(Near-)Linear-Time Randomized Algorithms for Row Minima in Monge Partial Matrices and Related Problems

Timothy M. Chan*

Abstract

We revisit classical problems about searching in totally monotone and Monge matrices, which have many applications in computational geometry and other areas. We present a number of new results, including the following:

- A randomized algorithm that finds the row minima in an $n \times n$ Monge staircase matrix in $O(n)$ expected time; this improves a longstanding $O(n\alpha(n))$ bound by Klawe and Kleitman (1990) for totally monotone staircase matrices.
- A randomized algorithm that reports the K smallest elements (in an arbitrary order) in an $n \times n$ Monge (complete or staircase) matrix in $O(n + K)$ expected time; this improves and extends a previous $O(n+K \log n)$ algorithm by Kravets and Park [SODA'90].
- A randomized algorithm that reports the K smallest elements (in an arbitrary order) in an $n \times n$ totally monotone (complete) matrix in $O(n+K \log^* n)$ expected time.
- A randomized algorithm that reports the k_i smallest elements in the i-th row, for every i, in an $n \times n$ totally monotone (complete) matrix in $O((n+K) \log^* n)$ expected time, where $K = \sum_i k_i$.
- A randomized algorithm that finds the row minima in an $n \times n$ totally monotone "v-matrix" in $O(n\alpha(n) \log^* n \log\log n)$ expected time; this answers an open question by Klawe [SODA'90]. The $\log^* n$ factor can be removed in the Monge case.

1 Introduction

Totally monotone and Monge matrices. Totally monotone matrices and Monge matrices arise in many subareas of algorithms, including computational geometry, dynamic programming speedups, shortest paths in planar graphs, and combinatorial optimization. An $m \times n$ matrix[1] A is *concave totally monotone* iff for every $i < i'$ and $j < j'$,

$$A[i, j] \geq A[i, j'] \implies A[i', j] \geq A[i', j'].$$

The matrix A is *concave Monge* iff for every $i < i'$ and $j < j'$,

$$A[i, j] + A[i', j'] \leq A[i, j'] + A[i', j].$$

Clearly, the Monge property (also known as the "quadrangle inequality") implies total monotonicity. Although the converse is not necessarily true, in most applications, total monotonicity is actually proved by establishing the Monge property.

Convex total monotonicity and the convex Monge property can be defined similarly but with $\geq$ and $\leq$ reversed. (Note that a convex totally monotone or Monge matrix can be turned into a concave totally monotone or Monge matrix by reversing the order of the columns.)

By default, a matrix will refer to a complete matrix where all entries are filled. In the case of a *partial* matrix where some entries may be unfilled, the definitions are similar: the conditions should hold for all $i < i'$ and $j < j'$ whenever all four elements $A[i, j], A[i', j'], A[i, j'], A[i', j]$ are filled.

The seminal work on the topic is the SMAWK algorithm by Aggarwal, Klawe, Moran, Shor, and Wilber [4] (named after the initials of the authors), which can compute the minima of all the rows of an $m \times n$ totally monotone (complete) matrix A in linear $O(m+n)$ time. In fact, the time bound is $O(n(1 + \log \lceil \frac{m}{n} \rceil))$ if a compact output representation is allowed; this bound is optimal. The input A may be given implicitly—we only assume that any matrix entry can be evaluated on demand in constant time. (Even more restrictively, the only primitive operation required is comparing two elements in a common row.) The SMAWK algorithm has numerous applications, which we will refrain from exhaustively listing (see various surveys, e.g., [9, 21]).

Row minima in partial matrices. An important line of work have followed subsequently, aiming to develop row minima algorithms for more general types of totally monotone *partial* matrices. Aggarwal and Klawe [3] began studying the case of staircase matrices: A *falling staircase matrix* is a partial matrix such that the filled entries in each row form a suffix of the row, and the filled entries in each column form a prefix of the column. For example, this include the case of an upper triangular matrix, with entries $A[i, j]$ filled for $i \leq j$. Similarly, a *rising staircase matrix* is a partial matrix such that

*Department of Computer Science, University of Illinois at Urbana-Champaign (tmc@illinois.edu). This research has been supported in part by NSF Grant CCF-1814026.

