Submatrix Maximum Queries in Monge Matrices are Equivalent to Predecessor Search[⋆]

Paweł Gawrychowski[1⋆⋆], Shay Mozes[2⋆⋆⋆], and Oren Weimann[3⋆⋆⋆]

[1] University of Warsaw, gawry@mimuw.edu.pl
[2] IDC Herzliya, smozes@idc.ac.il
[3] University of Haifa, oren@cs.haifa.ac.il

Abstract. We present an optimal data structure for submatrix maximum queries in $n \times n$ Monge matrices. Our result is a two-way reduction showing that the problem is equivalent to the classical predecessor problem in a universe of polynomial size. This gives a data structure of $O(n)$ space that answers submatrix maximum queries in $O(\log\log n)$ time, as well as a matching lower bound, showing that $O(\log\log n)$ query-time is optimal for any data structure of size $O(n\,\mathrm{polylog}(n))$. Our result settles the problem, improving on the $O(\log^2 n)$ query-time in SODA'12, and on the $O(\log n)$ query-time in ICALP'14.
In addition, we show that partial Monge matrices can be handled in the same bounds as full Monge matrices. In both previous results, partial Monge matrices incurred additional inverse-Ackerman factors.

1 Introduction

Data structures for range queries and for predecessor queries are among the most studied data structures in computer science. Given an $n \times n$ matrix M, a *range maximum* (also called submatrix maximum) data structure can report the maximum entry in any query submatrix (a set of consecutive rows and a set of consecutive columns) of M. Given a set $S \subseteq [0, U)$ of n integers from a polynomial universe U, a *predecessor* data structure can report the predecessor (and successor) in S of any query integer $x \in [0, U)$. In this paper, we prove that these two seemingly unrelated problems are in fact equivalent when the matrix M is a *Monge* matrix.

Range maximum queries. A long line of research over the last three decades including [3,9,10,13,20] achieved range maximum data structures of $\tilde{O}(n^2)$ space and $\tilde{O}(1)$ query time[4], culminating with the $O(n^2)$-space $O(1)$-query data structure of Yuan and Atallah [20]. In general matrices, this is optimal since

⋆ A full version of this paper can be found as Arxiv preprint arXiv:1502.07663.

⋆⋆ PG is currently holding a post-doctoral position at Warsaw Center of Mathematics and Computer Science.

⋆⋆⋆ SM and OW partially supported by Israel Science Foundation grant 794/13. SM partially supported by the Israeli ministry of absorption.

4 The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in n.

representing the input matrix already requires $\Theta(n^2)$ space. In fact, reducing the additional space to $O(n^2/c)$ is known to incur an $\Omega(c)$ query-time [5] and such tradeoffs can indeed be achieved for any value of c [4,5].

However, in many applications, the matrix M is not stored explicitly but any entry of M can be computed when needed in $O(1)$ time. One such case is when M is a sparse matrix with $N = o(n^2)$ nonzero entries. In this case the problem is known in computational geometry as the *orthogonal range searching* problem on the $n \times n$ grid. Various data structures with $\tilde{O}(N)$-space and $\tilde{O}(1)$-query appear in a long history of results including [2,7,8,11,13]. For a survey on orthogonal range searching see [18]. Another case where the additional space can be made $o(n^2)$ (and in fact even $O(n)$) is when the matrix is a Monge matrix.

Range maximum queries in Monge matrices. A matrix M is Monge if for any pair of rows $i < j$ and columns $k < \ell$ we have that $M[i,k] + M[j,\ell] \geq M[i,\ell] + M[j,k]$. Submatrix maximum queries on Monge matrices have various important applications in combinatorial optimization and computational geometry such as problems involving distances in the plane, and in problems on convex n-gons. See [6] for a survey on Monge matrices and their uses in combinatorial optimization. Submatrix maximum queries on Monge matrices are used in algorithms that efficiently find the largest empty rectangle containing a query point, in dynamic distance oracles for planar graphs, and in algorithms for maximum flow in planar graphs. See [15] for more details.

Given an $n \times n$ Monge matrix M it is possible to obtain compact data structures of only $\tilde{O}(n)$ space that can answer submatrix maximum queries in $\tilde{O}(1)$ time. The first such data structure was given by Kaplan, Mozes, Nussbaum and Sharir [15]. They presented an $O(n \log n)$-space data structure with $O(\log^2 n)$ query time. This was improved in [14] to $O(n)$ space and $O(\log n)$ query time.

Breakpoints and Partial Monge matrices. Given an $m \times n$ Monge matrix M, let $r(c)$ be the row containing the maximum element in the c-th column of M. It is easy to verify that the $r(\cdot)$ values are monotone, i.e., $r(1) \leq r(2) \leq \ldots \leq r(n)$. Columns c such that $r(c-1) < r(c)$ are called the *breakpoints* of M. A Monge matrix consisting of $m < n$ rows has $O(m)$ breakpoints, which can be found in $O(n)$ time using the SMAWK algorithm [1].

