. Every node p in D_A^k is labeled with some closed interval $p = [a,b]$, where $1 \le a, b \le n$. We use Top-$k(p)$ to refer to the sorted Top-$k(1,2,a,b,A)$ query. For a node $p = [a,b]$ in D_A^k and $1 \le i \le k$, let (p_r^i, p_c^i) be the position of the i-th largest element in $A[1,2][a \dots b]$. Now we define D_A^k as follows (see Figure 1 for an example.).

1. The root of D_A^k is labeled with the range $[1,n]$.
2. A node $[a,b]$ does not have any child node (i.e leaf node) if $2(b-a+1) \le k$.
3. Suppose there exists a non-leaf node $p = [a,b]$ in D_A^k, and let a' and b', where $a \le a' \le b' \le b$, be the leftmost and rightmost column indices among the answers of Top-$k(p)$, respectively. If $a < b'$, then the node p has a node $[a, b'-1]$ as a left child. Similarly, if $a' < b$, the node p has a node $[a'+1, b]$ as a right child.

The following lemma states some useful properties of D_A^k. All the statements in the lemma can be proved by simply extending the proofs of the lemmas in [11].

▶ **Lemma 5** ([11]). *Let A be a $2 \times n$ array. For any two distinct nodes $p = [a_p, b_p]$ and $q = [a_q, b_q]$ in D_A^k, following statements hold.*

(i) *Top-$k(p) \ne$ Top-$k(q)$ (i.e., any two distinct nodes have different Top-k answers).*

(ii) *$p \subset q$ if and only if p is descendant of q.*

(iii) *For any interval $[a,b]$ with $1 \le a \le b \le n$, there exists a unique node $p_{[a,b]}$ in D_A^k such that $[a,b] \subset p_{[a,b]}$, and any descendant of $p_{[a,b]}$ does not contain $[a,b]$. Furthermore, for such a node $p_{[a,b]}$, Top-$k([a,b]) =$ Top-$k(p_{[a,b]})$.*

A_1	1	21	17	12	20	3	15	11	10
A_2	6	5	16	14	19	2	18	4	7

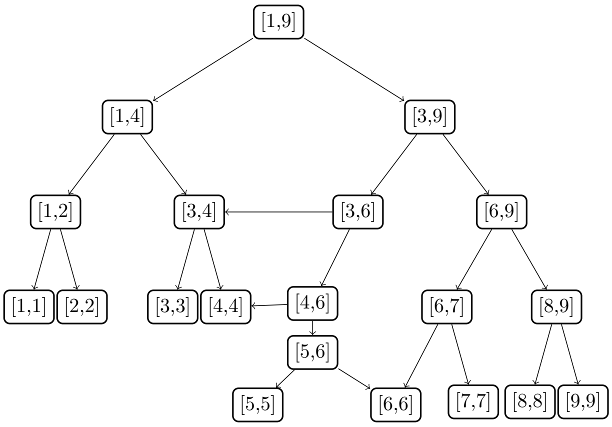

Figure 1 $2 \times n$ array A and the DAG D_A^3.

By Lemma 5(iii), if the DAG D_A^k and the answers for each sorted 2-sided Top-k queries corresponding to all the nodes in D_A^k are given, then we can answer any sorted Top-$k(1, 2, a, b, A)$ query by finding the corresponding node in $p_{[a,b]}$ in D_A^k.

Now we describe how to encode D_A^k to answer the Top-$k(p)$ query for each node $p \in D_A^k$. Our encoding of D_A^k uses a different approach from the encoding of Jo et al. [11], which encodes by traversing D_A^2 in level order. We say that a node $p = [a, b]$ *picks* the position (x, y) if we store the information that (x, y) is the position of the i-th largest element in $A[1, 2][a \dots b]$, for some $i \leq k$. To encode the DAG D_A^k, we traverse its nodes in a *modified level order*, which we describe later. While traversing the nodes of D_A^k in the modified level order, we classify the nodes as *visited*, *half-visited*, or *unvisited*. All the nodes are initially unvisited, and the traversal continues until all the nodes in D_k^A are visited. During the traversal of unvisited or half-visited node, we may *pick* a position whose column index is contained in that node (under some conditions, described later). Whenever we pick a position, we store one bit of information to resolve some of the queries. We bound the overall additional space to $4n$ bits by showing that each position in A is picked at most twice. For two nodes $p_i = [a_i, b_i]$ and $p_j = [a_j, b_j]$ with $p_i \not\subset p_j$ and $p_j \not\subset p_i$, we say the node p_i *preceeds* the node p_j if $a_i < a_j$.

