Algorithmica
DOI 10.1007/s00453-011-9499-0

On Space Efficient Two Dimensional Range Minimum Data Structures

Gerth Stølting Brodal · **Pooya Davoodi** ·
S. Srinivasa Rao

March 18, 2011

Abstract The two dimensional range minimum query problem is to preprocess a static m by n matrix (two dimensional array) A of size $N = m \cdot n$, such that subsequent queries, asking for the position of the minimum element in a rectangular range within A, can be answered efficiently. We study the trade-off between the space and query time of the problem. We show that every algorithm enabled to access A during the query and using a data structure of size $O(N/c)$ bits requires $\Omega(c)$ query time, for any c where $1 \leq c \leq N$. This lower bound holds for arrays of any dimension. In particular, for the one dimensional version of the problem, the lower bound is tight up to a constant factor. In two dimensions, we complement the lower bound with an indexing data structure of size $O(N/c)$ bits which can be preprocessed in $O(N)$ time to support $O(c \log^2 c)$ query time. For $c = O(1)$, this is the first $O(1)$ query time algorithm using a data structure of optimal size $O(N)$ bits. For the case where queries can not probe A, we give a data structure of size $O(N \cdot \min\{m, \log n\})$ bits with $O(1)$ query time, assuming $m \leq n$. This leaves a gap to the space lower bound of $\Omega(N \log m)$ bits for this version of the problem.

Keywords Range minimum query, Cartesian tree, Time-space trade-off, Indexing model, Encoding model

1 Introduction

In this paper, we study time-space trade-offs for the two dimensional range minimum query problem (2D-RMQ). The input is an m by n matrix (two dimensional

Gerth Stølting Brodal · Pooya Davoodi
MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation), Department of Computer Science, Aarhus University, IT Parken, Åbogade 34, DK-8200 Århus N, Denmark E-mail: {gerth,pdavoodi}@cs.au.dk

S. Srinivasa Rao
School of Computer Science and Engineering Seoul National University, 599 Gwanakro, Gwanak-Gu, Seoul 151-744, S. Korea E-mail: ssrao@cse.snu.ac.kr

array) A of total of $N = m \cdot n$ elements from a totally ordered set. A query asks for the position of the minimum element in a query range $q = [i_1 \cdots i_2] \times [j_1 \cdots j_2]$, where $1 \leq i_1 \leq i_2 \leq m$ and $1 \leq j_1 \leq j_2 \leq n$, i.e., $\mathrm{RMQ}(A,q) = \mathrm{argmin}_{(i,j)\in q} A[i,j]$. W.l.o.g., we assume that $m \leq n$ and that all the entries of A are distinct (identical entries of A are ordered lexicographically by their index).

Applications. The 2D-RMQ problem has applications in computer graphics, image processing (e.g., finding the lightest/darkest point in a range; dilate/erode filters), computational Biology (e.g., finding min/max number in an alignment tableau; genome sequence analysis), and databases (e.g., range min/max query in OLAP data cubes [25]). The 1D-RMQ problem has applications in e.g., range queries [28], text indexing [1,16,26], text compression [11], document retrieval [24,27,30], flowgraphs [19], and position-restricted pattern matching [22].

Naïve structures. A naïve solution for the RMQ problem is to perform a brute force search through all the entries of the query in worst case $\Theta(N)$ time. Preprocessing A can reduce the query time. A naïve preprocessing is to store the answers to all the $O(N^2)$ possible queries in a lookup table of size $O(N^2 \log N)$ bits. The query time becomes $O(1)$ with no probe into A.

The focus of this paper is to study the time-space trade-offs between the space usage of the data structure and the query time in the two settings, where the query algorithm can access the input array A and where the query algorithm do not have access to A.

1.1 Previous Work

1.1.1 One Dimensional RMQ.

The 1D-RMQ problem is the special case of the two dimensional version where $m = 1$. It has been studied extensively. Several solutions achieve $O(1)$ query time using a data structure of size $O(n \log n)$ bits, by transforming RMQ queries into lowest common ancestor (LCA) queries [2] on the *Cartesian tree* [31] of A, see [17,21,29,9,7]. Alstrup et al. [3] solved the problem with the same bounds but without using Cartesian trees. Sadakane [27] gave the first $O(n)$-bit structure for the problem. In particular, his structure has size $4n + o(n)$ bits, achieves $O(1)$ query time, and moreover its query algorithm does not access A during the query. Later, Fischer and Heun [15] improved the problem by presenting a structure of size $2n + o(n)$ bits with $O(1)$ query time, while its query algorithm accesses the input. Their structure uses a Cartesian tree but makes no use of the LCA structure, and gives a simple solution for the static LCA problem[1]. Recently, Fischer [13] gave another structure of size $2n + o(n)$ bits, where its $O(1)$-time query algorithm does not access the input. He introduced a new data structure named 2d-Min-Heap instead of using the Cartesian tree. Table 1 summarizes these results along with the results of this paper.

[1] Fischer and Heun [15] claim a $2n - o(n)$ bits lower bound for the size of the data structure, however their proof is incorrect which, e.g., follows from Theorem 2.

Table 1 Results for the 1D-RMQ problem for an input array of n elements. The parameter c is an integer, where $1 \leq c \leq n$. The term $|A|$ denotes the size of the input A in bits. The results that $|A|$ is included in their space bound, construct a data structure of size smaller than $|A|$, although A is also stored and their query algorithm requires A. The last line is a lower bound result.

Reference	Using Cartesian tree	Using LCA	Space (bits)	Query Time
[17,21,29,9,7]	Yes	Yes	$O(n\log n)$	$O(1)$
[3,14]	No	No	$O(n\log n)$	$O(1)$
[27]	Yes	Yes	$4n+o(n)$	$O(1)$
[15]	Yes	No	$2n+o(n)+\|A\|$	$O(1)$
[13]	No	Yes	$2n+o(n)$	$O(1)$
Theorem 2	Yes	Yes	$O(n/c)+\|A\|$	$O(c)$
Theorem 1	-	-	$O(n/c)+\|A\|$	$\Omega(c)$

