# 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 $O(n \operatorname{polylog}(n))$. 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-Ackermann 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 [4,11,12,15,24] 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 [24]. In general matrices, this is optimal since 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 [6] and such tradeoffs can indeed be achieved for any value of $c$ [5,6].

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 the matrix $M$ is sparse, i.e., has $N = o(n^2)$ nonzero entries. In this case the problem is known in computational geometry as the *orthogonal range searching* problem on the $n \times n$ grid. In this case as well, various data structures with $\tilde{O}(N)$-space and $\tilde{O}(1)$-query appear in a long history of results including [3,9,10,13,15]. For a survey on orthogonal range searching see [22]. 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 $M[i,k] + M[j,\ell] \geq M[i,\ell] + M[j,k]$.[5] A matrix $M$ is *Totally Monotone* (or TM) if for any pair of rows $i < j$ and columns $k < \ell$ we have that if $M[i,k] \leq M[j,k]$ then $M[i,\ell] \leq M[j,\ell]$. Notice that the Monge property implies total monotonicity but the converse is not true. Whenever possible, we state our results for the more general class of TM matrices. Throughout the paper we use a top-down

* This paper is based on two preliminary papers that appeared in ICALP 2014 and ICALP 2015.
** Mozes and Weimann supported in part by Israel Science Foundation grant 794/13.
[4] The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors in $n$.
[5] Monge matrices are often defined with a $\leq$ (rather than $\geq$) in the condition. Our results apply to both definitions, as well as to minimum (rather than maximum) queries.

and left-to-right ordering of the elements of a matrix. We say that $M[i,k]$ is above $M[j,\ell]$ if $i<j$, and to the left of $M[j,\ell]$ if $k<\ell$.

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 [7] 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 [18] for more details on the history of this problem and its applications.

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 [18]. They presented an $O(n\log n)$-space data structure with $O(\log^2 n)$ query time. This was improved in [17] 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., $r(1) \le r(2) \le \ldots \le r(n)$. Columns $c$ such that $r(c-1) < r(c)$ (or $c=1$) 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 [2] (total monotonicity suffices for SMAWK).

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 $O(m)$ (as we show in Section 5), and they can be found in $O(n \cdot \alpha(n))$ time[6] using an algorithm of Klawe and Kleitman [19]. This was used in [17,18] to extend their solutions to partial Monge matrices at the cost of an additional $\alpha(n)$ factor to the query time.[7]

**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 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 $O(n \operatorname{polylog}(n))$. 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 constant number of predecessor problems, and vice versa, the lower bound reduces the predecessor problem to a submatrix query problem. In fact, the lower bound holds even for the more restricted case where the submatrix query is a subcolumn.

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[8]. 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 [17,18] store, for each node $u$ in $\mathcal{T}$ a data structure $D_u$. The goal of $D_u$ is to answer submatrix maximum queries that include an arbitrary interval of columns and *exactly all rows* of $M_u$. This way, an arbitrary query is covered in [17,18] 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 $D_u$s. 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.

[6] Here $\alpha(n)$ is the inverse-Ackermann function.

[7] In [18], there was also an additional $\log n$ factor to the space.

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

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 maximum queries on $M$. The predecessor lower bound of Pătraşcu and Thorup [23] 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.

**Computational model.** We assume the standard word RAM model with word size $\Omega(\log n)$. However, this is just an internal assumption and the elements of the matrix $M$ are only accessed through a comparison oracle, that is, we only assume that we are able to check in constant time if $M[i,j] \le M[i',j']$ and no arithmetical manipulation on the elements of $M$ is performed.

**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, and the extension to partial Monge matrices in Section 5.

## 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 $\{0, \dots, n-1\}$. It is well known that there are $O(m)$-space 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 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 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 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 $M_\ell$ 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 $M_\ell$ and the latter starts at the first row of $M_r$. We equip every node with two data structures supporting 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 suffices to show how to answer subcolumn maximum queries starting at the first row.

**Lemma 1.** *Given an $m \times n$ TM 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 then 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 TM 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. Observe 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 [20],[9] 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 $k \leq i$ be the breakpoints of $M_i$. As long as $M[i+1, c_k] \geq M[r(c_k), c_k]$ 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 $k \geq 1$ and $M[i+1, c_k] < M[r(c_k), c_k]$, we check if $M[i+1, n] \geq M[r(c_k), n]$. If so, we need to create a new breakpoint. To this end, we need to find the smallest $j$ such that $M[i+1, j] \geq M[r(c_k), j]$. 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 $m \times n$ TM 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.

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 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 $M_\ell$ 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 suffices 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.*

[9] The reduction described in [20] needs $O(\log^* m)$ additional time and (adaptively) queries two sets. The additional time is required to reduce the total size of the sets to $O(m)$, which is done by recursively decomposing the tree. However, this recursive decomposition can be avoided using atomic heaps as explained in Lemma 11 of [16]. Then in $O(1)$ additional time we are able to reduce a weighted ancestor query to a single predecessor query in one of the sets.

*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 $O(pred(n, m))$ time, 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 $(i, i']$ over the precomputed values $m_i$ associated to the breakpoints. 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 $m_i$. 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 [12] 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. □

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 $\mathcal{T}$. We thus obtained the following theorem.

**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 [17]. 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 [17], 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.

**Lemma 3 (the micro data structure).** *Given an $x \times n$ TM matrix and $r > 0$, one can construct in $O(x \log n / \log r)$ time, a data structure of size $O(x)$ that given a query column can report the maximum entry in the entire column in $O(r + pred(x, n))$ time.*

*Proof.* Out of all $n$ columns of the input matrix $M$, we will designate $O(x)$ columns as *special* columns. For each of these special columns we will eventually compute its maximum element. The first $x$ special columns of $M$ are columns $1, n/x, 2n/x, 3n/x, \ldots, n$ and are denoted $j_1, \ldots, j_x$.

Let $X$ denote the $x \times x$ submatrix obtained by taking all $x$ rows but only the $x$ special columns $j_1, \ldots, j_x$. It is easy to verify that $X$ is TM. We can therefore run the SMAWK algorithm [2] on $X$ in $O(x)$ time and obtain the column maxima of all special columns. Let $r(j)$ denote the row containing the maximum element in column $j$ [10] Since $M$ is TM, the $r(j)$ values are monotonically non-decreasing. Consequently, $r(j)$ of a non-special column $j$ must be between $r(j_i)$ and $r(j_{i+1})$ where $j_i < j$ and $j_{i+1} > j$ are the two special columns bracketing $j$ (see Figure 1).

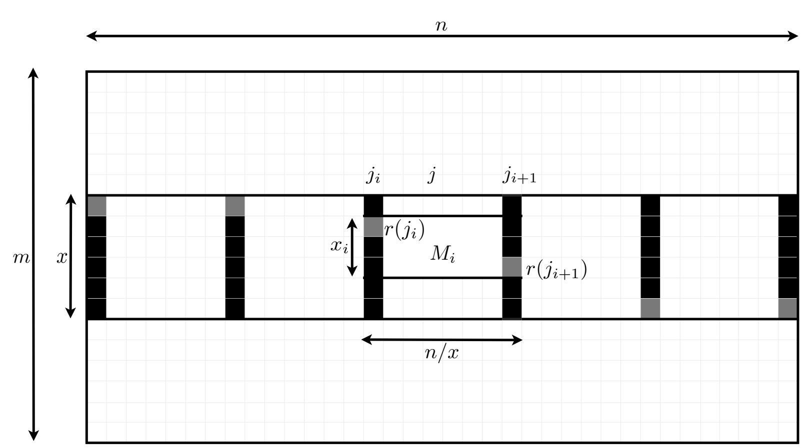


**Fig. 1.** An $x \times n$ matrix inside an $m \times n$ matrix. The black columns are the first $x$ special columns. The (monotonically non-decreasing) gray cells inside these special columns are the column maxima (i.e., the $r(j_i)$ values of breakpoints $j_i$). The maximum element of column $j$ in the $x \times n$ matrix must be between $r(j_i)$ and $r(j_{i+1})$ (i.e., in matrix $M_i$).