[1]Note that some papers on this topic switch m and n for rectangular matrices.

the filled entries in each row form a prefix of the row, and the filled entries in each column form a prefix of the column. (See Figure 1(a,b).) Row minima in concave totally monotone, falling staircase matrices (or convex totally monotone, rising staircase matrices) can be found in linear time directly by SMAWK, since the unfilled entries may be filled with large numbers while preserving total monotonicity. However, computing row minima in concave totally monotone, rising staircase matrices (or convex totally monotone, falling staircase matrices) is more challenging, and arises in a number of applications. Aggarwal and Klawe gave an $O((m+n)\log\log m)$-time algorithm for this problem, which was later improved to an $O(n\alpha(m)+m)$-time algorithm by Klawe and Kleitman [27] in 1990. Their result is notable for being one of a few algorithms in the literature that has an original recursion with inverse Ackermann complexity (not directly due to the use of union-find data structures or the combinatorics of Davenport-Schinzel sequences).

In SODA'90, Klawe [26] introduced more general classes of partial matrices: A *skyline matrix* is a partial matrix such that all the defined entries in each column form a suffix of the column. A *v-matrix* is a partial matrix such that all the defined entries in each column occur contiguously (but not necessarily as a prefix or suffix). Similarly, an *h-matrix* is a partial matrix such that all undefined entries in each row occur contiguously. (See Figure 1.) Klawe proved an $\Omega(n\alpha(n))$ lower bound for the number of evaluations of matrix entries to compute row minima for totally monotone v-matrices and h-matrices for $m = \Theta(n)$. She then described row minima algorithms for totally monotone skyline matrices requiring $O(n\alpha(m)\log\log m+m)$ time and $O(n\alpha(m)+m)$ comparisons, or alternatively $O(n\log\log m+m)$ time and comparisons. In the introduction of her paper, she raised the question of finding any $o(n\log m+m)$-time algorithm more generally for totally monotone v-matrices (or h-matrices).

In the intervening 30 years, no further improvements have been reported on these problems since the papers by Klawe and Kleitman and by Klawe.

New results. In this paper, we give the first linear-time algorithm for computing row minima in an $m \times n$ concave Monge, rising staircase matrix (or convex Monge, falling staircase matrix). The algorithm is randomized, Las-Vegas style. This improves Klawe and Kleitman's longstanding $O(n\alpha(m)+m)$ bound. More precisely, the expected time bound of our algorithm (assuming a compact output representation) is $O(n(1+\log\lceil\frac{m}{n}\rceil))$, matching that of SMAWK. From the practical perspective, the improvement of an α factor may seem slight, but from the theoretical perspective, the result interestingly demonstrates that appearance of inverse Ackermann is unnecessary for this and related problems, and cleans up some of the time bounds reported in the literature.

To be fair, we should reiterate that Klawe and Kleitman's algorithm works more generally for totally monotone staircase matrices, not just Monge staircase matrices. But as mentioned, in most (if not all) known applications involving total monotonicity, the Monge property is satisfied. Indeed, in Appendix A, we list a number of applications of our new algorithm in computational geometry and dynamic programming speedups. A more limiting disadvantage of our algorithm is that it does not generalize to the "online" setting, unlike Klawe and Kleitman's (see their paper for the definition), which is required in some applications to dynamic programming speedups.

We also answer Klawe's open question [26] about v-matrices, by giving the first nontrivial algorithm for finding the row minima in an $m \times n$ totally monotone v-matrix, with expected running time $O(n\alpha(m)\log^* m\log\log m + m)$ (or more precisely, $O(n\alpha(m)\log^* m\log\log m + n\alpha(m)\log\lceil\frac{m}{n}\rceil)$). The $\log^*$ factor can be removed in the Monge case.

