Encoding Two-Dimensional Range Top-k Queries

Seungbum Jo[1], Rahul Lingala[2], and Srinivasa Rao Satti[3]

1 **Seoul National University, Korea**
`sbcho@tcs.snu.ac.kr`
2 **IIT Bombay, India**
`lingalarahul7@gmail.com`
3 **Seoul National University, Korea**
`ssrao@cse.snu.ac.kr`

Abstract

We consider various encodings that support range Top-k queries on a two-dimensional array containing elements from a total order. For an $m \times n$ array, with $m \leq n$, we first propose an almost optimal encoding for answering one-sided Top-k queries, whose query range is restricted to $[1 \dots m][1 \dots a]$, for $1 \leq a \leq n$. Next, we propose an encoding for the general Top-k queries that takes $m^2 \lg \binom{(k+1)n}{n} + m \lg m + o(n)$ bits. This generalizes the one-dimensional Top-k encoding of Gawrychowski and Nicholson [ICALP, 2015]. Finally, for a $2 \times n$ array, we obtain a $2 \lg \binom{3n}{n} + 3n + o(n)$-bit encoding for answering Top-2 queries.

1998 ACM Subject Classification E.1 Data Structures

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

Digital Object Identifier 10.4230/LIPIcs.CPM.2016.3

1 Introduction

Given a one-dimensional (1D) array $A[1 \dots n]$ from a total order and $1 \leq k \leq n$, 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 $A[i \dots j]$. We can extend this query to the two-dimensional (2D) array case. Given a 2D array $A[1 \dots m][1 \dots n]$, from a total order and $1 \leq k \leq mn$, the *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 $A[i \dots j][a \dots b]$. Without loss of generality, we assume that all elements in A are distinct by ordering equal elements in the lexicographic order of their positions, and also assume that $m \leq n$. If the k positions of a Top-k query are reported in sorted order of the corresponding values, we refer to the query as *sorted* Top-k query; and refer to it as *unsorted* Top-k query, otherwise. For $1 \leq i, j \leq m$ and $1 \leq a, b \leq n$, we can also classify Top-k queries on 2D array by its range as follows.

1-sided query: The query range is $[1 \dots m][1 \dots b]$.
4-sided query: The query range is $[i \dots j][a \dots b]$.

We can also consider 2-sided and 3-sided queries which correspond to the ranges $[1 \dots j][1 \dots a]$ and $[1 \dots j][a \dots b]$ respectively. We consider how to support the Top-k queries in the *encoding model* in which we do not have access to the original input array A at query time. The minimum size of an encoding is also referred to as the *effective entropy* of the input data (with respect to the queries) [7].

In the rest of the paper, we assume that for Top-k encodings, k is at most the size of the array (either 1D or 2D). Also, unless otherwise mentioned, we assume that all Top-k queries are sorted Top-k queries.

Table 1 The summary of our results for Top-k queries on $m \times n$ 2D array. The value T is given by the formula $T = \sum_{i=0}^{\min(m,k)} i!\binom{m}{i}\binom{k}{i}$.

Array size	Query range	Space	Query time
$m \times n$	one-sided	$n\lceil \lg T \rceil$ bits	-
$2 \times n$	four-sided, $k \leq 2$	$2\lg\binom{3n}{n} + 3n + o(n)$ bits	-
$m \times n$	four-sided	$O(mn \lg n)$ bits	$O(k)$
$m \times n$	four-sided	$m^2 \lg\binom{(k+1)n}{n} + m \lg m + o(n)$ bits	-

1.1 Previous Work

Encoding Top-k queries on 1D array has been widely studied in the recent years. For a 1D array $A[1 \ldots n]$, Chan and Wilkinson [4] proposed a data structure that uses $\Theta(n)$ words and answers selection queries (i.e., selecting the k-th largest element) in $O(\lg k / \lg \lg n)$ time[1]. Grossi et al. [8] considered the Top-k encoding problem, and obtained an $O(n \lg \kappa)$-bit encoding which can answer the Top-k queries for any $k \leq \kappa$ in $O(\kappa)$ time or alternately, using $O(n \lg^2 \kappa)$ bits with $O(k)$ query time. (They also considered one-sided Top-k query, they proposed $n \lg k + O(n)$-bit encoding with $O(k)$ query time.) The space usage of this encoding was improved to $O(n \lg \kappa)$ bits, maintaining the $O(k)$ query time, by Navarro et al. [10]. Recently, Gawrychowski and Nicholson [6] proposed an $(k+1)nH(1/(k+1)) + o(n)$-bit[2] encoding for Top-k queries and showed that at least $(k+1)nH(1/(k+1))(1 - o(1))$ bits are required to encode Top-k queries.

To the best of our knowledge, there are no results on range Top-k queries for 2D array with general k. For $k = 1$, the Top-k query is same as the *Range Maximum Query (RMQ)*, which has been well-studied for 1D as well as for 2D arrays. For a 2D $m \times n$ array, Brodal et al. [1] proposed an $O(nm \min(m, \lg n))$-bit encoding which answers RMQ queries in $O(1)$ time. Brodal et al. [2] improved the space bound to the optimal $O(nm \lg m)$ bits, although this encoding does not support the queries efficiently.

1.2 Our Results

For an $m \times n$ 2D array A, we first obtain an $n\lceil \lg T \rceil$-bit encoding for answering one-sided Top-k queries, where $T = \sum_{i=0}^{\min(m,k)} i!\binom{m}{i}\binom{k}{i}$. We then show that any encoding that supports Top-k queries on A must use at least $n \lg T$ bits.