For every $i$, let $x_i = r(j_{i+1}) - r(j_i)$. If $x_i \le r$ then *no* column between $j_i$ and $j_{i+1}$ will ever be a special column. When we will query such a column $j$ we can simply check (at query-time) the $r$ elements of $j$ between rows $r(j_i)$ and $r(j_{i+1})$ in $O(r)$ time. If, however, $x_i > r$, then we designate more special columns between $j_i$ and $j_{i+1}$. This is done recursively on the $x_i \times (n/x)$ matrix $M_i$ composed of rows $r(j_i), \ldots, r(j_{i+1})$ and columns $j_i, \ldots, j_{i+1}$. That is, we mark $x_i$ evenly-spread columns of $M_i$ as special columns, and run SMAWK in $O(x_i)$ time on the $x_i \times x_i$ submatrix $X_i$ obtained by taking all $x_i$ rows but only these $x_i$ special columns. We continue recursively until either $x_i \le r$ or the number of columns in $M_i$ is at most $r$. In the latter case, before terminating, the recursive call runs SMAWK in $O(x_i + r) = O(x_i)$ time on the $x_i \times r$ submatrix $X_i$ obtained by taking the $x_i$ rows and *all* columns of $M_i$ (i.e., all columns of $M_i$ will become special).

[10] We assume that no elements of the matrix are equal. The ties are resolved lexicographically.

After the recursion terminates, every column $j$ of $M$ is either special (in which case we computed its maximum), or its maximum is known to be in one of at most $r$ rows (these rows are specified by the $r(\cdot)$ values of the two special columns bracketing $j$). Let $s$ denote the total number of columns that are marked as special. We claim that $s = O(x \log n / \log r)$. To see this, notice that the number of columns in every recursive call decreases by a factor of at least $r$ and so the recursion depth is $O(\log_r n) = O(\log n / \log r)$. In every recursive level, the number of added special columns is $\sum x_i$ over all $x_i's$ in this level that are at least $r$. In every recursive level, this sum is bounded by $2x$ because each one of the $x$ rows of $M$ can appear in at most two $M_i$'s (as the last row of one and the first row of the other). Overall, we get $2x \cdot O(\log n / \log r) = O(x \log n / \log r)$.

Notice that $s = O(x \log n / \log r)$ implies that the total time complexity of the above procedure is also $O(x \log n / \log r)$. This is because whenever we run SMAWK on a $y \times y$ matrix it takes $O(y)$ time and $y$ new columns are marked as special. To complete the construction, we go over the $s$ special columns from left to right in $O(s)$ time and throw away (mark as non-special) any column whose $r(\cdot)$ value is the same as that of the preceding special column. This way we are left with only $O(x)$ special columns, and the difference in $r(\cdot)$ between consecutive special columns is at least 1 and at most $r$. In fact, it is easy to maintain $O(x)$ (and not $O(s)$) space *during* the construction by only recursing on sub matrices $M_i$ where $x_i > 1$. We note that when $r = 1$, the eventual special columns are exactly the set of breakpoints of the input matrix $M$.

The final data structure is a predecessor data structure that holds the $O(x)$ special columns and their associated $r(\cdot)$ values. Upon query of some column $j$, we search in $pred(x, n)$ time for the predecessor and successor of $j$ and obtain the two $r(\cdot)$ values. We then search for the maximum of column $j$ by explicitly checking all the (at most $r$) relevant rows of column $j$. The query time is therefore $O(r + pred(x, n))$ and the space $O(x)$. ◻

In the case of $x = O(\log n)$, using atomic heaps [14] (which support predecessor searches in constant time) we obtain the following corollary:

**Corollary 1.** *Given an $x \times n$ TM 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 $x = O(\log n)$.*

It is possible to use Lemma 3 to obtain a subcolumn data structure with faster $O(n \log n / \log \log n)$ preprocessing time, at the cost of slower $O(\log n)$ query time (cf. [17, Lemma 2]). We next describe our new subcolumn data structure, which uses the above corollary 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 $m/x$ matrices $M_1, M_2, \ldots, M_{m/x}$, 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 Corollary 1 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 in $O(1)$ time. Furthermore, it can be easily verified that $M'$ and all $M_i'$s are also Monge. To prove this, it is enough to argue that if $N$ is an $4 \times 2$ Monge matrix, the $2 \times 2$ matrix $N'$ created by partitioning $N$ into two slices, each consisting of two rows, whose elements are the maxima in every column of each slice, is also Monge. To this end, we need to compare:

$$N'[1, 1] + N'[2, 2] = \max(N[1, 1], N[2, 1]) + \max(N[3, 2], N[4, 2])$$

and

$$N'[1, 2] + N'[2, 1] = \max(N[1, 2], N[2, 2]) + \max(N[3, 1], N[4, 1]).$$

Let $\max(N[1, 2], N[2, 2]) = N[i, 2]$, where $i \in \{1, 2\}$, and similarly $\max(N[3, 1], N[4, 1]) = N[i', 1]$, where $i' \in \{3, 4\}$. Then

$$(N'[1, 1] + N'[2, 2]) - (N'[1, 2] + N'[2, 1]) \geq (N[i, 1] + N[i', 2]) - (N[i, 2] + N[i', 1])$$

which is at least 0 because of $N$ being Monge.

Therefore, because $M'$ and all $M'_i$ are all Monge, and by Corollary 1 their entries can be accessed in $O(1)$ time, 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 Corollary 1. 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 also apply Theorem 3 to the transpose of the $\frac{n}{\log\log n}$-by-$n$ matrix obtained by stacking the $\frac{n}{\log n}$ $M'_i$ matrices. This takes $O(n)$ space and $O(n\log n)$ time. This serves as a subrow maximum data structure for each $M'_i$, so we can apply Theorem 2 to each $M'_i$ separately, which takes a total of $O(\frac{n}{\log n}\frac{\log n}{\log\log n}\log(\frac{\log n}{\log\log n})) = O(n)$ additional space and $O(n\log\log n)$ time.

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 $M_{i,j'}$. The query on the infix can be answered using the data structure for $M'_i$. Consequently, we reduced the query in $O(\log\log n)$ time to four 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 $M$, by taking the union of the rows of $M$ corresponding to all these $M_{i,j}$'s and also including the row containing the maximum in the infixes, 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 [2] 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 $y \le x$. As shown by Pătraşcu and Thorup [23], 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 maximum 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 maximum 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 twofold. First, $M$ should be Monge. Second, we are working in the indexing model, i.e., the data structure for submatrix maximum 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 maximum 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 a consequence of the following lemma.

**Lemma 4.** *For any constant $c \geq 2$, 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 $O(n)$.*

*Proof.* First we describe a weaker version of the reduction for $c = 4$, where the resulting set of integers is $S' \subseteq [0, 3n^2)$.