($\leq K$)-selection and other related problems. In addition to row minima, we also consider a number of related, basic problems about searching in a totally monotone or Monge matrix:

- ($\leq K$)-*selection*: report the K smallest elements (in an arbitrary order).
- ($\leq t$)-*reporting*: report all elements that are at most t (in an arbitrary order).
- *row* ($\leq k_1,\ldots,k_m$)-*selection*: for each $i = 1,\ldots,m$, report the k_i smallest elements (in an arbitrary order) in the i-th row.
- *row* ($\leq t_1,\ldots,t_m$)-*reporting*: for each $i = 1,\ldots,m$, report all elements that are at most t_i (in an arbitrary order).

For the last three problems, we use K to denote the total output size (e.g., for the third problem, $K = \sum_i k_i$). For example, the standard row minima problem corresponds to row ($\leq 1,\ldots,1$)-selection. Note that row ($\leq t_1,\ldots,t_m$)-reporting is equivalent to ($\leq t$)-reporting, by translating the values in each row (i.e., changing $A[i,j]$ to $A[i,j]-t_i$); this operation preserves total monotonicity and the Monge property. The ($\leq K$)-selection and the ($\leq t$)-reporting problem have similar complexity, since there is a straightforward reduction

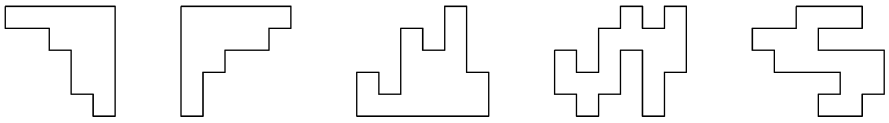

Figure 1: (a) Falling staircase matrix; (b) rising staircase matrix; (c) skyline matrix; (d) v-matrix; (e) h-matrix.

of the latter to the former, and a simple randomized reduction of the former to the latter (see Section 5.1). In a companion paper [14], we explore more related problems such as selection of the K-th smallest element.

In SODA'90, Kravets and Park [29] studied the row $(\leq k, \ldots, k)$-selection problem for a totally monotone (complete) matrix, and obtained an algorithm running in $O((m+n)k)$ time. Note that although mk may be a trivial lower bound if we require an explicit output representation of size $\Omega(mk)$, the nk term isn't obviously a lower bound.

Kravets and Park further obtained an $O(m+n+K\log\frac{mn}{K})$-time algorithm for the $(\leq K)$-selection problem for a totally monotone (complete) matrix whose transpose is also totally monotone—in particular, a Monge matrix satisfies this extra condition. (One natural application is in computing the K farthest pairs for a planar point set in convex position.) They left open the question of whether a comparable result exists for arbitrary totally monotone matrices.

More new results. We give the first linear ($O(m+n+K)$) time algorithm for the $(\leq t)$-reporting and the $(\leq K)$-selection problem for an $m \times n$ totally monotone (complete) matrix whose transpose is also totally monotone (and thus for a Monge matrix). The algorithm is randomized. This improves Kravets and Park's result by a logarithmic factor. The same result holds for Monge staircase matrices (for which no previous results were known).

We also obtain an almost linear-time randomized algorithm for $(\leq t)$-reporting and $(\leq K)$-selection for an arbitrary $m \times n$ totally monotone (complete) matrix. The running time is $O(m+n+K\log^* n)$ (or alternatively, $O(m+n\log^{(c)} n+K)$ for an arbitrarily large constant c). This answers the above-mentioned open question by Kravets and Park.

We similarly obtain an almost linear-time randomized algorithm for the row $(\leq k_1, \ldots, k_m)$-selection problem for an arbitrary $m \times n$ totally monotone (complete) matrix. The running time is $O((n+K)\log^* n)$ (or alternatively, $O((m+n)\log^{(c)} n+K)$). Notice that even in the special case of row $(\leq k, \ldots, k)$-selection, our time bound, approaching $O(mk+n)$, improves Kravets and Park's previous $O((m+n)k)$ bound, ignoring iterated logarithmic factors.

Techniques, and connection with pseudo-lines. To obtain our solutions, instead of working with matrices, we take a geometric perspective (ironically, the original SMAWK paper took the opposite philosophy, of reducing geometric problems to matrix searching).