Some applications involve *partial* Monge matrices rather than full Monge matrices. A partial Monge matrix is a Monge matrix where some of the entries are undefined, but the defined entries in each row and in each column are contiguous. The total number of breakpoints in a partial Monge matrix is still $O(m)$ [14], and they can be found in $O(n \cdot \alpha(n))$ time[5] using an algorithm of Klawe and Kleitman [16]. This was used in [14,15] to extend their solutions to partial Monge matrices at the cost of an additional $\alpha(n)$ factor to the query time.[6]

Our results. In this paper, we fully resolve the submatrix maximum query problem in $n \times n$ Monge matrices by presenting a data structure of $O(n)$ space and $O(\log \log n)$ query time. Consequently, we obtain an improved query time

[5] Here $\alpha(n)$ is the inverse-Ackerman function.
[6] In [15], there was also an additional $\log n$ factor to the space.

for other applications such as finding the largest empty rectangle containing a query point. We compliment our upper bound with a matching lower bound, showing that $O(\log\log n)$ query-time is optimal for any data structure of size $O(n \operatorname{polylog}(n))$. In fact, implicit in our upper and lower bound is an equivalence between the predecessor problem in a universe of polynomial size and the range maximum query problem in Monge matrices. The upper bound essentially reduces a submatrix query to a predecessor problem, and vice versa, the lower bound reduces the predecessor problem to a submatrix query problem.

Finally, we extend our result to partial Monge matrices with the exact same bounds (i.e., $O(n)$ space and $O(\log\log n)$ query time). Our result is the first to achieve such extension with no overhead.

Techniques. Let M be an $n \times n$ Monge matrix[7]. Consider a full binary tree $\mathcal{T}$ whose leaves are the rows of M. Let M_u be the submatrix of M composed of all rows (i.e., leaves) in the subtree of a node u in $\mathcal{T}$. Both existing data structures for submatrix maximum queries [14,15] store, for each node u in $\mathcal{T}$ a data structure D_u. The goal of D_u is to answer submatrix maximum queries for queries that include an arbitrary interval of columns and *exactly all rows* of M_u. This way, an arbitrary query is covered in [14,15] by querying the D_u structures of $O(\log n)$ canonical nodes of $\mathcal{T}$. An $\Omega(\log n)$ bound is thus inherent for any solution that examines the canonical nodes. We overcome this obstacle by designing a stronger data structure D_u. Namely, one that supports queries that include an arbitrary interval of columns and *a prefix of rows* or *a suffix of rows* of M_u. This way, an arbitrary query can be covered by just two D_us. The idea behind the new design is to efficiently encode the changes in column maxima as we add rows to M_u one by one. Retrieving this information is done using weighted ancestor search and range maximum queries on trees. This is a novel use of these techniques.

For our lower bound, we show that for any set of n integers $S \subseteq [0, n^2)$ there exists an $n \times n$ Monge matrix M such that the predecessor of x in S can be found with submatrix minimum queries on M. The predecessor lower bound of Pătraşcu and Thorup [19] then implies that $O(n \operatorname{polylog}(n))$ space requires $\Omega(\log\log n)$ query time. We overcome two technical difficulties here: First, M should be Monge. Second, there must be an $O(n \operatorname{polylog}(n))$-size representation of M which can retrieve any entry $M[i,j]$ in $O(1)$ time.

Finally, for handling partial Monge matrices, and unlike previous solutions for this case, we do not directly adapt the solution for the full Monge case to partial Monge matrices. Instead we decompose the partial Monge matrix into many full Monge matrices, that can be preprocessed to be queried cumulatively in an efficient way. This requires significant technical work and careful use of the structure of the decomposition.

Roadmap. In Section 2 we present an $O(n \log n)$-space data structure for Monge matrices that answers submatrix maximum queries in $O(\log\log n)$ time. In Section 3 we reduce the space to $O(n)$. Our lower bound is given in Section 4.

[7] We consider $m \times n$ matrices, but for simplicity we sometimes state the results for $n \times n$ matrices.

The extension to partial Monge matrices, that we believe is a significant contribution of our paper, is deferred to the full version due to lack of space.

2 Data structure for Monge matrices

Our goal in this section is to construct, for a given $m \times n$ Monge matrix M, a data structure of size $O(m \log n)$ that answers submatrix maximum queries in $O(\log \log n)$ time. In Section 3 we show how to reduce the space from $O(n \log n)$ to $O(n)$ when $m = n$. We will actually show a stronger result, namely the structure allows us to reduce in $O(1)$ time a submatrix maximum query into $O(1)$ predecessor queries on a set consisting of n integers from a polynomial universe.