1.1.2 Multidimensional RMQ.

Gabow et al. [17] considered a problem where the input is a d-dimensional point set containing N points and the query is finding the point with minimum value in a rectangular range. They utilized the range trees [8] to make a data structure of size $O(N\log^d N)$ bits in $O(N\log^{d-1} N)$ preprocessing time to achieve $O(\log^{d-1} N)$ query time. Their structure can be used to solve the d-dimensional RMQ problem by mapping the N elements of the input array to N points in a d-dimensional grid. Chazelle and Rosenberg [10] gave a data structure for the Partial-sums problem in a multidimensional array of N elements. Since their structure is in the semigroup model, it also solves the d-dimensional RMQ problem in the semigroup model. Their structure has size $O(M\log N)$ bits, constructed in $O(M)$ preprocessing time, and achieves $O((\alpha(M,N))^d)$ query time for any M, where $M \geq 14^d N$, and $\alpha(M,N)$ is the functional inverse of Ackermann's function. Amir et al. [4] considered the two dimensional version of the problem. They presented a data structure of size $O(kN\log N)$ bits constructed in $O(N\log^{(k+1)} N)$ preprocessing time, which achieves $O(1)$ query time for any $k > 1$, where $\log^{(k+1)} N$ is the result of applying the log function $k+1$ times on N. Recently, Atallah and Yuan [5] gave the first linear time preprocessing algorithm for d-dimensional RMQ. Their structure has size $O(N\log N)$ bits and achieves $O(1)$ query time.

Demaine et al. [12] proved that the number of different n by n 2D-RMQ matrices is $\Omega((\frac{n}{4}!)^{n/4})$, where two 2D-RMQ matrices are considered different only if their range minima are in different locations for some rectangular range. For the 2D-RMQ problem, if the query algorithm cannot access the input matrix, the above bound implies a lower bound of $\Omega(n^2\log n)$ for both the number of preprocessing comparisons and the number of bits required for the data structure.

Table 2 summarizes the above results along with the results of this paper.

1.2 Our Results

We consider the 2D-RMQ problem in the following two models: 1) *indexing model* in which the query algorithm has access to the input matrix in addition to the data

Table 2 Results for the 2D-RMQ problem for an m by n input matrix, where $m \cdot n = N$, and $m \leq n$. The parameter c is an integer, where $1 \leq c \leq n$. The lower bound of [12] is for an n by n input matrix, where $n^2 = N$. The processing time of [10] is for any M, where $M \geq 14^2 \cdot N$. The term $|A|$ denotes the size of the input matrix A in bits. The results that include $|A|$ in their space bound, store $|A|$ and their query algorithms access A. The contributions of [17, 10, 5] and Theorem 1 can be generalized to the multidimensional version of the problem. The last three lines are lower bound results.

Reference	Query time	Space (bits)	Preprocessing time
[17]	$O(\log N)$	$O(N \log^2 N)$	$O(N \log N)$
[10]	$O((\alpha(M,N))^2)$	$O(M \log N)$	$O(M)$
[4]	$O(1)$	$O(kN \log N)$	$O(N \log^{(k+1)} N)$
[5]	$O(1)$	$O(N \log N)$	$O(N)$
Theorem 3	$O(1)$	$O(N) + \|A\|$	$O(N)$
Theorem 4	$O(c \log^2 c)$	$O(N/c) + \|A\|$	$O(N)$
Section 3.1	$O(1)$	$O(N \cdot \min\{m, \log n\})$	$O(N)$
Theorem 1	$\Omega(c)$	$O(N/c) + \|A\|$	-
[12]	-	$\Omega(n^2 \log n)$ $_{(m=n)}$	-
Theorem 5	-	$\Omega(N \log m)$	-

structure constructed by preprocessing the input. In this case, the data structure is called an *index*, and its size is referred to as the *additional space*; and 2) *encoding model* in which the query algorithm has no access to the input matrix and can only access the data structure constructed by preprocessing the input. In this case, the data structure is called an *encoding*[2].

In the indexing model, we initiate the study of the trade-off between the query time and the additional space for the 2D-RMQ problem. We prove the lower bound trade-off that $\Omega(c)$ query time is required if the additional space is $O(N/c)$ bits, for any c where $1 \leq c \leq N$. The proof is in a non-uniform cell probe model [23] which is more powerful than the indexing model. Our lower bound proof is similar to the proof of Theorem 3.1 of Golynski [20]. We complement the lower bound with an upper bound trade-off: using an index of size $O(N/c)$ bits we can achieve $O(c \log^2 c)$ query time. Note that, for the time-space product, there remains a gap of $\log^2 c$ between the upper and lower bounds. For the indexing model, this is the first $O(N)$-bit index which answers queries in $O(1)$ time.

In the encoding model, the only earlier result on the 2D-RMQ problem is the information-theoretic lower bound of Demaine et al. [12] who showed a lower bound of $\Omega(N \log n)$ bits for n by n matrices. We generalize their result to m by n (rectangular) matrices to show a lower bound of $\Omega(N \log m)$ bits, assuming $m \leq n$. We also present an encoding structure of size $O(N \cdot \min\{m, \log n\})$ bits with $O(1)$ query time. Note that the upper and lower bounds are not tight for non-constant $m = n^{o(1)}$: the lower bound states that the space requirement per element is $\Omega(\log m)$ bits, whereas the upper bound requires $O(\min\{m, \log n\})$ bits per element.

[2] Gál and Miltersen [18], and Fischer [13] suggest the names systematic and non-systematic schemes for the indexing and the encoding models respectively.

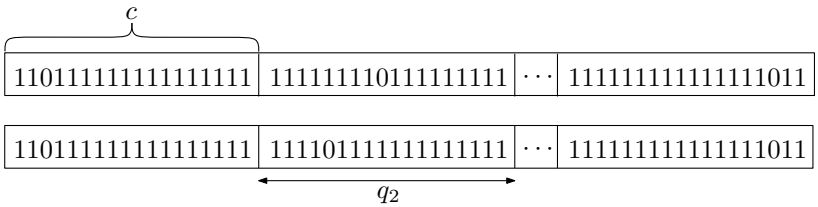

Fig. 1 Two arrays from $\mathscr{C}$, each one has n/c blocks. In this example $c = 18$. The query q_2 has different answers for these arrays.

2 Indexing Model

2.1 Lower Bound

In the indexing model, we prove a lower bound for the query time of the 1D-RMQ problem where the input is a one dimensional array of n elements, and then we show that the bound also holds for the RMQ problem in any dimension. The proof is in the non-uniform cell probe model [23]. In this model, computation is free, and time is counted as the number of cells accessed (probed) by the query algorithm. The algorithm is also allowed to be non-uniform, i.e., for different values of input parameter n, we can have different algorithms.