When the traversal starts at the root node $[1, n]$, we pick all positions which are answers to the Top-$k(1, n, A)$ query. Since we know the answers to the Top-$k(1, n, A_1)$ and Top-$k(1, n, A_2)$ queries, the positions that are picked at the root can be encoded using a k-bit sequence $a_1 \dots a_k$ where a_i represents the row index of the i-th largest element in $A[1, 2][1 \dots n]$, for $1 \leq i \leq k$. From the definition of D_A^k, if the label of a node p and the answers of the Top-$k(p)$ query are given, then it is easy to compute the labels of the children of p.

Since it is trivial to answer the Top-k query at a leaf node, we only focus on the non-leaf nodes. Suppose that we traverse to a non-leaf node $p = [a, b]$, and let q be one of its parent nodes (note that a node can have multiple parents in a DAG). Note that $1 \leq |\text{Top-}k(q) - \text{Top-}k(p)| \leq 2$, since p contains all the Top-k answers of q except one or both positions from the column $a - 1$ or from the column $b + 1$. We first consider the case when $|\text{Top-}k(q) - \text{Top-}k(p)| = 1$ (this also includes the case when there exists another parent node q' of p such that $|\text{Top-}k(q) - \text{Top-}k(p)| = 1$ and $|\text{Top-}k(q') - \text{Top-}k(p)| = 2$). In this case, traversal visits the node p only once in modified level order, and picks at most one position at node p. Let Top-$k(q)$ − Top-$k(p) = \{(q_r^{k'}, q_c^{k'})\}$ for some $k' \leq k$. From the construction of D_A^k and Lemma 5(ii), it is clear that $(p_r^\ell, p_c^\ell) = (q_r^\ell, q_c^\ell)$ if $\ell < k'$; $(p_r^\ell, p_c^\ell) = (q_r^{\ell+1}, q_c^{\ell+1})$ if $k' \leq \ell < k$; and Top-$k(p)$ − Top-$k(q) = \{(p_r^k, p_c^k)\}$. Therefore, if the answers of the Top-$(k-1)(p)$ query are composed of f_p positions from the first row and $s_p = (k - 1 - f_p)$ positions from the second row, then we can find (p_r^k, p_c^k) by comparing $(f_p + 1)$-th largest element in A_1 and $(s_p + 1)$-th largest element in A_2 using Top-$k(a, b, A_1)$ and Top-$k(a, b, A_2)$ queries, (we define the position of these elements as the *first-candidates* of node p), and choosing the position with larger element. Note that if the answers of the Top-$(k-1)(p)$ query contains all positions in $A[1][a \dots b]$ or $A[2][a \dots b]$, there is no first-candidate of node p at the first or second row respectively. In this case we do not pick any positions at node p. Now suppose that $(1, x)$ and $(2, y)$ are the first-candidates of node p, and without loss of generality suppose $A[1][x] > A[2][y]$, and hence $(p_r^k, p_c^k) = (1, x)$. Then we consider the following cases.

1. If both $(1, x)$ and $(2, y)$ is not picked in the former nodes in D_A^k in modified level order, we pick $(1, x)$.
2. Suppose $(1, x)$ or $(2, y)$ is already picked by a visited or half-visited node $p' = [a', b']$. Then we pick $(1, x)$ at node p if and only if for all such p' does not contain both x and y.

Suppose that Top-$(k-1)(p)$ is given and one of the first-candidates is picked at node p. Then we can store its information using one bit, by representing the row index of the first-candidate picked at p.