Let $S = \{x_1, x_2, \ldots, x_n\}$. We represent every $x_i$ in base $n^2$ as $x_i = y_i \cdot n^2 + z_i$, where $y_i, z_i \in [0, n^2)$. We create a new set $Y \subseteq [0, n^2)$ storing all $y_i$s and a new set $Z \subseteq [0, n^2)$ storing all $z_i$s. Let $\operatorname{rank}_Y(x)$ and $\operatorname{rank}_Z(x)$ denote the rank of $x$ in $Y$ and $Z$, respectively, where rank is the number of smaller elements in the set. We create another set $R \subseteq [0, n^2)$ storing elements of the form $\operatorname{rank}_Y(y_i) \cdot n + \operatorname{rank}_Z(z_i)$. To find the predecessor of $x$ in $S$, we first represent it as $x = y \cdot n^2 + z$. We claim that it is always possible to reduce locating the predecessor of $x$ in $S$ to the case where $y \in Y$ and $z \in Z$ in two steps. Let $y'$ denote the predecessor of $y$ in $Y$ and $z'$ denote the predecessor of $z$ in $Z$.

1. If $z'$ is not defined, we decrease $y$ by one (adjusting $y'$ if necessary) and replace $z$ by the largest element of $Z$. Otherwise, we replace $z$ by $z'$.
2. If $y'$ is not defined, $x$ has no predecessor in $S$. Otherwise, if $y' < y$ we replace $y$ by $y'$ and $z$ by the largest element of $Z$.

Both steps maintain the predecessor of $x$ in $S$ and take $O(1)$ time. Finally, having reduced the general case so that $y \in Y$ and $z \in Z$, we locate the predecessor of $x' = \operatorname{rank}_Y(y) \cdot n + \operatorname{rank}_Z(z)$ in $R$. The predecessor of $x'$ in $R$ corresponds to the predecessor of $x$ in $S$, because comparing two elements of the same set is equivalent to comparing their ranks there. Formally, $x_i \leq x$ iff $y_i < y$ or $y_i = y$ and $z_i \leq z$, which is equivalent to $\operatorname{rank}_Y(y_i) < \operatorname{rank}_Y(y)$ or $\operatorname{rank}_Y(y_i) = \operatorname{rank}_Y(y)$ and $\operatorname{rank}_Z(z_i) \leq \operatorname{rank}_Z(z)$, which because the ranks are all from $[0, n)$ can be stated as $\operatorname{rank}_Y(y_i) \cdot n + \operatorname{rank}_Z(z_i) \leq \operatorname{rank}_Y(y) \cdot n + \operatorname{rank}_Z(z)$. Consequently, a predecessor query on $S$ can be reduced into one predecessor query into each of $Y, Z, R$. These three sets can be combined into a single set $S' \subseteq [0, 3n^2)$, such that predecessor queries in either of them can be answered with predecessor queries on $S'$, by simply shifting every element of $Z$ by $n^2$ and every element of $R$ by $2n^2$. Finally, the size of $S'$, which is up to $3n$ right now, can be reduced to $n$ by storing every third element in the predecessor structure. Knowing the predecessor of $x$ among these chosen elements allows us to find the true predecessor in $O(1)$ time by inspecting at most three elements stored explicitly for every element of the reduced $S'$.

Now we explain how to extend the above reduction for any constant $c \geq 2$, while also ensuring that the resulting set of integers is $S' \subseteq [0, n^c)$. If $n \geq 3$, by modifying the reduction so that every $x_i$ is represented as $y_i \cdot n^2 + z_i$, where $y_i \in [0, n^{c-2})$ and $z_i \in [0, n^2)$, we obtain a set of $n$ integers from $[0, n^{c-1})$. Hence, by iterating $c - 3$ times we finally obtain a set of $n$ integers from $[0, n^3)$. Then one final iteration, where we represent every $x_i$ as $y_i \cdot B^2 z_i$, with $y_i, z_i \in [0, B^2)$ with $B = \lceil n^{1/4} \rceil$, allows us to reduce the size of the universe to $B^6$, which is at most $n^2$ for $n \geq 48$. For smaller $n$ we answer predecessor queries naively in $O(1)$ time. ☐

The following propositions are easy to verify:

**Proposition 1.** *An $m \times n$ matrix $M$ is Monge iff $M[i,j] + M[i+1,j+1] \geq M[i+1,j] + M[i,j+1]$ for all $i = 1,2,\ldots,m-1$ and $j = 1,2,\ldots,n-1$.*

**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 maximum 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 $2 + |S \cap [i \cdot n, (i+1) \cdot n)|$ rows and $n$ columns. The first and the last row are artificial, and others encode the elements of $S \cap [i \cdot n, (i+1) \cdot n)$. The idea is to encode the predecessor of $x \in [0, n^2)$ by the maximum 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 $0 \leq i < n$. 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 $n-1, n-2, \ldots, 1$. 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 $n$'s. The elements in the left part increase by 1 and end with $n-1$ at the $a'_j$-th column, similarly the elements in the right part (if any) start with $n-1$ at the $(a'_{j+1}+1)$-th column and decrease by 1. Formally, the $k$-th element of the $(j+1)$-th row, denoted $M_i[j+1, k]$, is defined as follows.

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

Finally, we end with an artificial bottom row containing $1, 2, \ldots, n$. See Figure 2 for an example. 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 $r'_2[j] \geq r'_1[j]$ for all $j \geq 2$. 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 $n$. This is exactly a subcolumn maximum 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 $1, 2, \ldots, n$. 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 $n, n-1, \ldots, 1$. 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 $n-1, n-2, \ldots, 1$, 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 maximum queries on $M_i$ can be done with subcolumn maximum queries on the final $M$. The right side of Figure 2 depicts the final Monge matrix $M$.

We need to argue that elements of $M$ can be accessed in $O(1)$ time using a data structure of size $O(n)$. 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 $M_i[j', k] + (n-1-i) \cdot H[k]$. 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. $\square$

$$M_0 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 5&6&7&8&8&8&8&8 \\ 1&2&3&4&5&6&7&8 \end{bmatrix} \quad M_4 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 1&2&3&4&5&6&7&8 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 1&2&3&4&5&6&7&8 \end{bmatrix} \quad M_5 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 6&7&8&8&8&8&7&6 \\ 2&3&4&5&6&7&8&8 \\ 1&2&3&4&5&6&7&8 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 6&7&8&8&8&7&6&5 \\ 3&4&5&6&7&8&7&6 \\ 2&3&4&5&6&7&8&8 \\ 1&2&3&4&5&6&7&8 \end{bmatrix} \quad M_6 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 1&2&3&4&5&6&7&8 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 1&2&3&4&5&6&7&8 \end{bmatrix} \quad M_7 = \begin{bmatrix} 8&7&6&5&4&3&2&1 \\ 7&8&8&8&7&6&5&4 \\ 4&5&6&7&8&8&8&8 \\ 1&2&3&4&5&6&7&8 \end{bmatrix}$$

$$M = \begin{bmatrix} 57&42&27&12&-3&-18&-33&-48 \\ 54&41&28&15&1&-13&-27&-41 \\ 50&37&24&11&-2&-15&-28&-41 \\ 43&32&21&10&-1&-12&-23&-34 \\ 41&32&23&13&3&-8&-19&-30 \\ 38&29&20&11&2&-7&-18&-29 \\ 37&28&19&10&1&-8&-17&-27 \\ 36&27&18&9&0&-9&-18&-27 \\ 29&22&15&8&1&-6&-13&-20 \\ 22&17&12&7&2&-3&-8&-13 \\ 20&17&14&10&6&2&-3&-8 \\ 16&13&10&7&4&1&-2&-6 \\ 15&12&9&6&3&0&-3&-6 \\ 8&7&6&5&4&3&2&1 \\ 7&8&8&8&7&6&5&4 \\ 4&5&6&7&8&8&8&8 \\ 1&2&3&4&5&6&7&8 \end{bmatrix}$$

**Fig. 2.** Reduction for $n = 8$ and $S = \{8 \cdot 0 + 3, 8 \cdot 2 + 2, 8 \cdot 2 + 5, 8 \cdot 2 + 6, 8 \cdot 5 + 2, 8 \cdot 5 + 6, 8 \cdot 7 + 1, 8 \cdot 7 + 4\}$.