For integers n and c, where $1 \leq c \leq n$, we define a set of arrays $\mathscr{C}$, and a set of queries $\mathscr{Q}$. W.l.o.g., we assume that c divides n. We will argue that for any 1D-RMQ algorithm which has access to an index of size n/c bits (in addition to the input array A), there exists an array in $\mathscr{C}$ and a query in $\mathscr{Q}$ for which the algorithm performs $\Omega(c)$ probes into A.

Definition 1 Let n and c be two integers, where $1 \leq c \leq n$ and c divides n. The set $\mathscr{C}$ contains the arrays $A[1 \cdots n]$ such that the elements of A are from the set $\{0,1\}$, and in each block $A[(i-1)c+1 \cdots ic]$ for all $1 \leq i \leq n/c$, there is exactly a single zero element (see Fig. 1).

The number of possible data structures of size n/c bits is $2^{n/c}$, and the number of arrays in $\mathscr{C}$ is $c^{n/c}$. By the pigeonhole principle, for any algorithm $\mathscr{G}$ there exists a data structure $D_{\mathscr{G}}$ which is shared by at least $(\frac{c}{2})^{n/c}$ input arrays in $\mathscr{C}$. Let $\mathscr{C}_{D_{\mathscr{G}}} \subseteq \mathscr{C}$ be the set of these inputs.

Definition 2 Let $q_i = [(i-1)c+1 \cdots ic]$. The set $\mathscr{Q} = \{q_i \mid 1 \leq i \leq n/c\}$ contains n/c queries, each covering a distinct block of A.

For algorithm $\mathscr{G}$ and data structure $D_{\mathscr{G}}$, we define a binary decision tree capturing the behavior of $\mathscr{G}$ on the inputs from $\mathscr{C}_{D_{\mathscr{G}}}$ to answer a query $q \in \mathscr{Q}$.

Definition 3 Let $\mathscr{G}$ be a deterministic algorithm. For each query $q \in \mathscr{Q}$, we define a binary decision tree $T_q(D_{\mathscr{G}})$. Each internal node of $T_q(D_{\mathscr{G}})$ represents a probe into a cell of the input arrays from $\mathscr{C}_{D_{\mathscr{G}}}$. The left and right edges correspond to the output of the probe: left for reading a zero and right for reading a one. Each leaf is labelled with the answer to q, i.e., the position of the zero within the block covered by q.

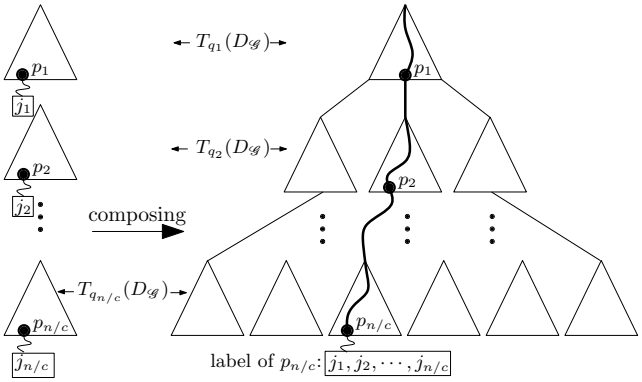

Fig. 2 Composing the n/c decision trees to obtain the large decision tree $T_{\mathscr{Q}}(D_{\mathscr{G}})$. Each leaf is labeled with a vector of positions of zeros in the input.

For each algorithm $\mathscr{G}$, we have defined n/c binary trees depicting the probes of the algorithm into the inputs from $\mathscr{C}_{D_{\mathscr{G}}}$ to answer the n/c queries in $\mathscr{Q}$. Note that the answers to all these n/c queries uniquely determine the input. We compose all the n/c binary trees into a single binary tree $T_{\mathscr{Q}}(D_{\mathscr{G}})$ in which every leaf determines a particular input. To obtain $T_{\mathscr{Q}}(D_{\mathscr{G}})$, we first replace each leaf of $T_{q_1}(D_{\mathscr{G}})$ with the whole $T_{q_2}(D_{\mathscr{G}})$, and then replace each leaf of the obtained tree with $T_{q_3}(D_{\mathscr{G}})$, and so on (see Fig. 2). Every leaf of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ is labelled with the answers to all the n/c queries in $\mathscr{Q}$ which were replaced on the path from the root to the leaf. Every two input arrays in $\mathscr{C}_{D_{\mathscr{G}}}$ correspond to different leaves of $T_{\mathscr{Q}}(D_{\mathscr{G}})$. Otherwise the answers to all the queries in $\mathscr{Q}$ are the same for both the inputs which is a contradiction. Therefore, the number of leaves of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ is at least $(\frac{c}{2})^{n/c}$, the minimum number of inputs in $\mathscr{C}_{D_{\mathscr{G}}}$.

We next prune $T_{\mathscr{Q}}(D_{\mathscr{G}})$ as follows: First we remove all nodes not reachable by any input from $\mathscr{C}_{D_{\mathscr{G}}}$. Then we repeatedly replace all nodes of degree one with their single child. Since the inputs from $\mathscr{C}_{D_{\mathscr{G}}}$ correspond to only reachable leaves, the number of leaves becomes equal to the number of inputs from $\mathscr{C}_{D_{\mathscr{G}}}$ which is at least $(\frac{c}{2})^{n/c}$. Note that the result of a *repeated* probe is known already, because the probe has been performed before. Therefore, before pruning, one child of the node corresponding to a repeated probe is unreachable, and after pruning where all the unreachable nodes are pruned, there is no repeated probe on a root to leaf path. Every path from the root to a leaf has at most n/c left edges (zero probes), since the number of zero elements in each input from $\mathscr{C}$ is n/c. Each of these paths represents a binary sequence of length at most d containing at most n/c zeros, where d is the depth of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ after pruning. By padding each of these sequences with further 0s and 1s, we can ensure that each sequence has length exactly $d+n/c$ and contains exactly n/c zeros. The number of such binary sequences is $\binom{d+n/c}{n/c}$, which becomes an upper bound for the number of leaves in the tree after prunning.

Lemma 1 *For all n and c, where* $1 \leq c \leq n$*, the worst case number of probes required to answer a query in* $\mathscr{Q}$ *over the inputs from* $\mathscr{C}$ *using a data structure of size* n/c *bits is* $\Omega(c)$.

Proof First, we prove a lower bound for d, the depth of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ after pruning. Then, we divide the lower bound by n/c, the number of binary trees, to prove the lower bound for the number of probes.