Now consider the case $|$Top-$k(q)$ − Top-$k(p)| = 2$, and let Top-$k(q)$ − Top-$k(p) = \{(q_r^{k'}, q_c^{k'}), (q_r^{k''}, q_c^{k''})\}$ for some $k' < k'' \leq k$. In this case, the traversal visits the node p twice in modified level order, and picks at most two positions at node p. From the construction of D_A^k and Lemma 5(ii), it is clear that $(p_r^\ell, p_c^\ell) = (q_r^\ell, q_c^\ell)$ if $\ell < k'$; $(p_r^\ell, p_c^\ell) = (q_r^{\ell+1}, q_c^{\ell+1})$ if $k' \leq \ell < k''$; $(p_r^\ell, p_c^\ell) = (q_r^{\ell+2}, q_c^{\ell+2})$ if $k'' \leq \ell < k - 1$; and Top-$k(p)$ − Top-$k(q) = \{(p_r^{k-1}, p_c^{k-1}), (p_r^k, p_c^k)\}$. Therefore, if the answers of Top-$(k-2)(p)$ query are composed of f_p and $s_p = (k - 2 - f_p)$ positions in the first and the second row respectively, we can find (p_r^{k-1}, p_c^{k-1}) by comparing $(f_p + 1)$-th largest element in A_1 and $(s_p + 1)$ in A_2 using Top-$k(a, b, A_1)$ and Top-$k(a, b, A_2)$ query (we again define the position of these elements as the *first-candidates* of node p), Suppose that $(1, x)$ and $(2, y)$ are first-candidates of node p, and without loss of generality suppose $A[1][x] > A[2][y]$, and hence $(p_r^{k-1}, p_c^{k-1}) = (1, x)$. In this case, we first pick $(1, x)$ or do not pick anything at node p by the procedure described above, when we first traverse p in modified level order. When we visit p for the second time, we can find (p_r^k, p_c^k) by comparing $A_2[y]$ with the $(f_p + 2)$-th largest element in A_1 (we define the positions of these elements as the *second-candidates* of node p), and choose the position with the larger element. Note that if the answers of the Top-$(k-1)(p)$ query contains all positions in $A[1][a \dots b]$ or $A[2][a \dots b]$, there is no second-candidate of node p at the first or second row respectively. In this case we do not pick any positions at node p during the second visit of p. Again, suppose that $(1, x')$ and $(2, y)$ are the second-candidates of node p and without loss of generality suppose $A[1][x'] < A[2][y]$, and hence$(p_r^k, p_c^k) = (2, y)$. Then we consider the following cases.

1. If both $(1, x')$ and $(2, y)$ is not picked in the former nodes in D_A^k in the modified level order, we pick $(2, y)$.
2. Suppose $(1, x')$ or $(2, y)$ is already picked by the visited or half-visited $p'' = [a'', b'']$. Then we pick $(2, y)$ at node p if and only if for all such p'' does not contain both x' and y.

Note that if Top-$(k-2)(p)$ is given and $(1, x)$ is picked at node p, we can store its information using one bit, by representing a row index of first-candidate picked at p. Similarly, if Top-$(k-1)(p)$ is given and $(2, y)$ is picked at node p, we can store its information using one more bit.

Now we describe the algorithm to traverse the nodes in D_A^k in the modified level order. In modified level order, for any two nodes $p = [i, j]$ and $p' = [i', j']$, we traverse p prior to p' if and only if all column indices of p''s first or second candidates are contained in p. Furthermore by the procedure described above, we do not pick any position at p' in this case if there exists a position which is the first or second candidate of both p and p'. In the DAG, the level of the node p, denoted by $l(p)$, is defined as the number of edges in the longest path from root to p.

1. Mark the root of D_A^k as visited, and add its children into *visit-list*, which is an ordered list such that for two nodes p and q in visit-list, p comes before q in visit-list if and only if $l(p) < l(q)$ or $l(p) = l(q)$ and p precedes q in the DAG.
2. Find the leftmost unvisited or half-visited node p from visit-list which satisfies one of the following conditions (without loss of generality, assume that $x \leq y$).
   - Number of first or second candidates of p is less than 2.
   - First or second candidates of p are $(1, x)$ and $(2, y)$, and there exists no node p' in visit-list such that (a) $p \subset p'$, or (b) p' precedes p and $x \in p'$, or (c) p precedes p' and $y \in p'$.

   Then we continue the traversal from p.
3. Let q be a parent of p. If (i) $|$Top-$k(q) -$ Top-$k(p)| = 1$, or (ii) $|$Top-$k(q) -$ Top-$k(p)| = 2$ and p is half-visited, or (iii) the number of first or second candidates of p is less than 2, then mark p as visited, delete p from the visit-list, and insert p's children (if any) to visit-list. If none of these three conditions hold, then mark p as half-visited.
4. Repeat Steps 2 and 3 until all the nodes in D_A^k are marked as visited.