## 5 Data structure for partial Monge matrices

Our goal in this section is to extend the solution described in Section 3 to *partial* Monge matrices. Recall that in a partial Monge matrix $M$, for any $i < j$ and $k < \ell$, the condition $M[i,k] + M[j,\ell] \geq M[i,\ell] + M[j,k]$ holds only if all of $M[i,k], M[j,\ell], M[i,\ell], M[j,k]$ are defined. Not all entries in $M$ are defined, but the defined entries in every row and every column are contiguous. Let $s_i$ and $t_i$ denote the first and last columns containing defined entries in the $i$'th row respectively. We assume that we know the coordinates of at least one of the defined entries. This allows us to find all $s_i$'s and $t_i$'s in $O(n \log n)$ time.

The following Lemma states that we can implicitly fill appropriate constants instead of the undefined (blank) entries to turn a partial Monge matrix into a full Monge matrix:

**Lemma 5.** *The blank entries in an $m \times n$ partial Monge matrix $M$ can be implicitly replaced so that $M$ becomes Monge and each $M_{ij}$ can be returned in $O(1)$ time.*

*Proof.* Let $s_i$ (resp. $t_i$) denote the index of the leftmost (resp. rightmost) column that is defined in row $i$. Since the defined (non-blank) entries of each row and column are continuous we have that the sequence $s_1, s_2, \ldots, s_m$ starts with a non-increasing prefix $s_1 \geq s_2 \geq \ldots \geq s_a$ and ends with a non-decreasing suffix $s_a \leq s_{a+1} \leq \ldots \leq s_m$. Similarly, the sequence $t_1, t_2, \ldots, t_n$ starts with a non-decreasing prefix $t_1 \leq t_2 \leq \ldots \leq t_b$ and ends with a non-increasing suffix $t_b \geq t_{b+1} \geq \ldots \geq t_m$.

We partition the blank region of $M$ into four regions: (I) entries that are above and to the left of $M[i, s_i]$ for $i = 1, \ldots, a$, (II) entries that are below and to the left of $M[i, s_i]$ for $i = a+1, \ldots, m$, (III) entries that are above and to the right of $M[i, t_i]$ for $i = 1, \ldots, b$, (IV) entries that are below and to the right of $M[i, t_i]$ for $i = b+1, \ldots, n$. We first describe how to replace all entries in region I to make them non-blank and obtain a valid partial Monge matrix (whose blank entries are only in regions II, III, and IV).

Let $W$ denote the largest absolute value of any entry in $M$ (We can find $W$ by applying the algorithm of Klawe and Kleitman [19]). Intuitively, we would like to make every $M[i,j]$ in region I very large. However, we cannot simply assign the same large value to each of them, because then the Monge inequality would not be guaranteed to hold if more than one of the four considered elements belongs to the replaced part of the matrix. A closer look at all possible cases shows that we need to make these large values substantially decreasing in both rows and columns. We replace every $M[i,j]$ in region I with $2W(m-i+n-j)$ to obtain a new matrix $M'$. To prove that the resulting new matrix $M'$ is partial Monge, consider any $i<j$ and $k<\ell$ such that all entries $M'[i,k], M'[j,\ell], M'[i,\ell], M'[j,k]$ are defined. We must verify that $M'[i,k]+M'[j,\ell]-M'[i,\ell]-M'[j,k] \geq 0$. To this end we consider the following cases:

1. All $M[i,k], M[j,\ell], M[i,\ell], M[j,k]$ are non-blank, then the inequality holds because $M$ is partial Monge.
2. $M[i,k]$ is blank and $M[j,\ell], M[i,\ell], M[j,k]$ are non-blank, then $M'[i,k]+M'[j,\ell]-M'[i,\ell]-M'[j,k] \geq M'[i,k]-3W \geq 4W-3W \geq 0$.
3. $M[i,k], M[i,\ell]$ are blank and $M[j,k], M[j,\ell]$ are non-blank, then $M'[i,k]+M'[j,\ell]-M'[i,\ell]-M'[j,k] \geq 2W(j-k)-2W \geq 0$.
4. $M[i,k], M[j,k]$ are blank and $M[i,\ell], M[j,\ell]$ are non-blank, then $M'[i,k]+M'[j,\ell]-M'[i,\ell]-M'[j,k] \geq 2W(j-i)-2W \geq 0$.
5. $M[i,k], M[i,\ell], M[j,k]$ are blank and $M[j,\ell]$ is non-blank, then $M'[i,k]+M'[j,\ell]-M'[i,\ell]-M'[j,k] \geq 2W(i+\ell)-W \geq 0$.

Hence the new matrix $M'$ is indeed partial Monge.

By a similar reasoning, we can replace every $M[i,j]$ in region II with $2W(i+j)$, every $M[i,j]$ in region III with $2W(i+n-j)$, and every $M[i,j]$ in region IV with $2W(m-i+j)$. Note that such a replacement is correct only if done for each region separately, hence to make $M$ full Monge we first need to fill the blanks in region I, then calculate the new value of $W$ and fill the blanks in region II accordingly, and so on. ◻

For subcolumn (or subrow) maximum queries, the above lemma implies that we can handle partial Monge matrices in the same bounds as full Monge matrices (i.e., the bounds of Theorem 3 and Corollary 1 also apply to partial Monge matrices). Upon subcolumn query (a column $c$ and a range of rows $R$) we first restrict $R$ to the defined entries in the column $c$ and only then query the data structure.

For submatrix queries however, this trick only works if the query range is entirely defined. In general, it does not work because the defined entries in the query range do not necessarily form a submatrix. Handling submatrix queries is therefore more complicated. Our solution is based on the following decomposition.

### 5.1 Decomposing a partial Monge matrix into staircase matrices

Our data structure relies on a decomposition of $M$ into *staircase* matrices. A partial matrix is staircase if the defined entries in its rows either all begin in the first column or all end in the last column. It is well known (cf. [1]) that by cutting $M$ along columns and rows, it can be decomposed into staircase matrices $\{M_i\}$ such that each row is covered by at most two matrices, and each column is covered by at most three matrices. For completeness, we describe such a decomposition below.