We denote by $pred(m,n)$ the complexity of a predecessor query on a set of m integers from a universe $\{0,\ldots,n-1\}$. It is well known that there are $O(m)$ data structures achieving $pred(m,n) = \min\{O(\log m), O(\log \log n)\}$.

Recall that a submatrix maximum query returns the maximum $M[i,j]$ over all $i \in [i_0,i_1]$ and $j \in [j_0,j_1]$ for a given $i_0 \le i_1$ and $j_0 \le j_1$. We start by answering the easier *subcolumn maximum queries* within these space and time bounds. That is, finding the maximum $M[i,j]$ over all $i \in [i_0,i_1]$ for a given $i_0 \le i_1$ and j.

We construct a full binary tree $\mathcal{T}$ over the rows of M. Every leaf of the tree corresponds to a single row of M, and every inner node corresponds to the range of rows in its subtree. To find the maximum $M[i,j]$ over all $i \in [i_0,i_1]$ for a given $i_0 \le i_1$ and j, we first locate the lowest common ancestor (lca) u of the leaves corresponding to i_0 and i_1 in the tree. Then we decompose the query into two parts: one fully within the range of rows M_ℓ of the left child of u, and one fully within the range of rows M_r of the right child of u. The former ends at the last row of M_ℓ and the latter starts at the first row of M_r. We equip every node with two data structures allowing us to answer such simpler subcolumn maximum queries. Because of symmetry (if M is Monge, so is M', where $M'[i,j] = M[n+1-i, n+1-j]$) it is enough to show how to answer subcolumn maximum queries starting at the first row.

Lemma 1. *Given an $m \times n$ Monge matrix M, a data structure of size $O(m)$ can be constructed in $O(m \log n)$ time to answer in $O(pred(m,n))$ time subcolumn maximum queries starting at the first row of M.*

Proof. Consider queries spanning an *entire* column c of M. To answer such a query, we only need to find the corresponding $r(c)$. If we store the breakpoints of M in a predecessor structure, where every breakpoint c links to its corresponding value of $r(c)$, a query can be answered with a single predecessor search. More precisely, to determine the maximum in the c-th column of M, we locate the largest breakpoint $c' \le c$, and set $r(c) = r(c')$. Hence we can construct a data structure of size $O(m)$ to answer *entire column* maximum queries in $O(pred(m,n))$ time.

Let M_i be a Monge matrix consisting of the first i rows of M. By applying the above reasoning to every M_i separately, we immediately get a structure of size $O(m^2)$ answering subcolumn maximum queries starting at the first row of M in $O(pred(m,n))$ time. We want to improve on this by utilizing the dependency of the structures constructed for different i's. Namely it can be observed that the list of breakpoints of M_{i+1} is a prefix of the list of breakpoints of M_i to which we append at most one new element. In other words, if the breakpoints of M_i are stored on a stack, we need to pop zero or more elements and push at most one new element to represent the breakpoints of M_{i+1}. Consequently, instead of storing a separate list for every M_i, we can succinctly describe the content of all stacks with a single tree T on at most $m+1$ nodes. For every i, we store a pointer to a node $s(i) \in T$, such that the ancestors of $s(i)$ (except for the root) are exactly the breakpoints of M_i. Whenever we pop an element from the current stack, we move to the parent of the current node, and whenever we push an element, we create a new node and make it a child of the current node. Initially, the tree consists of just the root. Every node is labelled with a column number and by construction these numbers are strictly increasing on any path starting at the root (the root is labelled with $-\infty$). Therefore, a predecessor search for j among the breakpoints of M_i reduces to finding the leafmost ancestor of $s(i)$ whose label is at most j. This is known as the *weighted ancestor* problem. Weighted ancestor queries on a tree of size $O(m)$ are equivalent to predecessor searching on a number of sets of $O(m)$ total size [17][8], achieving the claimed space and query time bounds.

To finish the proof, we need to bound the construction time. The bottleneck is constructing the tree T. Let $c_1 < c_2 < \ldots < c_k$ for some $k \le i$ be the breakpoints of M_i. As long as $M[i+1, c_k] \ge M[r(c_k), c_k]$ we decrease k by one, i.e., remove the last breakpoint. This process is repeated $O(m)$ times in total. If $k = 0$ we create a new breakpoint $c_1 = 1$. If $k \ge 1$ and $M[i+1, c_k] < M[r(c_k), c_k]$, we check if $M[i+1, n] \ge M[r(c_k), n]$. If so, we need to create a new breakpoint. To this end, we need to find the smallest j such that $M[i+1, j] \ge M[r(c_k), j]$. This can be done in $O(\log n)$ using binary search. Consequently, T can be constructed in $O(m \log n)$ time. Then augmenting it with a weighted ancestor structure takes $O(m)$ time. $\square$

We apply Lemma 1 twice to every node of the full version tree $\mathcal{T}$. Once for subcolumn maximum queries starting at the first row and once for queries ending at the last row. Since the total size of all structures at the same level of the tree is $O(m)$, the total size of our subcolumn maximum data structure becomes $O(m \log m)$, and it can be constructed in $O(m \log m \log n)$ time to answer queries in $O(pred(m, n))$ time. Hence we have proved the following.