Next, we observe that one can obtain an $O(mn \lg n)$-bit data structure which answers 4-sided Top-k queries on A in $O(k)$ time, by combining the results of [3] and [1]. We then propose an $m^2 \lg\binom{(k+1)n}{n} + m \lg m + o(n)$-bit encoding for 4-sided Top-k queries on A, by extending the Top-k encoding of Gawrychowski and Nicholson for 1D arrays [6].

When $k = 2$ and $m = 2$, the above encoding takes $4\lg\binom{3n}{n} + o(n) \approx 11.02n$ bits. For this case, we propose an alternative encoding which uses $2\lg\binom{3n}{n} + 3n + o(n) \approx 8.51n$ bits (and can answer the 4-sided Top-2 queries on A). All these results are summarized in Table 1.

We assume the standard word-RAM model [9] with word size $\Theta(\lg n)$.

2 Encoding one-sided range Top-k queries on two dimensional array

In this section, we consider the encoding of one-sided Top-k queries on a 2D array $A[1 \ldots m][1 \ldots n]$. We first introduce the encoding by simply extending the encoding of one-sided Top-k

[1] We use $\lg n$ to denote $\log_2 n$

[2] $H(x) = x \lg(1/x) + (1-x)\lg(1/(1-x))$

queries for 1D array proposed by Grossi et al. [8]. Next we propose an optimal encoding for one-sided Top-k queries on A.

For a 1D array $A'[1 \dots n]$, one can define another 1D array $X[1 \dots n]$ such that $X[i] = i$ for $1 \leq i \leq k$ and for $k < i \leq n$, $X[i] = X[i']$ if there exist a position $i' < i$ such that $A'[i]$ is larger than $A'[i']$ which is the k-th largest value in $A'[1 \dots i-1]$, and $X[i] = k+1$ otherwise. One can answer the Top-$k(1, i, A')$ by finding the rightmost occurrence of every element $1 \dots k$ in $X[1 \dots i]$. By representing X (along with some additional auxiliary structures) using $n \lg k + O(n)$ bits, Grossi et al. [8] obtained an encoding which supports 1-sided Top-k queries on A' in $O(k)$ time.

For a 2D array A, one can encode A to support one-sided Top-k queries by writing down the values of A in column-major order into a 1D array, and using the encoding described above – resulting in the following encoding.

▶ **Proposition 1.** *A 2D array $A[1 \dots m][1 \dots n]$ can be encoded using $mn \lg k + O(n)$ bits to support one-sided Top-k queries in $O(k)$ time.*

Now we describe an optimal encoding of A which supports one-sided Top-k queries. For 1D array $A'[1 \dots n]$, we can define another 1D array $B'[1 \dots n]$ such that for $1 \leq i \leq n$, $B'[i] = l$ if $A'[i]$ is the l-th largest element in $A'[1 \dots i]$ with $l \leq k$, and $B'[i] = k+1$ otherwise. Then we answer the Top-$k(1, i, A')$ query as follows. We first find the rightmost position $p_1 \leq i$ such that $B'[p_1] \leq k$. Then we find the positions $p_2 > p_3 \cdots > p_k$ such that for $2 \leq j \leq k$, p_j is the rightmost position in $A'[1 \dots p_{j-1} - 1]$ with $B'[p_j] \leq k - j + 1$. Finally, we return the positions $p_1, p_2, \dots, p_k$. Therefore by storing B' using $n\lceil \lg (k+1) \rceil$ bits, we can answer the one-sided Top-k queries on A'. Also we can sort $A'[p_1], \dots, A'[p_k]$ using the property that for $1 \leq b < a \leq k$, $A'[p_a] < A'[p_b]$ if and only if one of the following two conditions hold: (i) $B'[p_a] \geq B'[p_b]$, or (ii) $B'[p_a] < B'[p_b]$ and there exist $q = B'[p_b] - B'[p_a]$ positions $j_1, j_2, \dots, j_q$ such that $p_a < j_1 < \cdots < j_q < p_b$ and $B'[j_r] \leq B'[p_a]$ for $1 \leq r \leq q$.

We can extend this encoding for the one-sided Top-k queries on a 2D array A. For $1 \leq j \leq n$, we first define the elements of j-th column in A as $a_{1j} \dots a_{mj}$. Then we define the sequence $S_j = s_{1j} \dots s_{mj}$ such that for $1 \leq i \leq m$, $s_{ij} = l$ if a_{ij} is the l-th largest element in $A[1 \dots m][1 \dots j]$ with $l \leq k$ and $s_{ij} = k+1$ otherwise. Since there exist $T = \sum_{i=0}^{\min(m,k)} \binom{m}{i}\binom{k}{i} i!$ possible S_i sequences (T is the total number of ways in which we can choose i out of the m rows for new entries into the Top-k positions, summed over all possible values of i), we can store $S^A = S_1 \dots S_n$ using $n\lceil \lg T \rceil$ bits and we can answer the one-sided Top-$k(1, m, 1, j)$ queries on A by the following procedure.

1. Find the rightmost column q, for some $q \leq j$, such that S_q has $\ell > 0$ elements $s_{p_1 q}, \dots, s_{p_\ell q}$ where $s_{p_1 q} < \cdots < s_{p_\ell q} < k+1$. If $\ell = k$, we return the positions of $A[p_1][q] \dots A[p_k][q]$ as the answers of the query, and stop. Otherwise (if $\ell < k$), we return the positions of $A[p_1][q] \dots A[p_\ell][q]$, and
2. Repeat Step 1 by setting k to $k - \ell$, and j to $q - 1$.