For example we traverse the nodes of D_A^3 in Figure 1 as: $[1, 9] \to [1, 4] \to [1, 4] \to [3, 9] \to [1, 2] \to [1, 2] \to [3, 6] \to [6, 9] \to [6, 9] \to [1, 1] \to [2, 2] \to [3, 4] \to [4, 6] \to [6, 7] \to [8, 9] \to [8, 9] \to [3, 3] \to [4, 4] \to [5, 6] \to [8, 8] \to [9, 9] \to [5, 5] \to [6, 6]$. During the traversal (in the above order), the position(s) picked at each node are: $\{(1, 2), (1, 5), (2, 5)\} \to (1, 3) \to (2, 3) \to (2, 7) \to (1, 1) \to (1, 2) \to \epsilon \to (1, 7) \to (1, 8) \to \epsilon \to \epsilon \to (2, 4) \to \epsilon \to (1, 6) \to (1, 9) \to (2, 9) \to \epsilon \to \epsilon \to \epsilon \to \epsilon \to \epsilon \to \epsilon \to \epsilon$, respectively ($\epsilon$ indicates that no position is picked). Now we bound the total number of picked positions during the traversal of D_A^k.

▶ **Lemma 6.** (∗) *Given $2 \times n$ array $A[1, 2][1 \ldots n]$ and DAG D_A^k, any position in A is picked at most twice while we traverse all nodes in D_A^k in the modified level order.*

Now we prove our main theorem. To obtain a construction time of our encoding, we first introduce a lemma which states a maximum number of nodes in D_A^k.

▶ **Lemma 7.** (∗) *Given $2 \times n$ array A and DAG D_A^k, there are at most $6kn$ nodes in D_A^k.*

▶ **Theorem 8.** (∗) *Given a $2 \times n$ array A, if there exists an $S(n, k)$-bit encoding to answer sorted 2-sided Top-k queries on a 1D array of size n in $T(n, k)$ time and such encoding can be constructed in $C(n, k)$ time, then we can encode A in $2S(n, k) + 4n$ bits using $O(C(n, k) + k^2n^2 + knT(n, k))$ time.*

We can obtain an encoding for answering sorted 4-sided Top-k queries on an $m \times n$ array by extending the encoding of a $2 \times n$ array described in Theorem 8 as stated below.

▶ **Theorem 9.** (∗) *Given an $m \times n$ array A, if there exists an $S(n,k)$-bit encoding to answer sorted 2-sided* Top-k *queries on a 1D array of size n, then we can encode A in $mS(n,k) + 2nm(m-1)$ bits, to support sorted 4-sided* Top-k *queries on A.*

Data structure for answering 4-sided Top-k queries on $2 \times n$ array. The encoding of Theorem 8 shows that $4n$ bits are sufficient for answering Top-k queries whose range spans both rows, when encodings for answering sorted 2-sided Top-k queries for each row are given. However, this encoding does not support queries efficiently (takes $O(k^2n^2 + knT(n,k))$ time) since we need to reconstruct all the nodes in D_A to answer a query (in the worst case). We now show that the query time can be improved to $O(k^2 + kT(n,k))$ time if we use $(4k+7)n + ko(n)$ additional bits. Note that if we simply use the data structure of Grossi et al. [10] (which takes $44n \lg k + O(n \lg\lg k)$ bits to encode a 1D array of length n to support Top-k queries in $O(k)$ time) on the 1D array of size $2n$ obtained by writing the values of A in column-major order, we can answer Top-k queries on A in $O(k)$ time using $88n \lg k + O(n \lg\lg k)$ additional bits. Although our data structure takes more query time and takes asymptotically more space, it uses less space for small values of k (note that $4k + 7 < 88 \lg k$ for all integers $2 \le k < 160$) when n is sufficiently large. We now describe our data structure.