Theorem 1. *Given an $m \times n$ Monge matrix M, a data structure of size $O(m \log m)$ can be constructed in $O(m \log m \log n)$ time to answer subcolumn maximum queries in $O(pred(m, n))$ time.*

By symmetry (a transpose of a Monge matrix is Monge) we can answer subrow maximum queries (where the query is a single row and a range of columns) in $O(pred(n, m))$ time. We are now ready to tackle general submatrix maximum queries.

[8] Technically, the reduction adds $O(\log^* m)$ to the query time, but this can be avoided.

At a high level, the idea is identical to the one used for subcolumn maximum queries: we construct a full binary tree $\mathcal{T}$ over the rows of M, where every node corresponds to a range of rows. To find maximum $M[i,j]$ over all $i \in [i_0, i_1]$ and $j \in [j_0, j_1]$ for a given $i_0 \leq i_1$ and $j_0 \leq j_1$, we locate the lowest common ancestor of the leaves corresponding to i_0 and i_1 and decompose the query into two parts, the former ending at the last row of M_ℓ and the latter starting at the first row of M_r. Every node is equipped with two data structures allowing us to answer submatrix maximum queries starting at the first row or ending at the last row. As before, it is enough to show how to answer submatrix maximum queries starting at the first row.

Lemma 2. *Given an $m \times n$ Monge matrix M, and a data structure that answers subrow maximum queries on M in $O(pred(n,m))$ time, one can construct in $O(m \log m)$ time a data structure consuming $O(m)$ additional space, that answers submatrix maximum queries starting at the first row of M in $O(pred(m,n) + pred(n,m))$ time.*

Proof. We extend the proof of Lemma 1. Let $c_1 < c_2 < \ldots < c_k$ be the breakpoints of M stored in a predecessor structure. For every $i \geq 2$ we precompute and store the value $m_i = \max_{j \in [c_{i-1}, c_i)} M[r(c_{i-1}), j]$. These values are augmented with a (one dimensional) range maximum query data structure. To begin with, consider a submatrix maximum query starting at the first row of M and ending at the last row of M, i.e., we need to calculate the maximum $M[i,j]$ over all $i \in [1, m]$ and $j \in [j_0, j_1]$. We find in $O(pred(m,n))$ the successor of j_0, denoted c_i, and the predecessor of j_1, denoted $c_{i'}$. There are three possibilities:

1. The maximum is reached for $j \in [j_0, c_i)$,
2. The maximum is reached for $j \in [c_i, c_{i'})$,
3. The maximum is reached for $j \in [c_{i'}, j_1)$.

The first and the third possibilities can be calculated with subrow maximum queries in $O(pred(n,m))$, because both ranges span an interval of columns and a single row. The second possibility can be calculated with a range maximum query on the range $(i, i']$. Consequently, we can construct a data structure of size $O(m)$ to answer such submatrix maximum queries in $O(pred(m,n) + pred(n,m))$ time.

The above solution can be generalized to queries that start at the first row of M but do not necessarily end at the last row of M. This is done by considering the Monge matrices M_i consisting of the first i rows of M. For every such matrix, we need a predecessor structure storing all of its breakpoints, and additionally a range maximum structure over their associated values. Hence now we need to construct a similar tree T as in Lemma 1 on $O(m)$ nodes, but now every node has both a weight and a value. The weight of a node is the column number of the corresponding breakpoint c_k, and the value is its m_k (or undefined if $k = 1$). As in Lemma 1, the breakpoints of M_i are exactly the ancestors of the node $s(i)$. Note that every m_k is defined in terms of c_{k-1} and c_k, but this is not a problem because the predecessor of a breakpoint does not change during the whole construction. We maintain a weighted ancestor structure using the weights (in order to find c_i and $c_{i'}$ in $O(pred(m,n))$ time), and a *generalized range maximum structure* using the values. A generalized range maximum structure of a tree T, given two query nodes u and v, returns the maximum value on the unique u-to-v path in T. It can be implemented in $O(m)$ space and $O(1)$ query time after $O(m \log m)$ preprocessing [10] once we have the values. The values can be computed with subrow maximum queries in $O(m \cdot pred(n, m)) = O(m \log m)$ total time. $\square$

By applying Lemma 2 twice to every node of the full binary tree $\mathcal{T}$, we construct in $O(m \log^2 m)$ time a data structure of size $O(m \log m)$ to answer submatrix maximum queries in $O(pred(m, n) + pred(n, m))$ time. In order to apply Lemma 2 to a node of $\mathcal{T}$ we need a subrow maximum query data structure for the corresponding rows of the matrix M. Note, however, that a single subrow maximum query data structure for M can be used for all nodes of $\mathcal{T}$.