We can return the positions in the sorted order of their corresponding values similar to the 1D array case as described above. This encoding takes less space than the encoding in the Proposition 1 since $mn \lg k = n \lg (1 + (k-1))^m = n \lg \sum_{i=0}^{m} \binom{m}{i} (k-1)^i \geq n \lg T$. The following theorem shows that the space usage of this encoding is essentially optimal for answering one-sided Top-k queries on A.

▶ **Theorem 2.** *Any encoding of a 2D array $A[1 \dots m][1 \dots n]$ that supports one-sided Top-k queries requires $n \lg T$ bits, where $T = \sum_{i=0}^{\min(m,k)} i! \binom{m}{i}\binom{k}{i}$.*

Proof. Suppose there are two distinct sequences $S^A = S_1 \dots S_i$ and $S^{A'} = S'_1 \dots S'_i$ which give one-sided Top-k encodings of 2D arrays A and A', respectively. For $1 \leq b \leq n$, if $S_b \neq S'_b$

then Top-$k(1, m, 1, b, A) \neq$ Top-$k(1, m, 1, b, A')$ by the definition of S^A and $S^{A'}$. Since for an $m \times n$ array, there are T^n distinct sequences $S^{A_1} \dots S^{A_{T^n}}$, it is enough to prove that for $1 \leq q \leq T^n$, each $S^{A_q} = S_1^q \dots S_n^q$ has an array A such that $S^A = S^{A_q}$.

Without loss of generality, suppose that all elements in A come from the set $L = \{1, \dots, mn\}$. Then we can reconstruct A from the rightmost column using S^{A_q} as follows. If $s_{jn}^q \leq k$, for $1 \leq j \leq m$, we assign the s_{jn}^q-th largest element in L to $A[j][n]$. After we assign all values in the rightmost column with $s_{jn}^q \leq k$, we discard all assigned values from L, move to $(n-1)$-th column and repeat the procedure. After we assign all values in A whose corresponding values in S^{A_q} are smaller than $k+1$, we assign the remaining values in L to remaining positions in A_q which are not assigned yet. Thus for any $1 \leq b \leq n$, if S_b^q has $\ell > 0$ elements $s_{p_1 b}, \dots, s_{p_\ell b}$ where $s_{p_1 b} < \dots < s_{p_\ell b} < k+1$, then the b-th column in A contains ℓ-largest elements in $A[1 \dots m][1 \dots b]$ by the above procedure. This shows that $S^A = S^{A_q}$. ◀

3 Encoding range Top-k queries on two dimensional array

In this section, we give an encoding which supports general Top-k queries on 2D array. For an $m \times n$ 2D array, we first introduce an $O(mn \lg n)$-bit encoding which supports Top-k query in $O(k)$ time by using the RMQ encoding of Brodal et al. [2].

▶ **Proposition 3.** *A 2D array $A[1 \dots m][1 \dots n]$ can be encoded using $O(mn \lg n)$ bits to support unsorted Top-$k(i, j, a, b, A)$ in $O(k)$ time for $1 \leq a, b \leq m$ and $1 \leq i, j \leq n$.*

Proof. We use a data structure similar to the one outlined in [3] (based on Frederikson's heap selection algorithm [5]) for answering unsorted Top-k queries in 1D array[3]. First encode A using $O(mn \lg n)$ bits to support RMQ (range maximum) queries in constant time for any rectangular range in A. This encoding also supports finding the rank (i.e., the position in sorted order) of any element in A in $O(1)$ time [1]. Next, let $x = A[x_1][x_2]$ be the maximum value in $A[i \dots j][a \dots b]$, which can be found using an RMQ query on A. Then consider the 4-ary heap obtained by the following procedure. The root of the heap is x, and its four subtrees are formed by recursively constructing the 4-ary heap on the sub-arrays $A[i \dots x_1 - 1][a \dots b]$, $A[x_1 + 1 \dots j][a \dots b]$, $A[x_1][a \dots x_2 - 1]$ and $A[x_1][x_2 + 1 \dots b]$, respectively. Now, we can find the k largest elements in the above 4-ary heap in $O(k)$ time using the algorithm proposed by Frederickson [5] (note that this algorithm only builds a heap with $O(k)$ nodes which is a connected subgraph of the above 4-ary heap). ◀

We now introduce another encoding to support Top-k queries on an $m \times n$ 2D array A. This encoding extends the optimal Top-k encoding of Gawrychowski and Nicholson [6] for a 1D array. This encoding does not support the queries efficiently. Compared to the encoding of Proposition 3, this encoding uses less space when $n = \Omega(k^m)$. We first review the Gawrychowski and Nicholson [6]'s optimal Top-k encoding for 1D array, and show how to extend this encoding to the 2D array case.

For a given 1D array $A'[1 \dots n]$, we define the sequence of arrays $S^{A'} = S_1^{A'} \dots S_n^{A'}$, where for $1 \leq j \leq n$ and $1 \leq i \leq j$, $S_j^{A'}$ is an array of size j defined as follows.

$$S_j^{A'}[i] = \begin{cases} p & \text{if there are } p\ (< k) \text{ elements larger than } A'[i] \text{ in } A'[i+1 \dots j] \\ k & \text{otherwise} \end{cases}$$

[3] Brodal et al. [3] also give another structure to answer sorted Top-k queries, with the same time and space bounds.