**Lemma 6.** *A partial matrix $M$ can be decomposed into staircase matrices $\{M_i\}$ such that each row is covered by at most two matrices, and each column is covered by at most three matrices.*

*Proof.* Let $s_i$ and $t_i$ denote the smallest and largest column index in which an element in row $i$ is defined, respectively. The fact that the defined entries of $M$ are contiguous in both rows and columns implies that the sequence $s_1, s_2, \ldots, s_m$ consists of a non-increasing prefix and a non-decreasing suffix. Similarly, the sequence $t_1, t_2, \ldots, t_m$ consists of a non-decreasing prefix and a non-increasing suffix. It follows that the rows of $M$ can be divided into three ranges - a prefix where $s$ is non-increasing and $t$ is non-decreasing, an infix where both $s$ and $t$ have the same monotonicity property, and a suffix where $s$ is non-decreasing and $t$ is non-increasing. The defined entries in the prefix of the rows can be divided into two staircase matrices by splitting $M$ at $t_1$, the largest column where the first row has a defined entry. Similarly, the defined entries in the suffix of

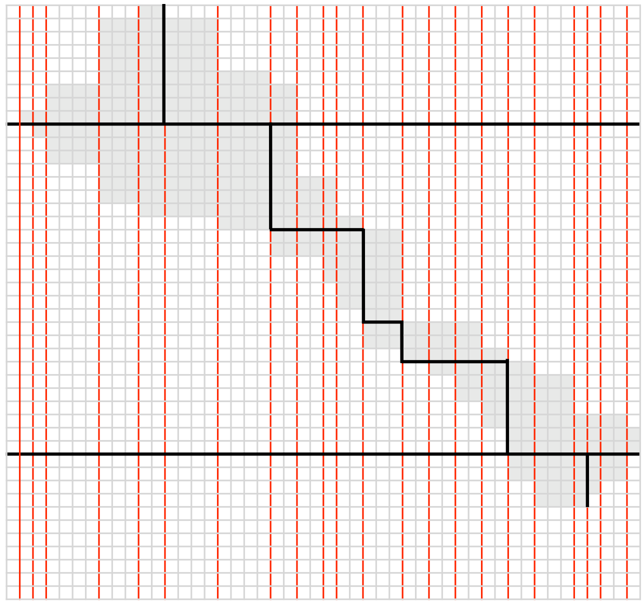

**Fig. 3.** A decomposition of a partial matrix (where the defined entries are gray and the undefined white) into staircase matrices (defined by solid thick black lines) and into blocks of consecutive columns with the same defined entries (indicated by thin vertical red lines).

the rows can be divided into two staircase matrices by splitting it at $t_m$, the largest column where the last row has a defined entry. The defined entries in the infix of the rows form a double staircase matrix. It can be broken into staircase matrices by dividing along alternating rows and columns as shown in Figure 3.

It is easy to verify that, in the resulting decomposition, each row is covered by at most two staircase matrices, and each column is covered by at most three staircase matrices. Also note that every set of consecutive columns whose defined elements are in exactly the same set of rows are covered in this decomposition by the same three row-disjoint staircase matrices. □

Before we use the above decomposition for our data structure, we show how it can be used to prove that, if $M$ is an $m \times n$ TM (or Monge) staircase matrix, then the number of breakpoints of $M$ is $O(m)$. This result illustrates the use of the decomposition, it was used in the data structure of [17], and we believe is of independent interest.

**Theorem 6.** *Let $M$ be a partial $m \times n$ matrix in which the defined entries in each row and in each column are contiguous. If $M$ is TM (i.e., for all $i < j, k < \ell$ where $M[i,k], M[i,\ell], M[j,k], M[j,\ell]$ are all defined, $M[i,k] \leq M[j,k] \implies M[i,\ell] \leq M[j,\ell]$), then the number of breakpoints of $M$ is $O(m)$.*

*Proof.* We first show that the number of breakpoints of an $m \times n$ TM *staircase* matrix is at most $2m$. We focus on the case where the defined entries of all rows begin in the first column and end in non-decreasing columns. In other words, for all $i$, $s_i{=}1$ and $t_i \leq t_{i+1}$. The other cases are symmetric.

A breakpoint is a situation where the maximum in column $c$ is at row $r_1$ and the maximum in column $c+1$ is at a different row $r_2$. We say that $r_1$ is the departure row of the breakpoint, and $r_2$ is the entry row of the breakpoint. There are two types of breakpoints: decreasing ($r_1 < r_2$), and increasing ($r_1 > r_2$). We show that (1) each row can be the entry row of at most one decreasing breakpoint, and (2) each row can be the departure row of at most one increasing breakpoint.

(1) Assume that row $r_2$ is an entry row of two decreasing breakpoints: One is the pair of entries $(r_1, c_1), (r_2, c_1 + 1)$ and the other is the pair $(r_3, c_2), (r_2, c_2 + 1)$. We know that $r_1 < r_2$, $r_3 < r_2$, and

wlog $c_2 > c_1+1$. Since the maximum in column $c_1+1$ is in row $r_2$, we have $M[r_3, c_1+1] < M[r_2, c_1+1]$. However, since the maximum in column $c_2$ is in row $r_3$, we have $M[r_3, c_2] > M[r_2, c_2]$, contradicting the total monotonicity of $M$. Note that $M[r_2, c_2]$ is defined since $M[r_2, c_2+1]$ is defined.

(2) Assume that row $r_1$ is a departure row of two increasing breakpoints: One is the pair of entries $(r_1, c_1), (r_2, c_1+1)$ and the other is the pair $(r_1, c_2), (r_3, c_2+1)$. We know that $r_1 > r_2$ and $r_1 > r_3$. Since the maximum in column $c_1$ is in row $r_1$, we have $M[r_2, c_1] < M[r_1, c_1]$. However, since the maximum in column $c_1+1$ is in row $r_2$, we have $M[r_2, c_1+1] > M[r_1, c_1+1]$, contradicting the total monotonicity of $M$. Note that $M[r_1, c_1+1]$ is defined since $M[r_1, c_2]$ is defined.

The above two claims prove that the number of breakpoints of a staircase matrix is at most $2m$. We use this fact, and the above decomposition to staircase matrices to prove an $O(m)$ bound for arbitrary partial matrices.

Let $bp(M_i)$ denote the number of breakpoints in matrix $M_i$. Let $m_i$ denote the number of rows in $M_i$. Since each row appears in at most two $M_i$s, $\sum_i m_i = O(m)$. The total number of breakpoints in all $M_i$s is $O(m)$ since $\sum_i bp(M_i) = \sum_i O(m_i) = O(m)$.

Consider now a partition of $M$ into rectangular blocks $B_j$ defined by maximal sets of contiguous columns whose defined entries are at the same set of rows, see Figure 3. There are $O(m)$ such blocks. Notice that the number of breakpoints of $M$ is $bp(M) = \sum_j bp(B_j) + O(m)$ (the $O(m)$ term accounts for the possibility of a new breakpoint between every two consecutive blocks). Therefore, it suffices to bound $\sum_j bp(B_j)$.

Consider some block $B_j$. As we mentioned above, the columns of $B_j$ appear in at most three row-disjoint staircase matrices $M_1, M_2, M_3$, in the decomposition of $M$. The column maxima of $B_j$ are a subset of the column maxima of $M_1, M_2, M_3$. Assume wlog that the indices of rows covered by $M_i$ are smaller than those covered by $M_{i+1}$ for every $i = 1, 2$.

The breakpoints of $B_j$ are either breakpoints of $M_1, M_2, M_3$, or breakpoints that occur when the maxima in consecutive columns of $B_j$ originate in different $M_i$. However, since $B_j$ is a (non-partial) TM matrix, its column maxima are monotone. So once a column maximum originates in $M_i$, no maximum in greater columns will ever originate in $M_j$ for $j < i$. It follows that the number of breakpoints in $B_j$ that are not breakpoints of $M_1, M_2, M_3$ is at most two. Since there are $O(m)$ blocks, $\sum_j bp(B_j) \le \sum_i bp(M_i) + O(m) = O(m)$. This completes the proof of Theorem 6. ◻

### 5.2 The data structure

We begin with a weaker result (Theorem 7), which is that one can answer submatrix maximum queries on an $n \times n$ staircase matrix in $O(\log \log n)$ time with a structure of size $O(n \log n)$. We will then (Theorem 8) show how to reduce the space to $O(n)$, and finally (Theorem 9) how to handle arbitrary partial Monge matrices using the decomposition into staircase matrices.