In the above discussion, we obtained the following upper bound for the number of leaves of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ after pruning.

$$\binom{d+\frac{n}{c}}{\frac{n}{c}} = \frac{(d+\frac{n}{c})!}{(\frac{n}{c})! \cdot (d+\frac{n}{c}-\frac{n}{c})!} \leq \frac{(d+\frac{n}{c})^{(\frac{n}{c})}}{(\frac{n}{c})!} .$$

Comparing this upper bound with the lower bound for the number of leaves of $T_{\mathscr{Q}}(D_{\mathscr{G}})$, we have

$$\left(\frac{c}{2}\right)^{n/c} \leq \frac{(d+\frac{n}{c})^{(\frac{n}{c})}}{(\frac{n}{c})!} .$$

By Stirling's formula, we obtain the following:

$$\frac{c}{2} \leq \frac{(d+\frac{n}{c})e}{\frac{n}{c}} ,$$

and therefore $d \geq n(\frac{1}{2e} - \frac{1}{c})$. For any arbitrary algorithm $\mathscr{G}$, the depth d of $T_{\mathscr{Q}}(D_{\mathscr{G}})$ is at most the sum of the depths of the n/c binary trees composed into $T_{\mathscr{Q}}(D_{\mathscr{G}})$. By the pigeonhole principle, there exists an input $x \in \mathscr{C}_{D_{\mathscr{G}}}$ and an i, where $1 \leq i \leq n/c$, such that the query q_i on x requires at least $d/(n/c) = \Omega(c)$ probes into the array maintaining the input. □

Theorem 1 *Any algorithm that uses* N/c *bits additional space to solve the RMQ problem for an input array of size N (in any dimension), requires* $\Omega(c)$ *query time, for any c, where* $1 \leq c \leq N$.

Proof Lemma 1 gives the lower bound for the 1D-RMQ problem. The proof for the 2D-RMQ is a simple extension of the proof of Lemma 1. The set $\mathscr{C}$ consists of matrices, each composed of mn/c submatrices $[ic_1+1 \cdots (i+1)c_1] \times [jc_2+1 \cdots (j+1)c_2]$ of size c_1 by c_2, for $1 \leq i < m/c_1$ and $1 \leq j < n/c_2$, where $c = c_1 \cdot c_2$ (w.l.o.g., assuming that c_1 divides m, and c_2 divides n). Each submatrix has exactly one zero element, and all the others are one. There are N/c queries in $\mathscr{Q}$, each one asks for the minimum of each submatrix. As in the proof of Lemma 1, we can argue that there exists a query requiring $\Omega(c)$ probes by utilizing the methods of decision trees, composing and pruning them, and bounding the number of leaves. The proof can be generalized straightforwardly to higher dimensional versions of the RMQ problem. □

The following theorem shows that the lower bound result of Theorem 1 is optimal for the 1D-RMQ problem.

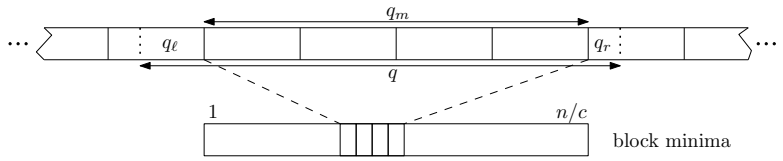

Fig. 3 The input is partitioned into n/c blocks of size c. The 1D-RMQ encoding structure $\mathscr{D}$ of size $O(n/c)$ bits is built for the list of the block minima. The query q is divided into three subqueries q_ℓ, q_m, and q_r.

Theorem 2 *The 1D-RMQ problem for a one dimensional input array of size n is solved in $O(n)$ preprocessing time and optimal $O(c)$ query time using $O(n/c)$ additional bits.*

Proof Partition the input array into n/c blocks of size c. Construct a 1D-RMQ encoding structure $\mathscr{D}$ for the list of n/c block minima (minimum elements of the blocks) in $O(n/c)$ bits [27]. The query is decomposed into three subqueries q_ℓ, q_m, and q_r (see Fig. 3). The subquery q_m contains all the blocks fully spanned by the query. To solve q_m, we first find the block containing the answer by querying the data structure $\mathscr{D}$ in $O(1)$ time, and then scan that block in $O(c)$ time to find the answer. Each of the subqueries q_ℓ and q_r, which is contained within a single block, is answered in $O(c)$ time by scanning the respective block. □

2.2 Linear Space Optimal Data Structure

2.2.1 Preliminaries

We introduce some terminology that we use to describe an indexing data structure for the 2D-RMQ problem in the following sections. A *block* is a rectangular range in a matrix. Let B be a block of size m' by n'. For the block B, the list MinColList$[1\cdots n']$ contains the minimum element of each column of B and MinRowList$[1\cdots m']$ contains the minimum element of each row of B. For integer ℓ where $1 \leq \ell \leq m'/2$, let TopSuffixes(B,ℓ) be the set of blocks $B[m'/2 - i\ell + 1\cdots m'/2] \times [1\cdots n']$, and BottomPrefixes$(B,\ell)$ be the set of blocks $B[m'/2+1\cdots m'-(i-1)\ell] \times [1\cdots n']$, for $1 \leq i \leq m'/(2\ell)$ (w.l.o.g., assuming that 2ℓ divides m').

2.2.2 Data Structure and Querying

In the following, we present an indexing data structure of size $O(N)$ bits achieving $O(1)$ query time to solve the 2D-RMQ problem for an m by n input matrix M of size $N = m \cdot n$. The basic idea of the construction is to solve the problem with four levels of recursion, reducing the queries to subqueries of size $\log\log m$ by $\log\log n$, which are solved by a tabulation idea of Atallah and Yuan [5]. We partition the input matrix M into $m/\log m$ blocks $\mathscr{B} = \{b_1,\ldots,b_{m/\log m}\}$ of size $\log m$ by n by cutting the input matrix at every $\log m$'th row. If a query is contained in a block b_i, the problem is solved recursively for this block. Otherwise, the query q is divided into

subqueries q_1, q_2 and q_3 such that q_1 is contained in b_j and q_3 is contained in b_k, and q_2 spans over $b_{j+1},\ldots,b_{k-1}$ vertically, where $1 \leq j < k \leq m/\log m$ (see Fig. 4). Since q_1 and q_3 are range minimum queries in the submatrices b_j and b_k respectively, they are answered recursively. The subquery q_2 is handled as described below. Lastly, the answers to q_1, q_2 and q_3, which are indices into three matrix elements, are used to find the index of the smallest element in q.