We first define a graph $G_{12} = (V(G_{12}), E(G_{12}))$ on A as follows. The set of vertices $V(G_{12}) = \{1, 2, \dots n\}$, and there exists a edge $(i,j) \in E(G_{12})$ if and only if (i) $i < j$ and $A[1][i] < A[2][j]$, (ii) there are at most $k-1$ positions in $A[1,2][i \dots j]$ whose corresponding values are larger than both $A[1][i]$ and $A[2][j]$, and (iii) there is no vertex $j' > i$ that satisfies the condition (ii) such that $A[1][j] < A[2][j'] < A[2][j]$. We also define a graph G_{21} on A which is analogous to G_{12}. Each of the graphs G_{12} and G_{21} have n vertices and at most n edges. Also for any vertex $v \in V(G_{12})$ (resp., $V(G_{21})$), there exists at most one vertex v' in G_{12} (resp., G_{21}) such that v is incident to v' and $v < v'$. We now show that G_{12} (thus, also G_{21}) is a *k-page graph*, i.e. there exist no $k+1$ edges $(i_1, j_1) \dots (i_{k+1}, j_{k+1}) \in E(G_{12})$ such that $i_1 < i_2 \dots < i_{k+1} < j_1 < j_2 \dots < j_{k+1}$.

▶ **Lemma 10.** *Given $2 \times n$ array A, a graph G_{12} on A is k-page graph.*

Proof. Suppose that there are $k+1$ edges $(i_1, j_1) \dots (i_{k+1}, j_{k+1}) \in E(G_{12})$ such that $i_1 < i_2 \dots < i_{k+1} < j_1 < j_2 \dots < j_{k+1}$, and for $1 \le t \le k+1$, let i_t be a position of the minimum element in $A_1[i_1 \dots i_{k+1}]$. Then by the definition fo G_{12}, there are at least k positions $(1, i_{t+1}), \dots, (1, i_{k+1}), (2, j_1), \dots, (2, j_{t-1})$ in $A[1,2][i_t \dots j_t]$ whose corresponding values in A are larger than both $A[1][i_t]$ and $A[2][j_t]$, which contradicts the definition of G_{12}. ◀

From the above lemma and the succinct representation of k-page graphs of Munro and Raman [15] (with minor modification as described in [6]), we can encode G_{12} and G_{21} using $(4k+4)n + ko(n)$ bits in total, and for any vertex v in $V(G_{12}) \cup V(G_{21})$, we can find a vertex with the largest index which incident to v in $O(k)$ time. Also to compare the elements in the same column, we maintain a bit string $P_A[1 \dots n]$ of size n such that for $1 \le i \le n$, $P_A[i] = 0$ if and only if $A[1][i] > A[2][i]$. Finally, for G_{12} (resp., G_{21}), we maintain another bit string $Q_{12}[1 \dots n-1]$ (resp., $Q_{21}[1 \dots n-1]$) such that for $1 \le i \le n-1$, $Q_{21}[i] = 1$ (resp., $Q_{21}[i] = 1$) if and only if all elements in $A_2[i+1 \dots n]$ (resp., $A_1[i+1 \dots n]$) are smaller than $A[1][i]$ (resp., $A[2][i]$). We now show that if there is an encoding which can answer the sorted Top-k queries on each row, then the encoding of G_{12}, G_{21}, and the additional arrays defined above are enough to answer 4-sided Top-k queries on A.

▶ **Theorem 11.** (∗) *Given a $2 \times n$ array A, if there exists an $S(n,k)$-bit encoding to answer sorted 2-sided* Top-k *queries on a 1D array of size n in $T(n,k)$ time, then there is a $2S(n,k) + (4k+7)n + ko(n)$-bit data structure which can answer* Top-k *queries on A in $O(k^2 + kT(n,k))$ time.*

4 Lower bounds for encoding range Top-k queries on $2 \times n$ array

In this section, we consider the lower bound on space for encoding a $2 \times n$ array A to support Top-k queries, when $k > 1$. Specifically for $1 \le i \le j \le n$, we consider to lower bound on extra space for answering i) unsorted and sorted 3-sided Top-$k(1,2,1,i)$ queries, assuming that we have access to the encodings of the individual rows of A that can answer unsorted or sorted 1-sided Top-k queries and ii) sorted 4-sided Top-$k(1,2,i,j)$ queries, assuming that we have access to the encodings of the individual rows of A that can answer sorted 2-sided Top-k queries. We show that for answering unsorted (or sorted) 3-sided Top-$k(1,2,1,i)$ queries on A, at least $1.27n - o(n)$ (or $2n - O(\lg n)$) extra bits are necessary, and for answering unsorted or sorted 4-sided Top-$k(1,2,i,j)$ queries on A, at least $2n - O(\lg n)$ extra bits are necessary.