A_1	3	7	8	2	6	4
A_2	6	4	10	3	5	2

$S_1^{A_1}$	0					
$S_2^{A_1}$	1	0				
$S_3^{A_1}$	2	1	0			
$S_4^{A_1}$	2	1	0	0		
$S_5^{A_1}$	2	2	0	1	0	
$S_6^{A_1}$	2	2	0	2	0	0

$S_1^{A_2}$	0					
$S_2^{A_2}$	0	0				
$S_3^{A_2}$	1	1	0			
$S_4^{A_2}$	1	1	0	0		
$S_5^{A_2}$	1	2	0	1	0	
$S_6^{A_2}$	1	2	0	1	0	0

$I_1^{(1,2)}$	1					
$I_2^{(1,2)}$	2	0				
$I_3^{(1,2)}$	2	1	1			
$I_4^{(1,2)}$	2	1	1	1		
$I_5^{(1,2)}$	2	1	1	2	0	
$I_6^{(1,2)}$	2	1	1	2	0	0

$I_1^{(2,1)}$	0					
$I_2^{(2,1)}$	1	0				
$I_3^{(2,1)}$	2	1	0			
$I_4^{(2,1)}$	2	1	0	0		
$I_5^{(2,1)}$	2	2	0	1	0	
$I_6^{(2,1)}$	2	2	0	2	0	0

Figure 1 Top-k encoding of the 2D array A when $k = 2$.

See Figure 1 for an example.

If $S_j^{A'}[i] < k$, we call $A[i]$ in $A[1 \dots j]$ as *active*, otherwise $A[i]$ is *inactive* in $A[1 \dots j]$.

Gawrychowski and Nicholson [6] show that for $1 \leq i, j \leq n$, Top-$k(i, j, A')$ can be answered using $S_j^{A'}[i \dots j]$. They obtained a $\lg \binom{(k+1)n}{n} + o(n)$-bit encoding of $S^{A'}$ by representing $\delta_1^{A'} \dots \delta_{n-1}^{A'}$ (where $\delta_i^{A'} = \sum_{l=1}^{i+1} S_{i+1}^{A'}[l] - \sum_{l=1}^{i} S_i^{A'}[l]$) in unary, and compressing the sequence using the following lemma.

▶ **Lemma 4** ([11]). *Let S be a string of length n over the alphabet $\Sigma = \{1, 0\}$ containing m 1s. One can encode S using $\lg \binom{n}{m} + o(n)$ bits to access any position in S in constant time.*

Since $\sum_{i-1}^{n-1} \delta_i^{A'} \leq kn$, the unary sequence has kn zeros and n ones. The following lemma states their result for 1D arrays.

▶ **Lemma 5** ([6]). *Given a 1D array $A[1 \dots n]$, there is an encoding of A using $\lg \binom{(k+1)n}{n} + o(n)$ bits which supports Top-k queries.*

We now describe how to extend this encoding to a 2D $m \times n$ array A. For $1 \leq i \leq m$, let $A_i[1 \dots n]$ be the array of the i-th row in A. We construct Top-k encodings for the rows $A_1 \dots A_m$ using Lemma 5, and this takes $m \lg \binom{(k+1)n}{n} + o(n)$ bits. In addition, for every $1 \leq i \neq j \leq m$, we define the sequence of arrays, $I^{(i,j)} = I_1^{(i,j)} \dots I_n^{(i,j)}$ to represent S^i with respect to the elements in A_j. For $1 \leq r \leq n$, $I_r^{(i,j)}$ is an array of size r defined as follows.

$$I_r^{(i,j)}[s] = \begin{cases} p & \text{if } i > j \text{ and there are } p\ (< k) \text{ elements which are larger than } A_i[s] \text{ in } A_j[s+1 \dots r] \\ q & \text{if } i < j \text{ and there are } q\ (< k) \text{ elements which are larger than } A_i[s] \text{ in } A_j[s \dots r] \\ k & \text{otherwise (if there are } \geq k \text{ elements, in the above two cases)} \end{cases}$$

See Figure 1 for an example.

We can answer the Top-$k(i, j, a, b, A)$ queries as follows. We first define the 1D array $B[1 \dots b(j-i+1)]$ by writing down the values of $A[i \dots j][1 \dots b]$ in column-major order. Then we observe that Top-$k(i, j, a, b, A)$ can be answered using $S^B_{b(j-i+1)}[a(j-i+1)+1 \dots b(j-i+1)]$.

The following lemma shows that we can compute the values in $S^B_{b(j-i+1)}$ using $S^{A_1} \dots S^{A_m}$ and all the arrays $I_b^{(c,d)}$, for $1 \le c \ne d \le m$.

▶ **Lemma 6.** *Given a 2D array $A[1 \dots m][1 \dots n]$, for $1 \le i \le j \le m$ and $1 \le b \le n$, let $B[1 \dots q]$ be the 1D array of size $q = (j-i+1)b$ obtained by writing the elements of $A[i \dots j][1 \dots b]$ in column-major order. Also, for any $1 \le s \le q$, let (s_{row}, s_{col}) be the position corresponding $B[s]$ in A (which can be computed using $s_{col} = \lceil s/(j-i+1) \rceil$ and $s_{row} = s - (s_{col}-1) \cdot (j-i+1) + (i-1)$). Then*

$$S^B_q[s] = \min\left(k, \left(S_b^{A_{s_{row}}}[s_{col}] + \sum_{i \le \ell \le j, \ell \ne s_{row}} I_b^{(s_{row},\ell)}[s_{col}]\right)\right).$$

Proof. It is enough to count the number of elements in B (i.e., in $A[i \dots j][a \dots b]$) which are larger than $B[s]$ (i.e., $A[s_{row}][s_{col}]$) in $B[s+1 \dots q]$ (i.e., the corresponding elements in A). Let L be the set of these elements. If $|L| \ge k$, then $S^B_q[s] = k$. In the following, we describe how to compute $S^B_q[s]$ when $|L| < k$.