Theorem 2. *Given an $m \times n$ Monge matrix M, and a data structure answering subrow maximum queries on M in $O(pred(n, m))$ time, one can construct in $O(m \log^2 m)$ time a data structure taking $O(m \log m)$ additional space, that answers submatrix maximum queries on M in $O(pred(m, n) + pred(n, m))$ time.*

By combining Theorem 1 with Theorem 2, given an $n \times n$ Monge matrix M, a data structure of size $O(n \log n)$ can be constructed in $O(n \log^2 n)$ time to answer submatrix maximum queries in $O(pred(n, n))$ time.

3 Obtaining linear space

In this section we show how to decrease the space of the data structure presented in Section 2 to be linear. We extend the idea developed in our previous paper [14]. The previous linear space solution was based on partitioning the matrix M into n/x matrices $M_1, M_2, \ldots, M_{n/x}$, where each M_i is a *slice* of M consisting of $x = \log n$ consecutive rows. Then, instead of working with the matrix M, we worked with the $(n/x) \times n$ matrix M', where $M'[i, j]$ is the maximum entry in the j-th column of M_i.

Subcolumn queries. Consider a subcolumn query. Suppose the query is entirely contained in some M_i. This means it spans less than $x = \log n$ rows. In [14], since the desired query time was $O(\log n)$, a query simply inspected all elements of the subcolumn. In our case however, since the desired query time is only $O(\log \log n)$, we apply the above partitioning scheme twice. We explain this now.

We start with the following lemma, that provides an efficient data structure for queries consisting of a single column and *all* rows in rectangular matrices. The statement of the lemma was taken almost verbatim from the previous solution [14]. Its query time was originally stated in terms of query to a predecessor structure, but here we prefer to directly plug in the bounds implied by atomic heaps [12] (which support predecessor searches in constant time provided x is $O(\log n)$). This requires only an additional $O(n)$ time and space preprocessing.

Lemma 3 ([14]). *Given an $x \times n$ Monge matrix, a data structure of size $O(x)$ can be constructed in $O(x \log n)$ time to answer entire-column maximum queries in $O(1)$ time, if $x = O(\log n)$.*

Our new subcolumn data structure is summarized in the following theorem. It uses the above lemma and two applications of the partitioning scheme.

Theorem 3. *Given an $m \times n$ Monge matrix M, a data structure of size $O(m)$ can be constructed in $O(m \log n)$ time to answer subcolumn maximum queries in $O(\log\log(n+m))$ time.*

Proof. We first partition M into n/x matrices $M_1, M_2, \ldots, M_{n/x}$, where $x = \log m$. Every M_i is a slice of M consisting of x consecutive rows. Next, we partition every M_i into x/x' matrices $M_{i,1}, M_{i,2}, \ldots, M_{i,x'}$, where $x' = \log\log m$. Every $M_{i,j}$ is a slice of M_i consisting of x' consecutive rows (without loss of generality, assume that x divides m and x' divides x). Now we define a new $(m/x) \times n$ matrix M', where $M'[i,j]$ is the maximum entry in the j-th column of M_i. Similarly, for every M_i we define a new $(x/x') \times n$ matrix M'_i, where $M'_i[j,k]$ is the maximum entry in the k-th column of $M_{i,j}$.

We apply Lemma 3 on every M_i and $M_{i,j}$ in $O(m \log n)$ total time and $O(m)$ total space, so that any $M'[i,j]$ or $M'_i[j,k]$ can be retrieved $O(1)$ time. Furthermore, it can be easily verified that M' and all M'_is are also Monge. Therefore, we can apply Theorem 1 on M' and every M'_i. The total construction time is $O((m/x)\log(m/x)\log n + (m/x)(x/x')\log(x/x')\log n) = O(m \log n)$, and the total size of all structures constructed so far is $O((m/x)\log(m/x) + (m/x)(x/x')\log(x/x')) = O(m)$.