We will need the following preliminary lemma, that follows quite easily from the persistent predecessor structure of Chan [8].

**Lemma 7.** *A collection $S$ of $O(n)$ weighted points on an $n \times n$ grid can be preprocessed in $O(n \log \log n)$ time and $O(n)$ space, so that, given any $(x, y)$, the maximum weight of a point $(x', y') \in S$ such that $x' \ge x$ and $y' \ge y$ can be calculated in $O(\log \log n)$ time.*

*Proof.* We use the standard geometric idea of sweeping the grid with a horizontal line while maintaining a data structure describing the current situation. The data structure is made partially persistent so that after sweeping, given a query $(x, y)$, we can retrieve the version of the structure corresponding to a horizontal line passing through $(x, y)$. Querying that version of the data structure will allow us to answer the request. The data structure will be a predecessor structure made persistent using the result of Chan [8]. See Theorem 5 of [21] for a more detailed description of a similar lemma.

Denote the points by $(x_i, y_i)$ and their corresponding weights by $w_i$. We assume that the weights are distinct. We sweep the grid with a horizontal line starting at $y = n$. The predecessor structure stores $x$-coordinates of some of the already seen points. Coordinate $x_i$ is stored in the predecessor structure iff $y_i \ge y$ and there is no $i'$ such that $y_{i'} \ge y$, $x_{i'} \ge x_i$ and $w_{i'} > w_i$. This is because otherwise the $i'$-th point is a better answer than the $i$-th point for any query processed using this or any future version of the data structure. Consequently, the points whose $x$-coordinates are stored in the predecessor structure can be arranged so that their $x$-coordinates are increasing and the weights decreasing. Then it follows that locating the maximum weight of a point $(x', y') \in S$ such that $x' \geq x$ and $y' \geq y$ can be done by finding the successor of $x$ in the version of the predecessor structure corresponding to $y$. Maintaining the structure while sweeping the grid is also done with a predecessor search. After having seen a new point $(x_i, y_i)$ we locate the predecessor of $x_i$. If the weight of the corresponding point is smaller than $w_i$, we remove it from the structure and repeat.

A persistent predecessor search structure can be implemented in space $O(n)$ while keeping the query time $O(\log \log n)$ [8]. Consequently, we can build in $O(n \log \log n)$ time a structure of size $O(n)$ answering queries in $O(\log \log n)$ time. ◻

**Theorem 7.** *Given an $n \times n$ staircase Monge matrix $M$, a data structure of size $O(n \log n)$ can be constructed in $O(n \log n)$ time to answer submatrix maximum queries in $O(\log \log n)$ time.*

*Proof.* Because of left-right symmetry, we can assume that the defined entries in row $i$ start in the first column and end in column $t_i$. Notice that either $t_1 \leq t_2 \leq \ldots \leq t_n$ or $t_1 \geq t_2 \geq \ldots \geq t_n$. Without loss of generality we will assume the latter. This is enough because we will not be explicitly using the Monge property in our solution, except for applying Theorem 4 on a copy of $M$ (called $\widetilde{M}$) where the undefined entries are appropriately filled.

We partition $M$ into full Monge matrices using a standard method: First, create a full Monge matrix by taking the upper-left fragment $[1, n/2] \times [1, t_{n/2}]$ of $M$. Then, recursively decompose the staircase matrices created by taking the upper-right fragment $[1, n/2] \times [t_{n/2}+1, n]$ and the lower-left fragment $[n/2+1, n] \times [1, n]$ of $M$. See Figure 4. It is easy to verify that the decomposition consists of at most $2n$ full Monge matrices (called fragments). The decomposition has other useful properties on which we elaborate further.

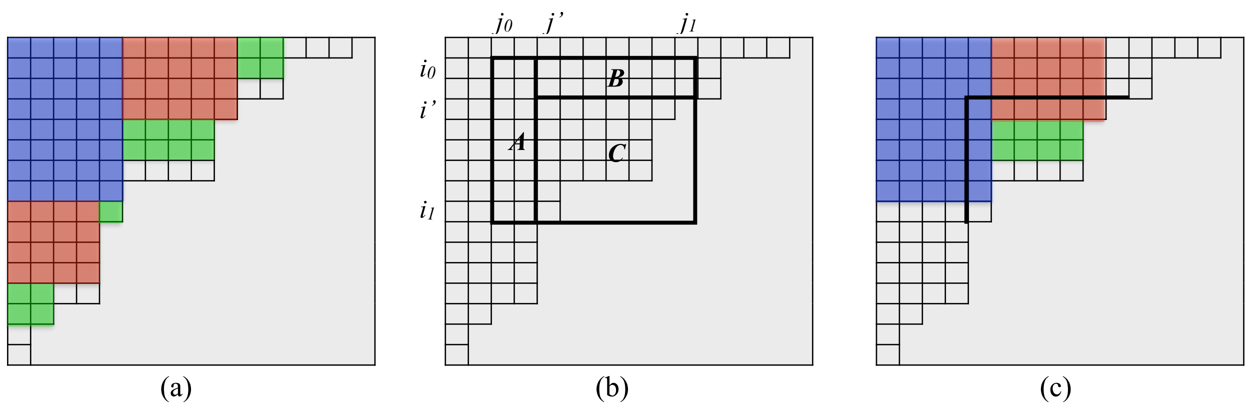


**Fig. 4.** (a) A staircase $n \times n$ Monge matrix partitioned into $2n$ smaller full Monge matrices (fragments). (b) A query range $[i_0, i_1] \times [j_0, j_1]$ decomposed into two full Monge matrices $A$ and $B$ and one dominance query $C$. (c) The dominance query as vertical and horizontal lines (the green fragment is fully inside the range and the blue and red fragment intersect the horizontal line).

Consider a query range $[i_0, i_1] \times [j_0, j_1]$. To find the maximum (defined) $M[i, j]$ over all $i \in [i_0, i_1]$ and $j \in [j_0, j_1]$ we proceed as follows. The simple case is when the query range is fully within the defined part of $M$. To handle this case, we apply Theorem 4 on a copy of $M$ (denoted $\widetilde{M}$) where the undefined entries are appropriately (and implicitly) filled using Lemma 5. This allows us to do submatrix queries in $O(\log \log n)$ time when the query range is fully defined. Otherwise, we decompose the query into three parts. The first part, which we call a *dominance maximum query*, is to find the maximum $M[i, j]$ over all $i \geq i'$ and $j \geq j'$, for $i', j'$ to be defined shortly. The other two are submatrix maximum queries fully within the defined part of $M$ (and hence can be processed by querying the structure built for $\widetilde{M}$ in $O(\log\log n)$ time). The decomposition is performed in $O(1)$ time by setting $j' = t_{i_1+1} + 1$ and choosing the smallest $i' \geq i_0$ such that $t_{i'} < j_1$ (which can be preprocessed for every possible $j_1$ in $O(n)$ space). The two submatrix maximum queries are therefore over the full Monge matrices $[i_0, i_1] \times [j_0, j' - 1]$ and $[i_0, i' - 1] \times [j', j_1]$. Hence, it is enough to focus on answering dominance maximum queries.

To answer a dominance maximum query (i.e., to find the maximum $M[i, j]$ over all $i \geq i'$ and $j \geq j'$) we use the partition of $M$ into full Monge matrices (fragments). Every such fragment is either fully inside the query range, fully outside of the query range, or intersected by the query range boundary.