From the definition of $S_b^{A_{s_{row}}}$, it follows that the number of elements in L which are in row s_{row} is $S_b^{A_{s_{row}}}[s_{col}]$.. Also, for any row $\ell \ne s_{row}$, $I_b^{(s_{row},\ell)}[s_{col}]$ is the number of elements in L that belong to row ℓ. From all these values, we can compute $|L|$. ◀

By Lemma 6, we can answer the Top-k queries on A using the Top-k encodings of all the rows $A_1, \dots, A_m$, together with all the arrays $I^{(i,j)}$, for all $1 \le i \ne j \le m$. Since we can recover the order of all active elements in the prefix of i-th row using S^{A_i} [6], we can decode $I_p^{(i,j)}$ using $I_{p-1}^{(i,j)}$ and $\gamma_p^{ij} = \sum_{l=1}^{p} I_p^{(i,j)}[l] - \sum_{l=1}^{p-1} I_{p-1}^{(i,j)}[l]$ by the following procedure, for $p > 1$.

1. Append 0 to $I_{p-1}^{(i,j)}$. Let this array be $J_{p-1}^{(i,j)}$.
2. Find the positions of $\gamma_{p-1}^{(i,j)}$ smallest active values in $A_i[1 \dots p]$ using S^{A_i}, and increase the values of $J_{p-1}^{(i,j)}$ in these positions by 1.

Therefore, using $I_1^{(i,j)}$, and $\gamma_2^{(i,j)}, \dots, \gamma_n^{(i,j)}$, we can encode $I^{(i,j)}$. Since the sum $\sum_{\ell=2}^{\ell=n} \gamma_\ell^{(i,j)}$ is at most kn, we can encode all the arrays $I^{(i,j)}$ (for all possible $i \ne j$) using $m(m-1)\lg\binom{(k+1)n}{n} + o(n)$ bits (by converting $\gamma_\ell^{(i,j)}$'s into unary, as in the encoding of Lemma 5). Also, to encode $I_1^{(i,j)}$ for $i < j$ (note that if $i > j$, $I_1^{(i,j)}$ is always 0), we need to store the ordering of all elements in the first column, which takes $m \lg m$ bits. This gives a proof of the following theorem.

▶ **Theorem 7.** *Given a 2D array $A[1 \dots m][1 \dots n]$, there is an encoding of A using $m^2 \lg\binom{(k+1)n}{n} + m \lg m + o(n)$ bits which can answer the Top-k queries.*

4 Encoding range Top-2 queries on $2 \times n$ array

In this section, we consider a special case of Top-k encodings for 2D arrays when the array has only two rows, and $k = 2$. Note that for these parameter values, Theorem 7 gives an encoding of size $4\lg\binom{3n}{n} + o(n) \approx 11.02n$ bits. We describe an alternative approach which results in an encoding of size $2\lg\binom{3n}{n} + 3n + o(n) \approx 8.51n$ bits.

For $i \in \{1, 2\}$, let A_i be the array $[a_{i1}, \dots, a_{in}]$ of size n constituting the i-th row of A. We maintain Top-k encodings for A_1 and A_2, which enable us to support the Top-k queries on the individual rows. To support queries that span both the rows, we store an auxiliary structure of size at most $3n$ bits.

We construct a weighted DAG, D_A, such that each node in D_A is labeled with a *range* $[a, b]$, where $1 \le a \le b \le n$, and has a *weight* $w([a, b]) \in \{1, 2\}$. In the rest of this section, we

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

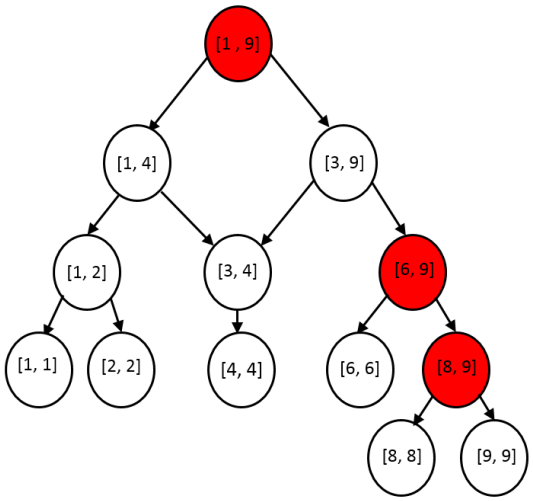

Figure 2 $2 \times n$ array A and the DAG D_A. Nodes with weight 2 are colored red, while those with weight 1 are not colored.

use the notation Top-2($[a, b]$) to refer to the query Top-2$(1, 2, a, b, A)$. We also use (i, a) to denote the position in the i-th row and a-th column in A. Now we define D_A as follows.