A binary tree structure is utilized to answer q_2. This binary tree has $m/\log m$ leaves, one for each block in $\mathscr{B}$. W.l.o.g., we assume that $m/\log m$ is a power of 2. Each leaf maintains a 1D-RMQ structure [27] for MinColList of its corresponding block b_i. Each internal node v with $2k$ leaf descendants corresponds to a submatrix M composed of $2k$ consecutive blocks of $\mathscr{B}$, for $1 \leq k \leq m/(2\log m)$. These $2k$ blocks correspond to the $2k$ leaf descendants of v. Note that each of the sets TopSuffixes$(M,\log m)$ and BottomPrefixes$(M,\log m)$ contains k blocks. For each of these $2k$ blocks, the internal node v stores a 1D-RMQ structure that is constructed for the MinColList of the block.

We also construct a 1D-RMQ structure for each of the rows and columns of the input matrix M.

In the binary tree structure, let p be the lowest common ancestor of the leaves corresponding to b_{j+1} and b_{k-1}, and let M be the submatrix corresponding to p. The subquery q_2 is composed of the top part $q_2^{\uparrow}$ and the bottom part $q_2^{\downarrow}$, where $q_2^{\uparrow}$ and $q_2^{\downarrow}$ are two blocks in the sets TopSuffixes$(M,\log m)$ and BottomPrefixes$(M,\log m)$, respectively. Two of the 1D-RMQ structures maintained in p, are constructed for MinColLists of $q_2^{\uparrow}$ and $q_2^{\downarrow}$. These 1D-RMQ structures are utilized to find two columns $c^{\uparrow}$ and $c^{\downarrow}$ containing the answer to $q_2^{\uparrow}$ and $q_2^{\downarrow}$. The 1D-RMQ structures constructed for these two columns are utilized to find the answer to $q_2^{\uparrow}$ and $q_2^{\downarrow}$. Then the answer to q_2 is determined by comparing the smallest element in $q_2^{\uparrow}$ and $q_2^{\downarrow}$.

In the second level of the recursion, each block of $\mathscr{B}$ is partitioned into blocks of size $\log m$ by $\log n$. The recursion continues for two more levels until the size of each block is $\log\log m$ by $\log\log n$. In the binary tree structures built for all the four recursion levels, we construct the 1D-RMQ structures for the appropriate MinColLists and MinRowLists respectively. The blocks that are used to make MinRowLists are defined similarly to TopSuffixes and BottomPrefixes, but for left suffixes and right prefixes respectively. In the second and fourth levels of recursion, where the binary tree structure gives two rows containing the minimum elements of $q_2^{\uparrow}$ and $q_2^{\downarrow}$, the 1D-RMQ structures constructed for the rows of the matrix are used to answer $q_2^{\uparrow}$ and $q_2^{\downarrow}$. Similar to the first level of the recursion, in the third level, where the binary tree structure gives two columns containing the minimum elements of $q_2^{\uparrow}$ and $q_2^{\downarrow}$, the 1D-RMQ structures constructed for the columns of the matrix are used to answer $q_2^{\uparrow}$ and $q_2^{\downarrow}$.

We solve the 2D-RMQ problem for a block of size $\log\log m$ by $\log\log n$ using the table lookup method given by Atallah and Yuan [5]. Their method preprocesses the block by making at most $c'G$ comparisons, for a constant c', where $G = \log\log m \cdot \log\log n$, such that any 2D-RMQ can be answered by performing four probes into the block. Each block is represented by a *block type* which is a binary sequence of length $c'G$, encoding the results of the comparisons. The lookup table has $2^{c'G}$ rows,

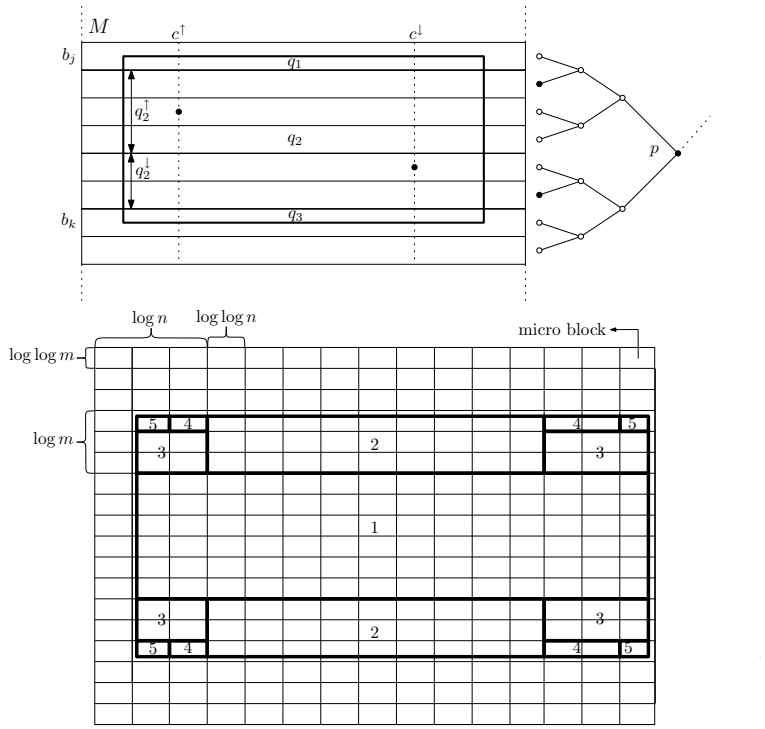

Fig. 4 Top: Partitioning the input and building the binary tree structure. The node p is the LCA of the leaves corresponding to b_{j+1} and b_{k-1}. The columns $c^{\uparrow}$ and $c^{\downarrow}$, which contain the answers to $q_2^{\uparrow}$ and $q_2^{\downarrow}$ respectively, are found using the 1D-RMQ structure stored in p. The minimum element in each of the columns $c^{\uparrow}$ and $c^{\downarrow}$ is found using the 1D-RMQ structure constructed for that column. Bottom: The numbers 1,2,3,4, and 5 on the subqueries depict the recursion level that answer the corresponding subqueries.

one for each possible block type, and G^2 columns, one for each possible query within a block. Each cell of the table contains four indices to address the four probes into the block. The block types of all the blocks of size G in the matrix are stored in another table T. The query within a block is answered by first recognizing the block type using T, and then checking the lookup table to obtain the four indices. Comparing the results of these four probes gives the answer to the query. For further details, we refer the reader to [5].