For simplicity (to avoid writing floors and ceilings, and to avoid considering some boundary cases), we assume that k is even. (Also, if k is odd we can consider the lower bound on extra space for answering 3-sided Top-k queries as the lower bound of extra space for answering 3-sided Top-$(k-1)$ queries – it is clear that former one requires more space.) For both unsorted and sorted query cases, we assume that all elements in A are distinct, and come from the set $\{1, 2, \ldots 2n\}$; and also that each row in A is sorted in the ascending order. Finally, for $1 \le \ell \le 2n$, we define the mapping $A^{-1}(\ell) = (i,j)$ if and only if $A[i][j] = \ell$.

Unsorted 3-sided Top-k query. If $n \le k/2$ we do not need any extra space since all positions are answers of unsorted Top-$k(1,2,1,i,A)$ queries for $i \le n$. If not ($n > k/2$), for $1 \le i \le n - k/2$, let U_i be a set of arrays of size $2 \times n$ such that i) for any $B \in U_i$, all of $\{1, 2 \ldots 2i\}$ are in $B[1,2][1 \ldots i]$ and each row in B is sorted in the ascending order, and ii) for any two distinct arrays $B, C \in U_i$, there exists $1 \le j \le i$ such that $\{B^{-1}(2j-1), B^{-1}(2j)\} \ne \{C^{-1}(2j-1), C^{-1}(2j)\}$. By the definition of U_i, it is easy to show that for any two distinct arrays $B, C \in U_i$, unsorted Top-$k(1,2,1,k/2+j,B) \ne$ Top-$k(1,2,1,k/2+j,C)$ if $\{B^{-1}(2j-1), B^{-1}(2j)\} \ne \{C^{-1}(2j-1), C^{-1}(2j)\}$ for some $j \le i$. We compute the size of U_i as follows. $|U_1| = 1$ since there exists only one case as $\{B^{-1}(1), B^{-1}(2)\} = \{(1,1),(2,1)\}$. For $i = 2$, we can consider three cases as $(1,2,3,4)$, $(1,3,2,4)$, or $(1,4,2,3)$ if we write the elements of $B[1,2][1,2]$ in U_2 in row-major order (note that each row is sorted in ascending order). By computing the size of U_i for $2 < i \le n - k/2$, we obtain a following theorem.

▶ **Theorem 12.** *Given a $2 \times n$ array A and encodings for answering unsorted (or sorted) 1-sided* Top-k *queries on both rows in A, at least $\lceil (n-k/2)\lg(1+\sqrt{2})\rceil - o(n) = 1.27(n - k/2) - o(n)$ additional bits are necessary for answering unsorted 3-sided* Top-k *queries on A.*

Proof. *(Sketch)* Since we need at least $\lg |U_{n-k/2}|$ bits of extra space for answering unsorted Top-k queries which span both rows, we only need to compute the size of $U_{n-k/2}$. To compute this, for $2 < i \le n - k/2$, we construct the arrays in U_i from the arrays in U_{i-1}, and obtain the recurrence relation: $|U_i| = 3|U_{i-2}| + 2(|U_{i-1}| - |U_{i-2}|)$. Solving this gives us the stated bound. Details of the proof are omitted due to space limitation. ◀

Sorted 3-sided and 4-sided Top-k query. In this case we divide a $2 \times n$ array A into $2n/k$ blocks $A_1 \ldots A_{2n/k}$ of size $2 \times k/2$ as for $1 \le \ell \le k/2$, $A_\ell[i][j] = A[i][2(\ell-1) + j]$ and all values of A_ℓ are in $\{k(\ell-1)+1 \ldots k\ell\}$. Then for any $2 \times n$ array A and

A', sorted Top-$k(1, 2, k(i-1)/2+1, ki/2, A) \neq$ Top-$k(1, 2, k(i-1)/2+1, ki/2, A')$, and Top-$k(1, 2, 1, ki/2, A) \neq$ Top-$k(1, 2, 1, ki/2, A')$ if $A_i \neq A'_i$ for $1 \leq i \leq 2n/k$. Let S_i be the set of arrays of size $2 \times i$ such that for any $B \in S_i$, all values of B are in $\{1, 2i\}$ and both rows of B are sorted in ascending order. Since the size of S_i is same as *central binomial number*, $\binom{2i}{i}$, which is well-known as at least $4^i/\sqrt{4i}$ [13]. Therefore, at least $\lceil 2n \lg |S_{k/2}|/k \rceil \geq 2n - O(\lg n)$ bits are necessary for answering sorted Top-k queries that span both the rows, when encodings for answering sorted (or unsorted) on both rows are given.