We view the columns of a totally monotone matrix as pseudo-lines in the plane: A set of n curves in the plane forms a *pseudo-line* family if each curve is x-monotone (i.e., each vertical line intersects the curve exactly once) and each pair of curves intersects at most once. From an $m \times n$ totally monotone matrix A, for each $j = 1, \ldots, n$, we can form a polygonal curve γ_j passing through the points $(-\infty, j), (1, A[1,j]), (2, A[2,j]), \ldots, (m, A[m,j]), (\infty, -j)$. Concave total monotonicity implies that these curves are indeed pseudo-lines (ignoring degeneracies), as shown in Figure 2(a). Consequently:

- The row minima problem corresponds to evaluating the lower envelope of these n pseudo-lines at the x-coordinates $1, \ldots, m$ (in other words, computing a "discretized" lower envelope). See Figure 2(b).

- The row $(\leq t_1, \ldots, t_m)$-reporting problem corresponds to reporting all (i,j) index pairs such that the point (i, t_i) is above the pseudo-line γ_j. This is equivalent to answering n offline "pseudo-halfplane range reporting queries" on a set of m points.

Certain types of partial matrices also have geometric interpretations: v-matrices correspond to pseudo-line segments, and skyline matrices correspond to pseudo-rays (pseudo-line segments that are unbounded in one direction).

The above viewpoint allows us to leverage the rich body of techniques from computational geometry, concerning lower envelopes of lines or line segments, halfplane range searching, and randomized geometric divide-and-conquer. (Even if some of the ideas could be translated back in matrix terms, the geometric perspective is helpful to avoid "reinventing the wheel".) Certain techniques for lines or line segments may be adapted for pseudo-lines or pseudo-line segments without much difficulty. However, not all existing techniques

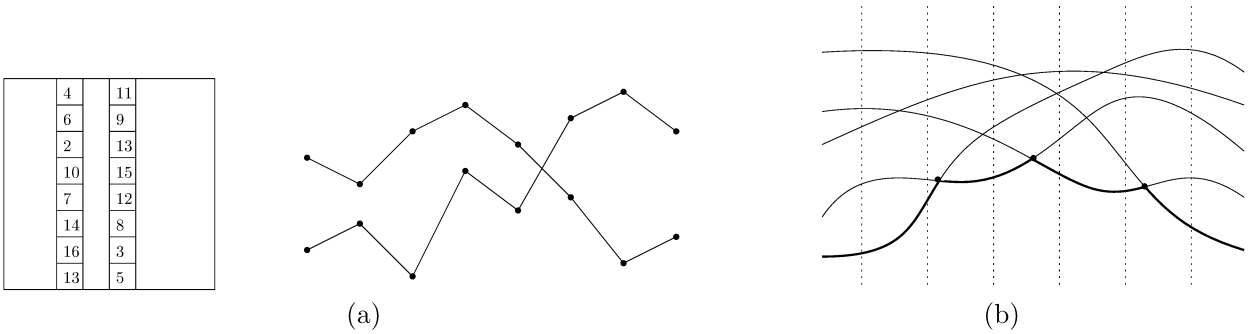

Figure 2: (a) Columns in a totally monotone matrix map to pseudo-lines. (b) A lower envelope of pseudo-lines.

can be generalized. The following key differences make the problems here more challenging:

- First, we are aiming for linear or almost linear time bounds. Traditional algorithms for computing lower envelopes of line segments or halfspace range reporting require at least $n \log n$ time. To beat $n \log n$ in a comparison model, we need to exploit the fact that the input x-coordinates ($\{1, \ldots, m\}$) are pre-sorted (trivially), and the input pseudo-lines are also pre-sorted (by y-values at $x = -\infty$ or $x = \infty$).