Now consider a subcolumn maximum query. If the range of rows is fully within a single $M_{i,j}$, the query can be answered naively in $O(x') = O(\log\log m)$ time. Otherwise, if the range of rows is fully within a single M_i, the query can be decomposed into a prefix fully within some $M_{i,j}$, an infix corresponding to a range of rows in M'_i, and a suffix fully within some $M_{i,j'}$. The maximum in the prefix and the suffix can be computed naively in $O(x') = O(\log\log m)$ time, and the maximum in the infix can be computed in $O(\log\log n)$ time using the structure constructed for M'_i. Finally, if the range of rows starts inside some M_i and ends inside another $M_{i'}$, the query can be decomposed into two queries fully within M_i and $M_{i'}$, respectively, which can be processed in $O(\log\log n)$ time as explained before, and an infix corresponding to a range of rows of M'. The maximum in the infix can be computed in $O(\log\log n)$ time using the structure constructed for M'. ☐

Submatrix queries. We are ready to present the final version of our data structure. It is based on two applications of the partitioning scheme, and an additional trick of transposing the matrix.

Theorem 4. *Given an $n \times n$ Monge matrix M, a data structure of size $O(n)$ can be constructed in $O(n \log n)$ time to answer submatrix maximum queries in $O(\log\log n)$ time.*

Proof. We partition M as described in the proof of Theorem 3, i.e., M is partitioned into n/x matrices $M_1, M_2, \ldots, M_{n/x}$, where $x = \log n$, and every M_i is then partitioned into x/x' matrices $M_{i,1}, M_{i,2}, \ldots, M_{i,x'}$, where $x' = \log\log n$.

Then we define smaller Monge matrices M' and M'_i, and provide $O(1)$ time access to their entries with Lemma 3. We apply Theorem 3 to the transpose of M' to get a subrow maximum query data structure for M'. This takes $O(n)$ space and $O(n \log n)$ time. With this data structure we can apply Theorem 2 on M', which takes an additional $O(\frac{n}{\log n} \log \frac{n}{\log n}) = O(n)$ space and $O(n \log n)$ time. We would have liked to apply Theorem 3 to the transpose of all M'_i as well, but this would require $O(n)$ space for each matrix, which we cannot afford. Since we do not have subrow maximum query data structure for the M'_is, we cannot apply Theorem 2 to them directly. However, note that the subrow maximum query data structure is used in Theorem 2 in two ways (see the proof of Lemma 2). The first use is in directly finding the subrow maximum in cases 1 and 3 in the proof of Lemma 2. In the absence of the subrow structure, we can still report the two rows containing the candidate maximum, although not the maximum itself. The second use is in computing the values for the generalized range maximum structure required to handle case 2 in that proof. In this case, we do not really need the fast query of the data structure of Theorem 3, and can use instead the slower linear space data structure from [14, Lemma 2] to compute the values in $O(n \log n)$ time. Thus, we can apply Theorem 2 to each M'_i, and get at most two candidate rows of M'_i (from cases 1 and 3), and one candidate entry of M'_i (from case 2), with the guarantee that the submatrix maximum is among these candidates.

We repeat the above preprocessing on the transpose of M. Now consider a submatrix maximum query. If the range of rows starts inside some M_i and ends inside another $M_{i'}$, the query can be decomposed into two queries fully within M_i and $M_{i'}$, respectively, and an infix corresponding to a range of rows of M'. The maximum in the infix can be computed in $O(\log \log n)$ time using the structure constructed for M'. Consequently, it is enough to show how to answer a query in $O(\log \log n)$ time when the range of rows is fully within a single M_i. In such case, if the range of rows starts inside some $M_{i,j}$ and ends inside another $M_{i,j'}$, the query can be decomposed into a prefix fully within $M_{i,j}$, an infix corresponding to a range of rows in M'_i and a suffix fully within some $M_{i,j'}$. As we explained above, even though we cannot locate the maximum in the infix exactly, we can isolate at most 2 rows (plus a single entry) of M'_i, such that the maximum lies in one of these rows. Each row of M'_i corresponds to a range of rows fully inside some $M_{i,j}$. Consequently, we reduced the query in $O(\log \log n)$ time to a constant number of queries such that the range of rows in each query is fully within a single $M_{i,j}$. Since each $M_{i,j}$ consists of $O(\log \log n)$ rows of M, we have identified, in $O(\log \log n)$ time, a set of $O(\log \log n)$ rows of M that contain the desired submatrix maximum.

Now we repeat the same procedure on the transpose of M to identify a set of $O(\log \log n)$ columns of M that contain the desired submatrix maximum. Since a submatrix of a Monge matrix is also Monge, the submatrix of M corresponding to these sets of candidate rows and columns is an $O(\log \log n) \times O(\log \log n)$ Monge matrix. By running the SMAWK algorithm [1] in $O(\log \log n)$ time on this small Monge matrix, we can finally determine the answer. □

4 Lower Bound

A predecessor structure stores a set of n integers $S \subseteq [0, U)$, so that given x we can determine the largest $y \in S$ such that $y \leq x$. As shown by Pătraşcu and Thorup [19], for $U = n^2$ any predecessor structure consisting of $O(n \operatorname{polylog}(n))$ words needs $\Omega(\log \log n)$ time to answer queries, assuming that the word size is $\Theta(\log n)$. We will use their result to prove that our structure is in fact optimal.