1. The root of D_A is labeled with the range $[1, n]$, and $w([1, n]) = 2$.
2. If $a = b$, then $[a, b]$ is a leaf node in D_A, with weight $w([a, b]) = 1$.
3. Suppose there exists a non-leaf node $[a, b]$ in D_A, such that the answers to the query Top-2($[a, b]$) are (i, a') and (j, b'), for some $1 \leq i, j \leq 2$ and $a \leq a' \leq b' \leq b$. Then the at most two children of the node $[a, b]$ are $[a, b' - 1]$and $[a' + 1, b]$.
   - **Case 1.** If $a' = b'$ and $a < b' - 1$, $w([a, b' - 1]) = 2$.
   - **Case 2.** If $a' = b'$ and $a' + 1 < b$, $w([a' + 1, b]) = 2$.
   - **Case 3.** In all other cases, $w([a, b' - 1]) = w([a' + 1, b]) = 1$.

See Figure 2 for an example. Note that a node can have at most two parents since each end point of the interval corresponding to a node can be shared by exactly one of its children. If the two parents of a node belong to two different cases, then the weight of the child node is set to be the smaller of the weights set in the two cases. For example, in Figure 2, the node $[3, 4]$ belongs to Case 3 through the parent node $[1, 4]$, and belongs to Case 1 through the parent $[3, 9]$. Hence, its weight is set to 1. Also, not all intervals of the form $[a, a]$ need to appear as leaves in D_A (eg., $[3, 3]$ in Figure 2).

From the construction of D_A, one can observe that if there is a node $[a, b]$ in D_A, with $1 < a \leq b < n$, then the columns $a - 1$ and $b + 1$ both contain at least one element that is larger than the second largest elements in the sub-array $A[1 \ldots 2][a \ldots b]$. From this observation, it follows that given any two distinct nodes x and y in D_A, the answers to the queries Top-2(x) and Top-2(y) are distinct (if there are two distinct nodes $[a, b]$ and $[a', b']$ with $b < b'$ such that Top-2($[a, b]$) = Top-2($[a', b']$), then Top-2($[a, b + 1]$) = Top-2($[a', b']$), contradicting the fact that Top-2($[a, b]$) $\neq$ Top-2($[a, b + 1]$). The case when $b > b'$, $a > a'$ or $a < a'$ is analogous). In addition, we use the following property of D_A in proving lemmas in this section.

▶ **Proposition 8.** *Let A be a $2 \times n$ array and D_A be its corresponding weighted DAG. For any distinct two nodes p and q in D_A, $p \subset q$ if and only if p is descendant of q.*

Proof. From the construction of D_A, it is the case that if p is a descendant of q, then $p \subset q$. Now, suppose $p \subset q$ and $p = [a_p, b_p]$ is not descendant of q. Then there exists a node q' which is a descendant of q such that $p \subset q'$, but no child of q' contains p. Since neither of the children of q' contain p, both column positions of Top-2(q') must belong to p (otherwise, at least one of the children of q' would contain p). But this would imply that Top-2(q') = Top-2(p), which leads to a contradiction since every node in D_A has distinct Top-2 answers. ◀

Furthermore, the following lemma shows that D_A contains all distinct answers for Top-2$([a, b])$, for $1 \leq a \leq b \leq n$ (in other words, the answers to any Top-2$([a, b])$ query on A are same as the answers to the Top-2 query on some node in the DAG).

▶ **Lemma 9.** *Let A be a $2 \times n$ array. For $1 \leq a \leq b \leq n$, for any interval $[a, b]$, there exist a node p in D_A such that Top-2$([a, b])$ = Top-2(p).*

Proof. We first show that there exists a unique p such that p contains the interval $[a, b]$ and none of the children of p (fully) contain $[a, b]$. We then show that the Top-2$([a, b])$ = Top-2(p).

Since the root in D_A contains all columns in A, it is easy to see that there exists at least one node $p = [a_p, b_p]$ in D_A such that $[a, b] \subset p$ but no child of p contains $[a, b]$. Suppose that there exists another node $p' = [a'_p, b'_p]$ such that $[a, b] \subset p'$ but no child of p' contains $[a, b]$. From Proposition 8, it follows that $p \not\subset p'$ and $p' \not\subset p$ (otherwise, one of them would be a descendant of the other, contradicting the conditions on p and p'). Now, suppose that $a_p < a'_p < b_p < b'_p$ (the case when $a'_p < a_p < b'_p < b_p$ is analogous). Then there exists a column $c < a'_p$ such that p has a child node $[c, b_p]$ which contains $[a, b]$ by the property of D_A (note that $a'_p \leq a \leq b \leq b_p$), contradicting the fact that p does not have such a child. This shows that there is a unique such p in D_A.

Now we claim that Top-2$([a, b])$ = Top-2(p). Suppose that there exist a $c \notin [a, b]$ in p such that column c contains at least one of the answers to Top-2(p). Also without loss of generality, we assume that $c < a$ (the case when $c > b$ can be handled in a similar way). Then by the property of D_A, p has a child $[c + 1, b_p]$ which still contains $[a, b]$, contradicting the fact that p does not have such a child. ◀

The following lemma shows that for any node $p = [a, b]$ in D_A, we can answer the query Top-2(p) using $w(p)$ additional bits if we know the answers to the Top-2 query on the parent node of p, and also the answers to the queries Top-2(a, b, A_1) and Top-2(a, b, A_2).