Theorem 3 *The 2D-RMQ problem for an m by n matrix of size $N = m \cdot n$ is solved in $O(N)$ preprocessing time and $O(1)$ query time using $O(N)$ bits additional space.*

Proof We first consider the query time. The subquery q_2 is answered in $O(1)$ time by using a constant query time LCA structure [6], querying the 1D-RMQ structures in constant time [27], and performing $O(1)$ probes into the input matrix. The number of recursion levels is four, and for each level, we perform at most four recursive subqueries (see Fig. 4). In the last level, the subqueries contained in blocks of size G are also answered in $O(1)$ time by using the lookup table and performing $O(1)$ probes into the matrix. Therefore the query q is answered in total $O(1)$ time.

We bound the space of the data structure as follows. The depth of the binary tree, in the first recursion level, is $O(\log(m/\log m))$. Each level of the tree has $O(m/\log m)$ 1D-RMQ structures for MinColLists of size n elements. Since a 1D-RMQ structure of a list of n elements is stored in $O(n)$ bits [27], the binary tree can be stored in $O(n \cdot m/\log m \cdot \log(m/\log m)) = O(N)$ bits. Since the number of recursion levels is $O(1)$, the binary trees in all the recursion levels are stored in $O(N)$ bits. The space used by the $m+n$ 1D-RMQ structures constructed for the columns and rows of M is $O(N)$ bits. Since $G = o(\log N)$, then $G \leq c'' \log N$ for any constant $c'' > 0$, and sufficiently large N. We can therefore bound the size of the lookup table by $O(2^{c'c'' \log N} G^2 \log G) = o(N)$ bits when $c'' < 1/c'$. The size of table T is $O(N/G \cdot \log(2^{c'G})) = O(N)$ bits. Hence the total additional space is $O(N)$ bits.

Finally, we consider the preprocessing time. In the binary tree, in the first level of the recursion, each leaf maintains a 1D-RMQ structure constructed for a MinColList of size n elements. These $m/\log m$ lists are constructed in $O(N)$ time by scanning the whole matrix. Each MinColList in the internal nodes is constructed by comparing the elements of two MinColLists built in the lower level in $O(n)$ time. Therefore constructing these lists, for the whole tree, takes $O(N + n \cdot m/\log m \cdot \log(m/\log m)) = O(N)$ time. Since a 1D-RMQ structure can be constructed in linear time [27], the 1D-RMQ structures in all the nodes of the binary tree are constructed in total $O(N)$ time. The LCA structure is also constructed in linear time [6]. Therefore the binary tree is built in $O(N)$ time. Since the number of recursion levels is $O(1)$, all the binary trees are built in $O(N)$ time. The lookup table and table T are also constructed in $O(N)$ time, see Sections 3.2 and 5 in [5]. □

Corollary 1 *The query algorithm performs at most 38 probes into the input to solve the query.*

Proof As shown at the top of Fig. 4, the subquery q_2 is answered by comparing the smallest elements in $q_2^{\uparrow}$ and $q_2^{\downarrow}$. To find these two smallest elements, the algorithm performs two probes into the input. For each of the subqueries solved in different recursion levels, shown at the bottom of Fig. 4, at most two probes are performed. As described earlier, to solve the subqueries contained in blocks of size $\log\log m$ by $\log\log n$, four probes are performed. Therefore, the total number of probes in the recursion levels is the sum: $2 + 2 \cdot 2 + 4 \cdot 2 + 4 \cdot 2 + 4 \cdot 4 = 38$. □

2.3 Space Time Trade-off Data Structure

We now describe how to use the data structure of Section 2.2 to achieve a trade-off between the additional space usage and the query time. We present an indexing data structure of size $O(N/c \cdot \log c)$ bits additional space solving the 2D-RMQ problem in $O(c \log c)$ query time and $O(N)$ preprocessing time, where $1 \leq c \leq N$. The input matrix is divided into N/c blocks of size 2^i by $c/2^i$, for each integer i in the range $[0 \cdots \log c]$; w.l.o.g., assuming that c is a power of 2. Let M_i be the matrix of size N/c containing the minimum elements of the blocks of size 2^i by $c/2^i$. Let D_i be the linear space data structure of Section 2.2 applied to the matrix M_i using $O(N/c)$ bits. Each D_i handles a different ratio between the number of rows and the number of

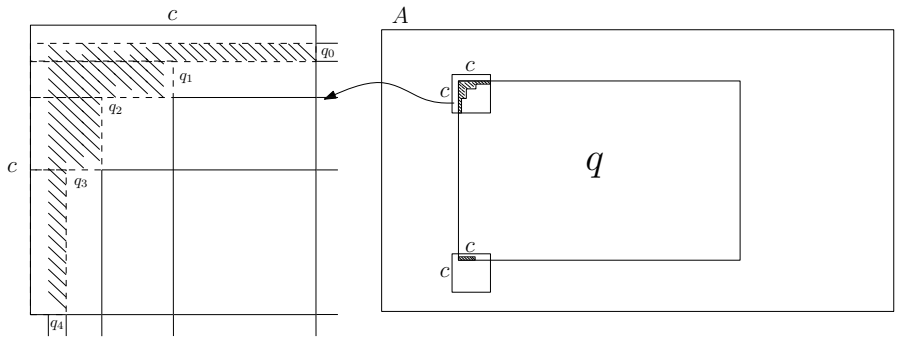

Fig. 5 Right: The white area of the query q contains the subqueries which completely span the blocks of size 2^i by $c/2^i$. Left: A corner of q which is contained in a block of size c by c. The shaded area contains $O(c \log c)$ elements.

columns of the blocks. Note that the matrices M_i are constructed temporarily during the preprocessing and are not maintained in the data structure.

A query q is resolved by answering $\log c + 1$ subqueries. Let q_i be the maximal subquery of q spanning blocks of size 2^i by $c/2^i$ for $0 \leq i \leq \log c$. The minimum elements of the blocks spanned by q_i assemble a query over M_i which has the same answer as q_i. Therefore, q_i is answered by using D_i. Note that whenever the algorithm wants to perform a probe into a cell of M_i, a corresponding block of size c of the input is searched for the minimum (since M_i is not explicitly stored in the data structure). The subqueries q_i overlap each other. Altogether, they compose q except for $O(c \log c)$ elements in each of the four corners of q (see the proof of Theorem 4). We search these corners for the minimum element. Eventually, we compare the minimum elements of all the subqueries to find the answer to q (see Fig. 5).

Theorem 4 *The 2D-RMQ problem for a matrix of size N is solved in $O(N)$ preprocessing time and $O(c \log^2 c)$ query time using $O(N/c)$ bits additional space.*

Proof The number of linear space data structures D_i is $\log c + 1$. Each data structure D_i requires $O(N/c)$ bits. Therefore, the total additional space is $O(\log c \cdot N/c)$ bits.