Given a set of n integers $S \subseteq [0, n^2)$ we want to construct $n \times n$ Monge matrix M such that the predecessor of any x in S can be found using one submatrix minimum query on M and $O(1)$ additional time (to decide which query to ask and then return the final answer). Then, assuming that for any $n \times n$ Monge matrix there exists a data structure of size $O(n \operatorname{polylog}(n))$ answering submatrix minimum queries in $o(\log \log n)$ time, we can construct a predecessor structure of size $O(n \operatorname{polylog}(n))$ answering queries in $o(\log \log n)$ time, which is not possible. The technical difficulty here is twofolds. First, M should be Monge. Second, we are working in the indexing model, i.e., the data structure for submatrix minimum queries can access the matrix. Therefore, for the lower bound to carry over, M should have the following property: there is a data structure of size $O(n \operatorname{polylog}(n))$ which retrieves any $M[i, j]$ in $O(1)$ time. Guaranteeing that both properties hold simultaneously is not trivial.

Before we proceed, let us comment on the condition $S \subseteq [0, n^2)$. While quadratic universe is enough to invoke the $\Omega(\log \log n)$ lower bound for structures of size $O(n \operatorname{polylog}(n))$, our reduction actually implies that even for larger polynomially bounded universes, i.e., $S \subseteq [0, n^c)$, for any fixed c, it is possible to construct $n \times n$ Monge matrix M such that the predecessor of x in S can be found with $O(1)$ submatrix minimum queries on M and $O(1)$ additional time (and, as previously, any $M[i, j]$ can be retrieved in $O(1)$ time with a structure of size $O(n)$). This is because any predecessor queries on a set of n integers $S \subseteq [0, n^c)$ can be reduced in $O(1)$ time to $O(1)$ predecessor queries on a set of n integers $S' \subseteq [0, n^2)$ with a structure of size $O(n)$. See full version of this paper.

The following propositions are easy to verify:

Proposition 1. *A matrix M is Monge iff $M[i, j] + M[i + 1, j + 1] \leq M[i + 1, j] + M[i, j + 1]$ for all i, j such that all these entries are defined.*

Proposition 2. *If a matrix M is Monge, then for any vector H the matrix M', where $M'[i, j] = M[i, j] + H[j]$ for all i, j, is also Monge.*

Theorem 5. *For any set of n integers $S \subseteq [0, n^2)$, there exists a data structure of size $O(n)$ returning any $M[i, j]$ in $O(1)$ time, where M is a Monge matrix such that the predecessor of x can be found using $O(1)$ time and one submatrix minimum query on M.*

Proof. We partition the universe $[0, n^2)$ into n parts $[0, n), [n, 2n), \ldots$. The i-th part $[i \cdot n, (i+1) \cdot n)$ defines a Monge matrix M_i consisting of $|S \cap [i \cdot n, (i+1) \cdot n)|$ rows and n columns. The idea is to encode the predecessor of $x \in [0, n^2)$ by the minimum element in the $(x \bmod n+1)$-th column of $M_{\lfloor x/n \rfloor}$. We first describe how these matrices are defined, and then show how to stack them together.

Consider any $0 \le i < n$. Every element in $S \cap [i \cdot n, (i+1) \cdot n) = \{a_1, a_2, \ldots, a_k\}$ has a unique corresponding row in M_i. Let $a_j = i \cdot n + a'_j$, so that $a'_1 < a'_2 < \ldots < a'_k$ and $a'_j \in [0, n)$ for all j, and also define $a'_{k+1} = n$. We describe an incremental construction of M_i. For technical reasons, we start with an artificial top row containing $1, 2, 3, \ldots, n$. Then we add the rows corresponding to $a'_1, a'_2, \ldots, a'_k$. The row corresponding to a'_j consists of three parts. The middle part starts at the $(a'_j + 1)$-th column, ends at the a'_{j+1}-th column, and contains only 1's. The elements in the left part decrease by 1 and end with 2 at the a'_j-th column, similarly the elements in the right part (if any) start with 2 at the $(a'_{j+1} + 1)$-th column and increase by 1. Formally, the k-th element of the $(j+1)$-th row, denoted $M_i[j+1, k]$, is defined as follows.

$$M_i[j+1, k] = \begin{cases} a'_j - k + 2 & \text{if } k \in [1, a'_j] \\ 1 & \text{if } k \in [a'_j + 1, a'_{j+1}] \\ k - a'_{j+1} + 1 & \text{if } k \in [a'_{j+1} + 1, n] \end{cases} \tag{1}$$

Finally, we end with an artificial bottom row containing $n, n-1, \ldots, 1$. We need to argue that every M_i is Monge. By Proposition 1, it is enough to consider every pair of adjacent rows r_1, r_2 there. Define $r'_1[j] = r_1[j] - r_1[j-1]$ and similarly $r'_2[j] = r_2[j] - r_2[j-1]$. To prove that M_i is Monge, it is enough to argue that $r'_2[j] \ge r'_1[j]$ for all $j \ge 2$. By construction, both r'_1 and r'_2 are of the form $-1, -1, \ldots, -1, 0, 0, \ldots, 0, 1, 1, \ldots, 1$, and all 0's in r'_2 are on the right of all 0's in r'_1. Therefore, M_i is Monge.