▶ **Lemma 10.** *Let A be a $2 \times n$ array. Given a non-root node $p = [a_p, b_p]$ in D_A, and its parent node $q = [a_q, b_q]$, if we know the answers to the query Top-2(q), then using the Top-2 encodings of A_1 and A_2 along with $w(p)$ additional bits, we can answer the query Top-2(p).*

Proof. If p is a leaf node (i.e., if $a_p = b_p$), we need $w(p) = 1$ extra bit to compare $A_1[a_p]$ and $A_2[a_p]$. If not, let (i_1, j_1) and (i_2, j_2), with $j_1 \leq j_2$, be the answers to the query Top-2(q). Also, for $i \in \{1, 2\}$, let f_i and s_i be the positions of the first and the second maxima, respectively, in the i-th row, $A_i[a_p \ldots b_p]$. Then we can answer the query Top-2(p) as follows. Without loss of generality, assume that $a_q < a_p$.

Case 1. $j_1 < j_2$: In this case, the interval p contains $f_{i_2} = j_2$, and this is the position of the maximum value in p. If $i_2 = 1$ ($i_2 = 2$), we can find the second maximum in p by comparing the values $A_1[s_1]$ and $A_2[f_2]$ ($A_2[s_2]$ and $A_1[f_1]$); the result of this comparison can be stored with $w(p) = 1$ extra bit.

Case 2. $j_1 = j_2$**:** In this case, the interval p does not contain j_1 $(= j_2)$. Therefore, to find the maximum element in p, we store the comparison between the values $A_1[f_1]$ and $A_2[f_2]$ using 1 bit. To find the second maximum element, if $A_1[f_1] > A_2[f_2]$ $(A_1[f_1] < A_2[f_2])$, then we store the comparison the values $A_2[f_2]$ and $A_1[s_1]$ $(A_1[f_1]$ and $A_2[s_2])$ using 1 extra bit. Thus the number of required extra bits is $w(p) = 2$. ◀

The following lemma bounds the total weight of all the nodes in D_A, which in turn bounds the extra space used by the Top-2 encoding of A in addition to the Top-2 encodings of the individual rows.

▶ **Lemma 11.** *For a $2 \times n$ array A, the sum of the weights of all nodes in D_A is at most $3n$.*

Proof. Let $f(p) = (r_p^f, c_p^f)$ and $s(p) = (r_p^s, c_p^s)$ be the positions of the first and the second largest elements in Top-2(p), respectively. Also, for each column $1 \leq j \leq n$, let f_j and s_j be the positions of the first and the second maxima in A, respectively, in the j-th column. We traverse D_A in level order. Whenever we visit a node $p = [a, b]$ in D_A, if $w(p) = 2$, then we pick the two positions $f(p)$ and $s(p)$, and otherwise (if $w(p) = 1$), we pick the position $s(p)$. We now claim that for all $1 \leq j \leq n$, f_j is picked at most twice, and s_j is picked at most once, during the level-order traversal of all the nodes in D_A. It is easy to show that statement of the lemma follows from this claim.

Case 1. Visiting a node p with $w(p) = 1$: We first show that any s_j is picked at most once. For $1 \leq j \leq n$, suppose that node p is the first node (in level order) which picks s_j. Since the only case in which this happens is when Top-2$(p) = \{f_j, s_j\}$, it follows that p is the unique node in D_A that picks s_j (as mentioned earlier, distinct nodes have distinct Top-2 answers, and s_j cannot be a position in the answers for a Top-2 query unless f_j is also an answer to the same query).

We now show that any f_j is picked at most twice. Suppose we pick f_j when we visit a node $p = [a_1, b_1]$. We need to prove that there can be at most one other node that can pick f_j. Assume, on the contrary, that there are two more distinct nodes $p_2 = [a_2, b_2]$, $p_3 = [a_3, b_3]$ such that we pick f_j when we visit these nodes. Since $w(p_2) = w(p_3) = 1$ (note that if the weight of a node is 2, then f_j can be picked at most once – see Case 2 in this proof), only f_j is picked as the second largest element when we visit p_2 and p_3. Also, by the construction of D_A, f_j is not picked if we pick f_j in any ancestor or descendant of p. Therefore, p_2 and p_3 are neither ancestor nor descendant of p, and by Proposition 8, for any two distinct $q, r \in \{p, p_2, p_3\}$, $q \not\subset r$ and $q \cap r \neq \emptyset$.

Now without loss of generality, suppose that $1 \leq a_1 < a_2 < a_3 < j < b_1 < b_2 < b_3 \leq n$. Note that if $f(p)$ exists between a_3-th and b_1-th column, Top-2(p) = Top-2(p_2) = Top-2(p_3) = $\{f(p), f_j\}$ since $s(p) = s(p_2) = s(p_3) = f_j$. This leads to a contradiction since distinct nodes should have distinct Top-2 answers (for the same reason, $f(p_2)$, and $f(p_3)$ cannot exist between a_3-th and b_1-th column). Therefore, $a_1 \leq c_p^f < a_3$ and $b_1 < c_{p_3}^f \leq b_3$. Now suppose that $b_1 < c_{p_2}^f \leq b_2$ (the case when $a_2 \leq c_{p_2}^f < a_3$ is analogous). Then f_j cannot be picked when we visit the node p_3 since the value in f_j is smaller than the values in both $f(p_2)$ and $f(p_3)$. This leads to a contradiction, proving that there can be at most two nodes whose weight is 1 which pick f_j during the traversal.