- Primitive operations are limited. Although we can evaluate the y-value of a pseudo-line at a given x-coordinate in constant time, computing the intersection of two pseudo-lines now requires binary search, taking $O(\log m)$ time.[2]

- Certain operations for points and lines do not easily generalize to points and pseudo-lines. For example, although it is possible to determine the orientation of three input pseudo-lines in $O(\log m)$ time, we cannot determine the orientation of three input points (in particular, we cannot compute the convex hull efficiently), since assigning orientation to a point triple require a global examination of the relationship of the points with all the pseudo-lines. And we cannot use standard point-line duality and work in dual space.[3]

From the geometric perspective, one can see that Klawe's $\Omega(n\alpha(n))$ lower bound for v-matrices [26] is not really evidence to suggest optimality of Klawe and Kleitman's result for staircase matrices, since v-matrices correspond to pseudo-line segments, whose lower envelopes have $\Theta(n\alpha(n))$ worst-case combinatorial complexity [38], but staircase (and skyline) matrices correspond to pseudo-rays, whose lower envelopes have linear combinatorial complexity.

The connection between totally monotone matrices and pseudo-lines is certainly not new. For example, see a recent paper by Kaplan et al. [25] (who used this pseudo-line perspective to obtain data structures for Monge matrices, though these were later improved [22]), or the paper by Klawe [26] (whose lower bounds were obtained by relating row minima in v-matrices to lower envelopes of line segments), or a work by Felzenszwalb and Huttenlocher [20] (who noted an application of SMAWK to solve a discrete lower envelope problem). On the other hand, the connection was overlooked by some researchers (for example, Millman et al. [32] studied the discrete lower envelope problem for pseudo-lines and proposed a new randomized linear-time algorithm that rederived the SMAWK result, without realizing it!).

As we have mentioned, point-line duality is no longer available, which limits the geometric techniques that we can use. New geometric ideas are thus needed to obtain our $O(m + n + K \log^* n)$-time ($\leq t$)-reporting algorithm for totally monotone matrices. In the Monge case, we observe that something analogous to duality is possible, simply by transposing the matrix! (The Monge property is preserved by transposition.) The ($\leq t$)-reporting problem is symmetric with respect to transposition, but the row minima problem is not. To obtain our linear-time results for Monge staircase matrices, it is essential that we solve both problems simultaneously, with the row minima algorithm invoking the ($\leq t$)-reporting algorithm, and vice versa. Our algorithms will use recursion in interesting, original ways.

[2] For example, this issue explains why the dualized Graham's scan [23], which can computing the lower envelope of n pre-sorted lines in linear time, cannot directly solve the row minima problem for totally monotone matrices; Graham's algorithm requires $O(n)$ intersection operations and would now take $O(n \log m)$ time. (However, a known randomized incremental algorithm for computing lower envelopes can be adapted—see Appendix B.)

[3] A duality transform for points and pseudo-lines actually exists, as shown by Agarwal and Sharir [2], but is too expensive to compute for our purposes.

2 ($\leq t$)-Reporting for Monge Matrices

We begin by studying the row $(\leq t_1, \ldots, t_m)$-reporting problem for an $m \times n$ (complete) matrix A in the case when both A and its transpose are totally monotone. In particular, a Monge matrix satisfies this property. As noted in the introduction, it suffices to consider $(\leq t)$-reporting problem. To enable recursive randomized algorithms, we allow the input to be random, in which case, we use m and n to denote the *expected* number of rows and columns in A respectively. We let K denote the *expected* number of output elements. Let $T(m, n, K)$ be the expected time needed to solve the problem. In case when we know that the number of columns is always upper-bounded by N, we let $T_N(m, n, K)$ be the expected time under these parameters.

In geometric terms, the problem reduces to the following:

> Given a set P of points of (expected) size m and a set L of pseudo-lines of (expected) size n in the plane, report all pairs $(p, \ell) \in P \times L$ with p above ℓ.

K is the (expected) number of output pairs. By a linear scan over the output pairs, we can easily report all pseudo-lines of L below p for each point $p \in P$, or equivalently, report all points of P above ℓ for each pseudo-line $\ell \in L$. This problem can thus be viewed as *offline pseudo-halfplane range reporting*.