▶ **Theorem 13.** *Given a $2 \times n$ array A, at least $2n - O(\lg n)$ additional bits are necessary for answering sorted 3-sided (resp., 4-sided) Top-k queries on A if encodings for answering unsorted (or sorted) 1-sided (resp., 2-sided) Top-k queries on both rows in A are given.*

5 Conclusion

In this paper, we proposed encodings for answering Top-k queries on 2D arrays. For $2 \times n$ arrays, we proposed upper and lower bounds on space for answering 3-sided sorted and unsorted Top-k queries. Finally, we obtained an $(m \lg \binom{(k+1)n}{n} + 2nm(m-1) + o(n))$-bit encoding for answering 4-sided sorted Top-k queries on $m \times n$ arrays. We end with the following open problems: (a) can we support 4-sided sorted Top-k queries with efficient query time on $m \times n$ arrays using less than $O(nm \lg n)$ bits when $m = o(\lg n)$? (b) is there any improved lower or upper bound for answering 4-sided sorted Top-k queries on $2 \times n$ arrays?

References

1. Omer Berkman and Uzi Vishkin. Recursive Star-Tree Parallel Data Structure. *SIAM J. Comput.*, 22(2):221–242, 1993. doi:10.1137/0222017.
2. Gerth Stølting Brodal, Andrej Brodnik, and Pooya Davoodi. The Encoding Complexity of Two Dimensional Range Minimum Data Structures. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 229–240, 2013. doi:10.1007/978-3-642-40450-4_20.
3. Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. On Space Efficient Two Dimensional Range Minimum Data Structures. *Algorithmica*, 63(4):815–830, 2012. doi:10.1007/s00453-011-9499-0.
4. Pooya Davoodi, Gonzalo Navarro, Rajeev Raman, and S. Srinivasa Rao. Encoding range minima and range top-2 queries. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 372(2016), 2014. doi:10.1098/rsta.2013.0131.
5. Johannes Fischer and Volker Heun. Finding Range Minima in the Middle: Approximations and Applications. *Mathematics in Computer Science*, 3(1):17–30, 2010. doi:10.1007/s11786-009-0007-8.
6. Pawel Gawrychowski and Patrick K. Nicholson. Encodings of Range Maximum-Sum Segment Queries and Applications. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings*, pages 196–206, 2015. doi:10.1007/978-3-319-19929-0_17.
7. Pawel Gawrychowski and Patrick K. Nicholson. Optimal Encodings for Range Top-k k , Selection, and Min-Max. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 593–604, 2015. doi:10.1007/978-3-662-47672-7_48.
8. Mordecai J. Golin, John Iacono, Danny Krizanc, Rajeev Raman, Srinivasa Rao Satti, and Sunil M. Shende. Encoding 2D range maximum queries. *Theor. Comput. Sci.*, 609:316–327, 2016. doi:10.1016/j.tcs.2015.10.012.

9 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 841–850, 2003. URL: `http://dl.acm.org/citation.cfm?id=644108.644250`.

10 Roberto Grossi, John Iacono, Gonzalo Navarro, Rajeev Raman, and S. Srinivasa Rao. Asymptotically Optimal Encodings of Range Data Structures for Selection and Top-k Queries. *ACM Trans. Algorithms*, 13(2):28:1–28:31, 2017. `doi:10.1145/3012939`.

11 Seungbum Jo, Rahul Lingala, and Srinivasa Rao Satti. Encoding Two-Dimensional Range Top-k Queries. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, pages 3:1–3:11, 2016. `doi:10.4230/LIPIcs.CPM.2016.3`.

12 Seungbum Jo and Srinivasa Rao Satti. Encoding two-dimensional range top-k queries revisited. *CoRR*, abs/1809.07067, 2018. `arXiv:1809.07067`.

13 N.D Kazarinoff. *Geometric inequalities.* New York: Random House, 1961.

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

15 J. Ian Munro and Venkatesh Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM J. Comput.*, 31(3):762–776, 2001. `doi:10.1137/S0097539799364092`.