The number of subqueries q_i is $\log c + 1$. Each q_i is answered by using D_i in $O(1)$ query time in addition to the $O(1)$ probes into M_i. Since each probe into M_i can be performed by $O(c)$ probes into the input matrix, the query q_i can be answered in $O(c)$ time. Each of the four corners of the query q not covered by the q_i queries, is contained in the union of at most $\log c + 1$ blocks, at most one block of each size 2^i by $c/2^i$ for $0 \leq i \leq \log c$ (see Fig. 5). The four corners are searched in $O(c \log c)$ time for the minimum element. In the end, the minimum elements of the subqueries are compared in $O(\log c)$ time to answer q. Consequently, the total query time is $O(c \log c)$.

Each D_i is constructed in $O(N/c)$ time (Section 2.2) after building the matrix M_i. To be able to make all M_i efficiently, we first construct an $O(N)$-bit space data structure of Section 2.2 for the input matrix in $O(N)$ time. Then, M_i is built in $O(N/c)$ time by querying a block of the input matrix in $O(1)$ time for each element of M_i.

Therefore, the total preprocessing time is $O(\log c \cdot N/c + N) = O(N)$. Substituting the parameter c by $c \log c$ gives the claimed bounds. □

3 Encoding Model

3.1 Upper Bound

The algorithm described in Section 2.2 can preprocess an m by n input matrix A of size $N = m \cdot n$ into a data structure of size $O(N)$ bits in $O(N)$ time. But the query algorithm in Section 2.2 is required to perform some probes into the input matrix. Since A is not accessible in the encoding model, we store another matrix maintaining the rank of all the N elements using $O(N \log N) = O(N \log n)$ bits. Whenever the algorithm wants to perform a probe into A, it does it into the rank matrix. Therefore the problem can be solved in the encoding model using $O(N \log n)$ preprocessing time (to sort A) and $O(1)$ query time using space $O(N \log n)$ bits.

Another solution in the encoding model is the following. For each of the n columns of A, we build a 1D-RMQ structure using space $O(m)$ bits [27], in total using $O(mn) = O(N)$ bits. Furthermore, for each possible pair of rows (i_1, i_2), $i_1 \leq i_2$, we construct a 1D-RMQ structure for the MinColList L_{i_1,i_2} of $A[i_1 \cdots i_2] \times [1 \cdots n]$, i.e., $L_{i_1,i_2}[j] = \min_{i_1 \leq i \leq i_2} A[i, j]$, using space $O(n)$ bits. Note that we only store the 1D-RMQ structure for L_{i_1,i_2}, but not L_{i_1,i_2} itself. In total we use space $O(m^2 n) = O(Nm)$ bits. The column j containing the answer to a query $q = [i_1 \cdots i_2] \times [j_1 \cdots j_2]$ is found by querying for the range $[j_1 \cdots j_2]$ in the 1D-RMQ structure for L_{i_1,i_2}. The query q is answered by querying for the range $[i_1 \cdots i_2]$ in the 1D-RMQ structure for column j. Since both 1D-RMQ queries take $O(1)$ time, the total query time is $O(1)$.

Selecting the most space efficient solution of the above two solutions gives an encoding structure of size $O(N \cdot \min\{m, \log n\})$ bits with $O(1)$ query time.

3.2 Lower Bound

To prove a lower bound for the space required in the encoding model, we generate a large class of input matrices which are distinguishable by the queries. We consider two matrices A_1 and A_2 *different* if there exists a 2D-RMQ with different answer for A_1 and A_2. We present a set of $\Omega((m!)^n)$ matrices which are pairwise different. The elements of the matrices are from the set $\{1, \ldots, mn\}$. In every matrix A of the set, the smallest mn' elements of A are placed in two parts $A' = A[1 \cdots m/2] \times [1 \cdots n']$ and A'' containing all the anti-diagonals of length $m/2$ within the block $A[m/2 + 1 \cdots m] \times [n' + 1 \cdots n]$ where $n' = \lfloor (n - m/2 + 1)/2 \rfloor$, w.l.o.g., assuming that m is even (see Fig. 6). The odd numbers from the set $\{1, \ldots, mn'\}$ are placed in A' in increasing order from left to right and then top to bottom, i.e. $A'[i, j] = 2((i-1)n' + j) - 1$. The even numbers of $\{1, \ldots, mn'\}$ are placed in A'' such that the elements of each anti-diagonal are not sorted but are larger than the elements of the anti-diagonals to the right. The total number of matrices constructed by permuting the $m/2$ elements of each of the n' anti-diagonals of A'' is $(\frac{m}{2}!)^{n'}$.

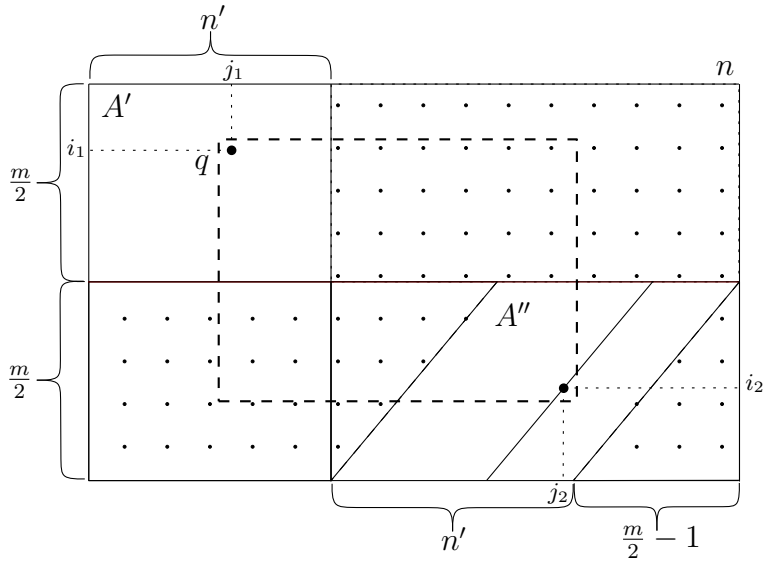

Fig. 6 All the elements in the dotted area are greater than the elements in the white area. The rectangle drawn with dashed line shows the query q. The smallest entry in $A' \cap q$ is $A[i_1, j_1]$ and the smallest entry in $A'' \cap q$ is $A[i_2, j_2]$.