The points of P are pre-sorted by x-coordinates, and the pseudo-lines of L are pre-sorted by pseudo-slope, where the *pseudo-slope* of a pseudo-line refers to its rank of its y-value at $x = \infty$ among all pseudo-lines of L. (In the $(\leq t)$-reporting problem, all points of P actually have the same y-coordinate t.) The only allowed primitive operation is evaluating the y-coordinate of a pseudo-line $\ell \in L$ at the x-coordinate of a point $p \in P$. For simplicity, we assume no degeneracies, e.g., no two pseudo-lines have the same y-coordinate at an x-coordinate of P (this may be avoided by perturbation).

Our algorithm builds on a randomized divide-and-conquer approach, which is well known in computational geometry since the work of Clarkson and Shor [17, 34], and has been used before in halfspace range reporting [11]. We incorporate some extra ideas, leading to a new recurrence with linear complexity.

Clarkson–Shor-style divide-and-conquer. Let s and b be parameters to be set later. Take a random sample R of L of size $\frac{n}{s}$. Consider the lower envelope $\mathrm{LE}(R)$, which has $O(\frac{n}{s})$ vertices, and take its *vertical decomposition* $\mathrm{VD}(R)$, i.e., a division of the region below $\mathrm{LE}(R)$ into $O(\frac{n}{s})$ cells formed by drawing downward rays from the vertices of $\mathrm{LE}(R)$. We use a discretized version of the envelope, with x-coordinates of vertices "rounded" to the x-coordinates of P. (See Figure 3.) By SMAWK,[4] the discretized lower envelope can be found in $O(m + \frac{n}{s})$ time. We can compute $P \cap \Delta$ for all cells $\Delta \in \mathrm{VD}(R)$ by a linear scan over the x-pre-sorted point set P and $\mathrm{LE}(R)$; each subset can be kept sorted by x.

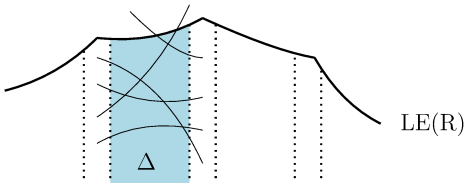

Figure 3: A cell Δ (after rounding) in the vertical decomposition of $\mathrm{LE}(R)$, and its conflict list L_Δ.

For each cell $\Delta \in \mathrm{VD}(R)$, we compute its *conflict list* L_Δ, i.e., the list of all pseudo-lines of L intersecting Δ. To this end, for each pseudo-line $\ell \in L$, we first find its predecessor and successor with respect to pseudo-slope, among the pseudo-lines appearing on $\mathrm{LE}(R)$; this can be done for all ℓ by one linear scan in $O(n)$ time, since the pseudo-lines are pre-sorted by pseudo-slope. For each $\ell \in L$, we can start at the vertex v of $\mathrm{LE}(R)$ that is defined by its predecessor and successor, and then do a linear search in both directions from v, to find all cells $\Delta \in \mathrm{VD}(R)$ intersected by ℓ. (Note that in these steps, we do not need to compute intersections of pseudo-lines; we just need to test whether a pseudo-line is above a point incident on another pseudo-line.) The total time of these linear searches is proportional to the total number of conflicts $O(\sum_{\Delta \in \mathrm{VD}(R)} |L_\Delta|)$. By a standard analysis of Clarkson and Shor [17, 34], $\sum_{\Delta \in \mathrm{VD}(R)} |L_\Delta|$ has expected value $O((\frac{n}{s})s) = O(n)$. By a linear scan over the pre-sorted set L, all the L_Δ's can be kept sorted by pseudo-slope.

For each $\Delta \in \mathrm{VD}(R)$ with $|L_\Delta| \leq bs$, we recursively solve the subproblem for $P \cap \Delta$ and L_Δ. Let P' be the set of all remaining points of P, i.e., those that are above $\mathrm{LE}(R)$ or lie in cells Δ with $|L_\Delta| > bs$. We solve the problem for P' and L by another recursive call.