**Fragments inside the query range.** A fragment $[r_0, r_1] \times [c_0, c_1]$ is fully inside the query range iff $r_0 \geq i'$ and $c_0 \geq j'$. This observation allows us to reduce computing the maximum over all matrices fully inside the query to the problem defined in Lemma 7. The reduction is simply that for every fragment $[r_0, r_1] \times [c_0, c_1]$ we create a point $(r_0, c_0)$ and set its weight to be the maximum inside the fragment. As a result, we create at most $O(n)$ points on the $n \times n$ grid. Using Theorem 2 on $\widetilde{M}$ to create every point separately takes total $O(n \log\log n)$ in the preprocessing time, so in $O(n \log\log n)$ time we can construct a structure of size $O(n)$ answering queries in $O(\log\log n)$ time.

**Fragments intersected by the query range.** We are left only with finding the maximum over all fragments intersected by the boundary of our dominance maximum query. We partition these fragments into three groups. The first consists of the single fragment containing $M[i', j']$. The maximum there can be found with a submatrix maximum query on $\widetilde{M}$ in $O(\log\log n)$ time. All other fragments intersected by the boundary are either intersected by the horizontal line $y = i'$ or the vertical line $x = j'$, but not both. We show how to find the maximum over all matrices intersected by the horizontal line $y = i'$ and fully to the right of the vertical line $x = j'$ (the other case is symmetric).

By the properties of our decomposition scheme, there are at most $\log n$ fragments intersected by any horizontal line, and they can be arranged in the natural left-to-right order. For every possible horizontal line, we store these at most $\log n$ fragments in an array. For every fragment we store the coordinates of its corresponding submatrix of $M$ and the maximum in all of its entries below the horizontal line. The array is additionally equipped with the maximum over all maxima in each one of its suffixes. Such preprocessed data allows us to find the maximum over all fragments intersected by a horizontal line $y = i'$ and fully on the right of a vertical line $x = j'$ in $O(\log\log n)$ time: First, we binary search over the array stored for $y = i'$ to locate the leftmost fragment completely on the right of $x = j'$. Then we return the stored corresponding maximum. Notice that the binary search also allow us to locate the fragment containing $M[i', j']$. Consequently, the whole query time is $O(\log\log n)$ using $O(n \log n)$ space for this part of the implementation. To guarantee $O(n \log n)$ preprocessing time, we run the SMAWK algorithm on every fragment in the decomposition in total $O(n \log n)$ time. This gives us the maximum in every row of every fragment. This is then enough to construct all arrays in $O(n \log n)$ time. $\square$

We now proceed to improving Theorem 7 so that the structure needs just linear space. The main idea is to partition the $n \times n$ staircase matrix $M$ into cells of size $\log n \times \log n$ and then define a new smaller $(n/\log n) \times (n/\log n)$ staircase matrix $M'$ (whose entries correspond to cell-maxima in $M$) on which we apply Theorem 7. To implement this idea we need a number of additional auxiliary data structures, which take $O(n)$ space in total. We start with an auxiliary lemma, which will be used to provide constant-time access to entries of $M'$.

**Lemma 8.** *Given an $n \times n$ Monge matrix $M$ partitioned into $\log n \times \log n$ cells, a data structure of size $O(n)$ can be constructed in $O(n \log n)$ time to find the maximum in a given cell in $O(1)$ time.*

*Proof.* We partition $M$ into $n/\log n$ horizontal slices, each consisting of $\log n$ rows (and all columns). Consider a single slice, which is a $\log n \times n$ Monge matrix. We store its breakpoints $c_1 < c_2 < \ldots < c_k$ (where $k \leq \log n$) in an atomic heap, consequently allowing predecessor queries in $O(1)$ time (this is exactly how the structure from Corollary 1 works). Additionally, similarly to Lemma 2, for every $i \geq 2$ we precompute the value of

$$m_i = \max_{j \in [c_{i-1}, c_i)} M[r(c_{i-1}), j]$$

and augment these values with a (one dimensional) range maximum data structure. Here, $r(c)$ denotes the row containing the maximum element in the $c$-th column of the slice in question. Using two predecessor queries and one range maximum query, the problem of finding the maximum in a given cell (which is fully contained in a single horizontal slice) reduces in $O(1)$ time to finding the maximum in at most two rows. The total space is $O(n/\log n \cdot \log n) = O(n)$ and the bottleneck in the preprocessing is computing the breakpoints for all slices. The breakpoints of a single slice can be computed in $O(\log^2 n)$ by adding one row at a time, as done in the proof of Lemma 1. In total, this takes $O(n/\log n \cdot \log^2 n) = O(n \log n)$ total time.

We repeat the above reasoning on the transpose of $M$. As a result, we either already know the maximum element, or we have isolated at most two rows and at most two columns, such that the maximum lies in one of these rows and one of these columns. This gives us at most four candidates for the maximum, which can be retrieved and compared naively. ◻

We are now ready to present our linear-space improvement to Theorem 7.

**Theorem 8.** *Given an $n \times n$ staircase 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.* As in the proof of Theorem 7, we can assume that the defined entries in row $i$ start in the first column and end in column $t_i$, and that $t_1 \geq t_2 \geq \ldots \geq t_n$.

We partition $M$ into cells of size $\log n \times \log n$ and then define a smaller $(n/\log n) \times (n/\log n)$ staircase matrix $M'$. Notice that, unlike Lemma 8, $M$ is a staircase Monge matrix (and not a full Monge matrix). This means that there are three types of cells in $M$: fully defined, partially defined, and fully undefined. An entry of $M'$ is defined iff its corresponding cell in $M$ is fully defined. In this case the entry is equal to the maximum in the corresponding cell. The undefined entries of $M'$ are the ones corresponding to either partially defined or fully undefined cells of $M$. We appropriately (and implicitly) fill these entries using Lemma 5 to turn $M$' into a full Monge matrix $\widetilde{M}'$, on which we apply Lemma 8. This gives us constant-time access to the entries of $M'$, so finally we can apply Theorem 7 to preprocess it in $O(n)$ space and $O(n \log n)$ time to answer submatrix maximum queries in $O(\log \log n)$ time.

Regarding partially defined cells, we observe that there are at most $2n/\log n$ of them. Furthermore, they can be arranged in a linear order, so that if the part of $M$ corresponding to the $i$-th partially defined cell is $[r_i, r'_i] \times [c_i, c'_i]$, then for all $i$ either $[r_i, r'_i] = [r_{i+1}, r'_{i+1}]$ and $c'_i + 1 = c_{i+1}$ or $r_i = r'_{i+1} + 1$ and $[c_i, c'_i] = [c_{i+1}, c'_{i+1}]$ (to be more precise, we might need to declare some fully defined cells partially defined to guarantee this property). We create a predecessor structure storing all $r_i$s and a separate predecessor structure storing all $c_i$s. We also compute the maximum in every partially defined cell and store them in an array (arranged in the aforementioned linear order) augmented with a (one dimensional) range maximum structure. Computing the maximum in all partially defined cells is done in $O(n/\log n \cdot \log n \cdot \alpha(\log n)) = O(n \cdot \alpha(\log n))$ time using [19].

By the same reasoning given in the proof of Theorem 7, it is enough to implement dominance maximum queries on $M$. A dominance maximum query can be decomposed into (i) a dominance maximum query on $\widetilde{M}'$, which can be answered in $O(\log \log n)$ time, (ii) finding the maximum inside all partially defined cells fully within the query range, and (iii) finding the maximum inside partially defined cells intersected by the boundaries of the query range. All partially defined cells fully within the query range create a contiguous interval in the linear order. The range can be determined in $O(\log \log n)$ using the predecessor structures storing all $r_i$s and $c_i$s, and then the maximum can be found in $O(1)$ time with a (one dimensional) range maximum query. It remains to calculate the maximum inside partially defined cells intersected by the boundaries of the query range. We will describe how to process all partially defined cells intersected by the horizontal boundary. Handling the vertical boundary is symmetric.