Case 2. Visiting a node p with $w(p) = 2$: In this case, we prove that neither $f(p)$ nor $s(p)$ are picked by any node other than p. (Thus, in this case, both $f(p)$ and $s(p)$ are picked only once.) By the construction of D_A, neither $f(p)$ nor $s(p)$ can be picked in any ancestor of p. Also, since neither $f(p)$ nor $s(p)$ can be the second largest elements in any of the descendants of p, we can't pick either of them after visiting the node p. We now claim that there is no node q such that $p \cap q \neq \emptyset$, $p \not\subset q$ and $q \not\subset p$. By Proposition 8, if

the claim is true, p has an intersection only with its ancestors or descendants, which do not pick both $f(p)$ and $s(p)$ during the traversal.

We assume, contrary to the above claim, that for the node $p = [a, b]$, there exists a node $q = [a_q, b_q]$ such that $p \cap q \neq \emptyset$, $p \not\subset q$ and $q \not\subset p$. Also without loss of generality, suppose that $1 \leq a_q < a < b_q < b \leq n$. Now consider the node r, which is an element in the lowest common ancestor (LCA)[4] of the nodes p and q. If any answer of the Top-2(r) query does not exist in $[a_q, b]$, one of r's child is a common ancestor of the nodes p and q, contradicting the fact that r is the LCA of p and q. Therefore, both answers of Top-2(r) exist in c-th and d-th column where $a_q \leq c \leq d \leq b$. Also, both nodes p and q can exist only if $a_q \leq c < a$ and $b_q < d \leq b$, in which case, $f([c + 1, b])$ exists in d-th column. Furthermore, by the construction of D_A, $c_s^f = d$ for any node s in the path from node $[c + 1, b]$ to node p. Therefore for any parent node of p, both answers of Top-2 exist in the d-th column since $w(p) = 2$, contradicting the fact that $b_q < d \leq b$. This leads to a contradiction that such q exists. ◀

▶ **Theorem 12.** *A $2 \times n$ array A can be encoded using $2 \lg \binom{3n}{n} + 3n + o(n)$ bits, to answer Top-2 queries.*

Proof. We first encode the first and the second rows in A using $2 \lg \binom{3n}{n} + o(n)$ bits, to answer Top-2 queries on each row, using the encoding in Lemma 5. For each node p in D_A in level order, we write down a $w(p)$-length bit-string which contains the additional bits needed to answer the query Top-2(p) (as mentioned in Lemma 10). The resulting bit-string, d_{D_A}, has length at most $3n$, by Lemma 11. A Top-2$(1, 2, a, b, A)$ query can be answered as follows. We find the last node $q = [a_q, b_q]$ in level order such that $a_q \leq a$ and $b \leq b_q$ using the Top-2 encodings for each row and the bit string d_{D_A}. Since Top-2(q) is same as the Top-2$(1, 2, a, b, A)$ by the Lemma 9, we can answer Top-2$(1, 2, a, b, A)$ by finding Top-2(a_q, b_q, A_1) and Top-2(a_q, b_q, A_2), and reading the appropriate $w(q)$ bits in d_{D_A} to pick the first and the second largest elements among these four candidates. ◀

5 Conclusion

In this paper, we obtained space-efficient encodings which answer Top-k queries on 2D arrays. In particular, for an $m \times n$ 2D array, we proposed an optimal encoding when the query is one-sided. We also proposed two different encodings that answer the general (four-sided) queries. Also when $k = 2$ and $m = 2$, we obtain an encoding which uses less space than the general encoding. We end with following open problems:

- Can we support the queries efficiently on our proposed encodings of Theorem 2, Theorem 7, and Theorem 12?
- For 2 and 3-sided queries, can we obtain encodings which use less space than the encoding for the 4-sided Top-k queries on 2D array?
- Is the effective entropy of unsorted Top-k queries smaller than the effective entropy of sorted Top-k queries on 2D arrays?

[4] For nodes p and q in DAG D, we define a LCA of p and q as the set of nodes whose out-degree is zero in the subgraph of D induced by the common ancestors of p and q.

References

1. Gerth S. 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.
2. Gerth Stølting Brodal, Andrej Brodnik, and Pooya Davoodi. The encoding complexity of two dimensional range minimum data structures. In *ESA 2013, 2013. Proceedings*, pages 229–240, 2013.
3. Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro López-Ortiz. Online sorted range reporting. In *ISAAC 2009, Proceedings*, pages 173–182, 2009. doi:10.1007/978-3-642-10631-6_19.
4. Timothy M. Chan and Bryan T. Wilkinson. Adaptive and approximate orthogonal range counting. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013,2013*, pages 241–251, 2013.
5. Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Inf. Comput.*, 104(2):197–214, 1993.
6. Pawel Gawrychowski and Patrick K. Nicholson. Optimal encodings for range top- k k , selection, and min-max. In *ICALP 2015, Proceedings, Part I*, pages 593–604, 2015. doi:10.1007/978-3-662-47672-7_48.
7. Mordecai J. Golin, John Iacono, Danny Krizanc, Rajeev Raman, and S. Srinivasa Rao. Encoding 2d range maximum queries. In *ISAAC*, pages 180–189, 2011. doi:10.1007/978-3-642-25591-5_20.
8. Roberto Grossi, John Iacono, Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Encodings for range selection and top-k queries. In *ESA 2013*, pages 553–564, 2013.
9. P. B. Miltersen. Cell probe complexity – a survey. *FSTTCS*, 1999.
10. Gonzalo Navarro, Rajeev Raman, and Srinivasa Rao Satti. Asymptotically optimal encodings for range selection. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014,2014*, pages 291–301, 2014.
11. Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):Article 43, 2007.