The number of points $p \in P$ that are above more than b pseudo-lines is at most $\frac{K}{b}$. For each point $p \in P$ that are above at most b pseudo-lines, the probability that p is above $\mathrm{LE}(R)$ is at most $\frac{b}{s}$. Furthermore, by Clarkson and Shor's analysis [17, 34] (see also [11]), the expected value of $|L_{\Delta(p)}|$ for the cell $\Delta(p) \in \mathrm{VD}(R)$ containing p is $O(s)$ for any fixed p. Thus, by Markov's inequality, the probability that $|L_{\Delta(p)}| > bs$ is $O(\frac{1}{b})$ for any fixed p. Hence, the expected size of P' is at most

[4] It is possible to avoid SMAWK and get a more self-contained algorithm—in fact, we will do just that in Section 3.

$O(\frac{K}{b} + \frac{mb}{s} + \frac{m}{b})$.

Choose $s = b^2$. The expected size of P' is then $O(\frac{m+K}{b})$. We then obtain the following recurrence:

$$T(m,n,K) \leq \sum_i T_{O(b^3)}(m_i, n_i, K_i) + T(O(\tfrac{m+K}{b}), n, K') + O(m+n), \tag{2.1}$$

for some m_i's, K_i's, and K' with $\sum_i n_i = O(n)$, $\sum_i m_i \leq m$, and $\sum_i K_i + K' \leq K$.

The above recurrence by itself does not yield good results (mainly because n does not decrease in the $T(O(\frac{m+K}{b}), n, K')$ term), but we will fix this by combining the recurrence with one extra ingredient...

Symmetry. Notice that so far, we have only used the total monotonicity of the input matrix A. Since both A and the transpose of A are assumed to be totally monotone, we can actually transpose the input and obtain

$$T(m,n,K) = T(n,m,K),$$

as the ($\leq t$)-reporting problem is unchanged after transposition. This is the extra ingredient we need.

Putting it all together. We will not use recursion for the subproblems associated with the cells $\Delta \in \mathrm{VD}(R)$, but instead use any naive algorithm with running time $O((m+n)\log^{O(1)} N + K)$ (there are many options, one of which is to use the $O(m+n\log^{(c)} N + K)$-time algorithm from Section 4.1, e.g., with $c = 1$). This way, we can replace $T_{O(b^3)}(m_i, n_i, K_i)$ with $O((m_i+n_i)\log^{O(1)} b + K_i)$. From (2.1), letting $K'' = K - K'$, we get:

$$T(m,n,K) \leq O((m+n)\log^{O(1)} b + K'') + T(O(\tfrac{K}{b}), n, K - K''). \tag{2.2}$$

Note that we have replaced $O(\frac{m+K}{b})$ with $O(\frac{K}{b})$. This is because we can initially reduce m to $O(K)$ by running SMAWK in $O(m+n)$ time, and removing points of P below $\mathrm{LE}(L)$ which do not contribute to the output. (This extra step is not essential, but simplifies calculations.)

Using symmetry to rewrite $T(O(\frac{K}{b}), n, K - K'')$ as $T(n, O(\frac{K}{b}), K - K'')$ and applying (2.2) a second time, we get:

$$T(m,n,K) \leq O\Big((m+n+\tfrac{K}{b})\log^{O(1)} b + K'''\Big) + T(O(\tfrac{K}{b}), O(\tfrac{K}{b}), K - K''')$$

for some $K''' \leq K$.

Now we expand the recurrence using an increasing sequence of parameters $b_1, b_2, \ldots$:

$$T(m,n,K) \leq O\Big((m+n+\tfrac{K}{b_1})\log^{O(1)} b_1 + K_1''' + (\tfrac{K}{b_1} + \tfrac{K}{b_2})\log^{O(1)} b_2 + K_2''' + (\tfrac{K}{b_2} + \tfrac{K}{b_3})\log^{O(1)} b_3 + K_3''' + \cdots\Big)$$

for some K_j''''s with $\sum_j K_j''' \leq K$.

Finally, choosing $b_j = 2^j$ (and noting that $\sum_j j^{O(1)}/2^j = O(1)$), we conclude that $T(m,n,K) = O(m+n+K)$.