Now one can observe that the predecessor of $x \in [0, n^2)$ can be found by looking at the $(x \bmod n+1)$-th column of $M_{\lfloor x/n \rfloor}$. We check if $x < a_1$, and if so return the predecessor of a_1 in the whole S. This can be done in $O(1)$ time and $O(n)$ additional space by explicitly storing a_1 and its predecessor for every i. Otherwise we know that the predecessor of x is a_j such that $x \bmod n \in [a'_j, a'_{j+1})$, and, by construction, we only need to find $j \in [1, k]$ such that the $(x \bmod n+1)$-th element of row $j+1$ in M_i is 1. This is exactly a subcolumn minimum query.

We cannot simply concatenate all M_i's to form a larger Monge matrix. We use Proposition 2 instead. Initially, we set $M = M_0$. Then we consider every other M_i one-by-one maintaining invariant that the current M is Monge and its last row is $n, n-1, \ldots, 1$. In every step we add the vector $H = [-n+1, -n+3, \ldots, n-1]$ to the current matrix M, obtaining a matrix M' whose last row is $1, 2, \ldots, n$. By Proposition 2, M' is Monge. Then we can construct the new M by appending M_i without its first row to M'. Because the first row of M_i is also $1, 2, \ldots, n$, the new M is also Monge. Furthermore, because we add the same value to all elements in the same column of M_i, answering subcolumn minimum queries on M_i can be done with subcolumn minimum queries on the final M.

We need to argue that elements of M can be accessed in $O(1)$ using a data structure of size $O(1)$. To retrieve $M[j, k]$, first we lookup in $O(1)$ time the appropriate M_i from which it originates. This can be preprocessed and stored for every j in $O(n)$ total space and allows us to reduce the question to retrieving $M_i[j', k]$. Because Proposition 2 is applied exactly $n-1-i$ times after appending M_i to the current M, then we can return $M_i[j', k] + (n-1-i)H[k]$. To find $M_i[j', k]$, we just directly use Equation 1, which requires only storing $a'_1, a'_2, \ldots, a'_n$ in $O(n)$ total space. $\square$

References

1. A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.
2. S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *41st FOCS*, pages 198–207, 2000.
3. A. Amir, J. Fischer, and M. Lewenstein. Two-dimensional range minimum queries. In *18th CPM*, pages 286–294, 2007.
4. G. S. Brodal, P. Davoodi, M. Lewenstein, R. Raman, and S. S. Rao. Two dimensional range minimum queries and Fibonacci lattices. In *20th ESA*, pages 217–228, 2012.
5. G. S. Brodal, P. Davoodi, and S. S. Rao. On space efficient two dimensional range minimum data structures. In *18th ESA*, pages 171–182, 2010.
6. R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Appl. Math.*, 70:95–161, 1996.
7. T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *27th SOCG*, pages 354–363, 2011.
8. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17:427–462, 1988.
9. B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *5th SOCG*, pages 131–139, 1989.
10. E. D. Demaine, G. M. Landau, and O. Weimann. On Cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014.
11. A. Farzan, J. I. Munro, and R. Raman. Succinct indices for range queries with applications to orthogonal range maxima. In *39th ICALP*, pages 327–338, 2012.
12. M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.
13. H. Gabow, J. L. Bentley, and R.E Tarjan. Scaling and related techniques for geometry problems. In *16th STOC*, pages 135–143, 1984.
14. P. Gawrychowski, S. Mozes, and O. Weimann. Improved submatrix maximum queries in Monge matrices. In *41st ICALP*, pages 525–537, 2014.
15. H. Kaplan, S. Mozes, Y. Nussbaum, and M. Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *23rd SODA*, pages 338–355, 2012.
16. M. M. Klawe and D J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM Journal Discret. Math.*, 3(1):81–97, 1990.
17. Tsvi Kopelowitz and Moshe Lewenstein. Dynamic weighted ancestors. In *18th SODA*, pages 565–574, 2007.
18. Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Comput. Geom.*, 42(4):342–351, 2009.
19. Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *38th STOC*, pages 232–240, 2006.
20. H. Yuan and M. J. Atallah. Data structures for range minimum queries in multidimensional arrays. In *21st SODA*, pages 150–160, 2010.