Let the dominance maximum query be specified by $(i', j')$. We want to compute the maximum inside the query range and belonging to a partially defined cell intersected by the horizontal line $y = i'$. All such cells create a contiguous interval in the linear order, which can be determined with two predecessor queries in $O(\log \log n)$ time. In the same complexity, we can find the leftmost such cell $u$ which is not fully on the left of the vertical line $x = j'$. We decompose the original query into a dominance maximum query inside $u$, and the remaining part. The remaining part starts at a left boundary of a partially defined cell and consists of the entries at or below $y = i'$ in all partially defined cells to the right of $u$. Consequently, the answer can be preprocessed for every point on a left boundary of a partially defined cell using $O(n/\log n \cdot \log n) = O(n)$ space and $O(n/\log n \cdot \log n \cdot \alpha(\log n)) = O(n \cdot \alpha(\log n))$ time using [19]. The bottleneck in the preprocessing is computing the maximum in every row of every partially defined cell.

It remains to describe how to handle the dominance query in $u$. In other words, after constructing in $O(n \log n)$ time an $O(n)$ size structure, we have, in $O(\log \log n)$ time, reduced an arbitrary dominance maximum query into a dominance maximum query inside a single partially defined cell. This cell is a smaller $\log n \times \log n$ staircase matrix, and furthermore there are at most $2n/\log n$ such cells. By recursing on each of these smaller staircase matrices separately, we construct in additional $O(n/\log n \cdot \log n \log \log n) = O(n \log \log n)$ time an $O(n/\log n \cdot \log n) = O(n)$ size structure, which reduces the original dominance query, in additional $O(\log \log \log n)$ time, into a dominance maximum query inside one of $O(n/\log n \cdot \log n/\log \log n) = O(n/\log \log n)$ tiny $\log \log n \times \log \log n$ staircase matrix (each of them being a submatrix of the original $M$). By recursing again on every tiny staircase matrix separately, we construct in additional $O(n \log \log \log n)$ time an $O(n)$ size structure, which reduces the original arbitrary dominance query in additional $O(\log \log \log \log n)$ time into a dominance maximum query inside an $(\log \log \log n) \times (\log \log \log n)$ submatrix of $M$. Such dominance maximum query can be answered naively resulting in $O(\log \log n + (\log \log \log n)^2) = O(\log \log n)$ total query time. ◻

We are now ready to prove the main theorem of this section, which is that using Theorem 8 we can actually implement submatrix maximum queries on arbitrary (and not just staircase) partial Monge matrices. The idea is to partition the partial Monge matrix into staircase matrices, so that each row and each column belong to $O(1)$ staircase matrices. Such partitioning was used in [1,17] . We build the data structure of Theorem 8 on each staircase matrix in the decomposition, and build an additional data structure for queries spanning more than one staircase matrix.

**Theorem 9.** *Given an $n \times n$ partial 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 $M$ into staircase matrices as done in the proof of Lemma 6 (depicted in Figure 3). Recall that the partition divides the rows of $M$ into three ranges. The first range contributes two staircase matrices, the second range creates a double staircase matrix, which is further broken into multiple staircase matrices, and the third range contributes two staircase matrices. It is easy to verify that, in the resulting decomposition, each row is covered by at most two staircase matrices, and each column is covered by at most three staircase matrices. Additionally, the staircase matrices contributed by the second range can be partitioned into two *collections*, such that any two matrices in the same collection are row-disjoint and column-disjoint.

The data structure consists of the following components. We apply Theorem 8 on every staircase matrix in our partition. We also store additional data for both collections. By left-right symmetry, we can assume that the ranges of rows and columns of the matrices in the collection are $[r_1, r_2), [r_2, r_3), \ldots$ and $[c_1, c_2), [c_2, c_3), \ldots$, respectively. We create a predecessor structure storing all $r_i$'s and a separate predecessor structure storing all $c_i$'s. We also compute and store the maximum inside every staircase matrix in the collection (this is done in total $O(n \cdot \alpha(n))$ time using the algorithm of Klawe and Kleitman [19]), and augment these maxima with a (one dimensional) range maximum structure.

Now consider a submatrix maximum query $[i_0, i_1] \times [j_0, j_1]$. We first query the $O(1)$ structures built for the staircase matrices corresponding to the first and the third range. Next, we consider each of the two collections separately. To find the maximum $M[i, j]$ over all $i \in [i_0, i_1]$ and $j \in [j_0, j_1]$, we use the predecessor structures to determine in $O(\log \log n)$ the following values (without loss of generality, they all exist):

1. $i'_0$ such that $i_0 \in [r_{i'_0}, r_{i'_0+1})$,
2. $i'_1$ such that $i_1 \in [r_{i'_1}, r_{i'_1+1})$,
3. $j'_0$ such that $j_0 \in [c_{j'_0}, c_{j'_0+1})$,
4. $j'_1$ such that $j_1 \in [c_{j'_1}, c_{j'_1+1})$.

We then query the structures built for the $(i'_0)$-th, $(i'_1)$-th, $(j'_0)$-th, and $(j'_1)$-th staircase matrix in the collection. Now either we have already found the maximum, or it belongs to one of the staircase matrices fully contained in the query range. Consequently, the maximum can be found in $O(1)$ time with a single (one dimensional) range maximum query. $\square$

## References


1. A. Aggarwal and M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discrete Appl. Math.*, 27:3–23, 1990.
2. 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.
3. S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *41st FOCS*, pages 198–207, 2000.
4. A. Amir, J. Fischer, and M. Lewenstein. Two-dimensional range minimum queries. In *18th CPM*, pages 286–294, 2007.
5. 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.
6. 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.
7. R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Appl. Math.*, 70:95–161, 1996.
8. T. M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Trans. Algorithms*, 9(3):22:1–22:22, 2013.
9. T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *27th SOCG*, pages 354–363, 2011.
10. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17:427–462, 1988.
11. B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *5th SOCG*, pages 131–139, 1989.
12. E. D. Demaine, G. M. Landau, and O. Weimann. On Cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014.
13. 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.
14. 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.
15. H. Gabow, J. L. Bentley, and R.E Tarjan. Scaling and related techniques for geometry problems. In *16th STOC*, pages 135–143, 1984.
16. P. Gawrychowski, M. Lewenstein, and P. K. Nicholson. Weighted ancestors in suffix trees. In *22th ESA*, pages 455–466, 2014.
17. P. Gawrychowski, S. Mozes, and O. Weimann. Improved submatrix maximum queries in Monge matrices. In *41st ICALP*, pages 525–537, 2014.
18. 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.
19. 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.
20. T. Kopelowitz and M. Lewenstein. Dynamic weighted ancestors. In *18th SODA*, pages 565–574, 2007.
21. M. Lewenstein. Orthogonal range searching for text indexing. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066, pages 267–302. Springer, 2013.
22. Y. Nekrich. Orthogonal range searching in linear and almost-linear space. *Comput. Geom.*, 42(4):342–351, 2009.
23. M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *38th STOC*, pages 232–240, 2006.
24. H. Yuan and M. J. Atallah. Data structures for range minimum queries in multidimensional arrays. In *21st SODA*, pages 150–160, 2010.