For any two matrices A_1 and A_2 in the set, there exists an index $[i_2, j_2]$ in the anti-diagonals of A'' such that $A_1[i_2, j_2] \neq A_2[i_2, j_2]$. W.l.o.g., assume that $A_1[i_2, j_2] < A_2[i_2, j_2]$. Let $[i_1, j_1]$ be the index of an arbitrary odd number in A' between $A_1[i_2, j_2]$ and $A_2[i_2, j_2]$. The query $q = [i_1 \cdots i_2] \times [j_1 \cdots j_2]$ has different answers for A_1 and A_2: For A_1 the answer is $[i_2, j_2]$ whereas the answer for A_2 is $[i_1, j_1]$ (see Fig. 6). It follows that any two matrices in the set are different.

Theorem 5 *The minimum space required to store an encoding data structure for the 2D-RMQ problem is $\Omega(mn \log m)$ bits, assuming that $m \leq n$.*

Proof Since the number of different matrices in the set is $(\frac{m}{2}!)^{n'}$, the space for a data structure encoding these matrices is $\Omega(\log(\frac{m}{2}!)^{n'}) = \Omega(mn' \log m) = \Omega(mn \log m)$ bits, since $n' \geq (n - \frac{m}{2})/2 \geq n/4$, where the last inequality follows from $m \leq n$. □

4 Conclusion and open problems

We studied the range minimum query (RMQ) problem in the indexing and encoding models and showed various time-space trade-offs for the problem in one and two dimensions. For the 1D-RMQ problem we obtained an optimal trade-off result in the indexing model by showing that any algorithm that has access to an index of size $O(n/c)$ bits has query complexity $\Theta(c)$, where n is the size of the input array, c is any parameter, and $c \leq n$. The lower bound also holds for the higher dimensional versions of the RMQ problem.

The upper bound for the 2D-RMQ problem in the indexing model supports queries in $O(c\log^2 c)$ time with access to an index of size $O(N/c)$ bits, where N is the size of the input matrix. This leaves a gap of $O(\log^2 c)$ factor between the upper and lower bounds, and closing this gap is an interesting open problem.

For the 2D-RMQ problem in the encoding model we obtained an upper bound of $O(mn \cdot \min\{m, \log n\})$, and a lower bound of $O(mn \log m)$. It would be interesting to settle the space complexity for this problem by closing this gap.

Acknowledgements

We wish to thank Rajeev Raman for fruitful discussions on this problem. We would also like to thank the anonymous referees for their helpful comments.

References

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Algorithms of Discrete Algorithms*, 2(1):53–86, 2004.
2. A. Aho, J. Hopcroft, and J. Ullman. On finding lowest common ancestors in trees. In *Proc. 5th Annual ACM Symposium on Theory of Computing*, pages 253–265. ACM, 1973.
3. S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *Proc. 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 258–264. ACM, 2002.
4. A. Amir, J. Fischer, and M. Lewenstein. Two-dimensional range minimum queries. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching*, volume 4580 of *LNCS*, pages 286–294. Springer-Verlag, 2007.
5. M. J. Atallah and H. Yuan. Data structures for range minimum queries in multidimensional arrays. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 150–160. SIAM, 2010.
6. M. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. 4th Latin American Theoretical Informatics Symposium*, volume 1776 of *LNCS*, pages 88–94. Springer-Verlag, 2000.
7. M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.
8. J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.
9. O. Berkman, Z. Galil, B. Schieber, and U. Vishkin. Highly parallelizable problems. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 309–319. ACM, 1989.
10. B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *Proc. 5th Annual Symposium on Computational Geometry*, pages 131–139. ACM, 1989.
11. G. Chen, S. J. Puglisi, and W. F. Smyth. Lempel-Ziv factorization using less time and space. *Mathematics in Computer Science*, 1:605–623, 2008.
12. E. D. Demaine, G. M. Landau, and O. Weimann. On cartesian trees and range minimum queries. In *Proc. 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *LNCS*, pages 341–353. Springer-Verlag, 2009.
13. J. Fischer. Optimal succinctness for range minimum queries. In *Proc. 9th Latin American Theoretical Informatics Symposium*, volume 6034 of *LNCS*, pages 158–169. Springer-Verlag, 2010.
14. J. Fischer and V. Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Proc. 17th Annual Symposium on Combinatorial Pattern Matching*, volume 4009 of *LNCS*, pages 36–48. Springer-Verlag, 2006.
15. J. Fischer and V. Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In *Proc. 1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, volume 4614 of *LNCS*, pages 459–470. Springer-Verlag, 2007.

16. J. Fischer, V. Mäkinen, and G. Navarro. An(other) entropy-bounded compressed suffix tree. In *Proc. 19th Annual Symposium on Combinatorial Pattern Matching*, volume 5029 of *LNCS*, pages 152–165. Springer-Verlag, 2008.
17. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symposium on Theory of Computing*, pages 135–143. ACM, 1984.
18. A. Gál and P. B. Miltersen. The cell probe complexity of succinct data structures. *Theoretical Computer Science*, 379(3):405–417, 2007.
19. L. Georgiadis and R. E. Tarjan. Finding dominators revisited: extended abstract. In *Proc. 15th Annual ACM-SIAM symposium on Discrete algorithms*, pages 869–878. SIAM, 2004.
20. A. Golynski. Optimal lower bounds for rank and select indexes. *Theoretical Computer Science*, 387(3):348–359, 2007.
21. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
22. C. S. Iliopoulos, M. Crochemore, M. Kubica, M. S. Rahman, and T. Walen. Improved algorithms for the range next value problem and applications. In *Proc. 25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics*, pages 205–216. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.
23. P. B. Miltersen. Cell probe complexity - a survey. Advances in Data Structures Workshop (Preworkshop of FSTTCS), 1999. `http://www.daimi.au.dk/~bromille/Papers/survey3.ps`.
24. S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM symposium on Discrete algorithms*, pages 657–666. SIAM, 2002.
25. C. K. Poon. Dynamic orthogonal range queries in OLAP. *Theory of Computing Systems*, 296(3):487–510, 2003.
26. K. Sadakane. Compressed suffix trees with full functionality. *Theory of Computing Systems*, 41(4):589–607, 2007.
27. K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.
28. S. Saxena. Dominance made simple. *Information Processing Letters*, 109(9):419–421, 2009.
29. B. Schieber and U. Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.
30. N. Välimäki and V. Mäkinen. Space-efficient algorithms for document retrieval. In *Proc. 18th Annual Symposium on Combinatorial Pattern Matching*, volume 4580 of *LNCS*, pages 205–215. Springer-Verlag, 2007.
31. J. Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980.