THEOREM 2.1. *Given an $m \times n$ totally monotone matrix whose transpose is also totally monotone, we can report all K elements that are at most a given value t (in an arbitrary order) in $O(m+n+K)$ expected time.*

Remarks. This result appears new even in the case of lines (rather than pseudo-lines), yielding an $O(m+n+K)$-time randomized algorithm for 2D offline halfplane range reporting, assuming that the input points are pre-sorted by x and the lines pre-sorted by slope. Here, symmetry $T(m,n,K) = T(n,m,K)$ follows from standard point-line duality. In contrast, standard algorithms for 2D halfplane or 3D halfspace range reporting [11, 16] require $O((m+n)\log n + K)$ time and do not exploit input pre-sortedness.

The ideas behind our algorithm (the usage of Clarkson–Shor divide-and-conquer combined with symmetry or duality) has similarities with some known work on offline 3D dominance range reporting; for example, see [1, proof of Theorem 3] (the algorithm there only considered one round of bootstrapping with duality, since the target time bound was $O(n\log\log n + K)$, whereas our algorithm uses multiple rounds and is more interesting).

Derandomization appears very difficult (lower envelopes of random samples may be derandomized using deterministic construction of *shallow cuttings* [15], but this requires at least $\Omega(n\log n)$ time).

3 Generalization to Monge Staircase Matrices

We now extend our linear-time algorithm for ($\leq t$)-reporting to Monge staircase matrices. At the same time, we obtain a linear-time algorithm for row minima in Monge staircase matrices. (As mentioned in the introduction, the difficult cases are concave Monge, rising staircase matrices and convex Monge, falling staircase matrices.)

Let $T_{\mathrm{minima}}(m,n)$ be the expected time needed to solve the row minima problem for a Monge staircase matrix where m and n are the *expected* number of rows and columns respectively.

In geometric terms, the generalization to staircase matrices corresponds to the setting where the pseudo-lines are replaced with pseudo-line segments, all unbounded on one side, say, the left side. Furthermore, the x-coordinates of the segments' right endpoints are monotonically increasing or decreasing with their pseudo-slopes: we refer to such curves as *monotone pseudo-rays*.

For a set P of points of (expected) size m and a set L of monotone pseudo-rays of (expected) size n, the $(\leq t)$-reporting problem is to find all pairs $(p, \ell) \in P \times L$ with p above ℓ. For a set X of values of (expected) size m and a set L of monotone pseudo-rays of (expected) size n, the row minima problem is to evaluate the lower envelope of L at the x-coordinates of X. It is assumed that the x-coordinates of P or X are pre-sorted, and the pseudo-rays of L are pre-sorted by pseudo-slope. (For the row minima problem, a compact representation of the output is allowed: if the answers for multiple consecutive values in X are defined by the same pseudo-ray in L, they may be reported once.)

For bootstrapping purposes, we will need an $O((m+n)\log^{O(1)} N + K)$ algorithm for $(\leq t)$-reporting. There are many options, one of which is to use binary divide-and-conquer to reduce to the pseudo-line case: We compute the median x-coordinate x_m of the rays' endpoints, let P_1 (resp. P_2) be the points left (resp. right) of $x = x_m$ and L_1 (resp. L_2) be the pseudo-rays whose endpoints are left (resp. right) of $x = x_m$. We recursively solve the problem for P_1 and L_1, and for P_2 and L_2, and finally solve the problem for P_1 and L_2 by viewing the pseudo-rays of L_2 as pseudo-lines and invoking, say, the $O(m + n\log^{(c)} N + K)$ algorithm from Section 4.1, e.g., with $c = 1$. This yields a total time bound of $O((m + n\log^{(c)} N)\log N + K)$.

$(\leq t)$-reporting algorithm. We solve the $(\leq t)$-reporting problem for monotone pseudo-rays by adapting the Clarkson–Shor-style divide-and-conquer algorithm from Section 2. We point out the key differences:

The lower envelope of pseudo-rays (even without monotonicity) still has a linear number of vertices (which include both intersections and segment endpoints that are visible from below), since their lower envelope complexity is related to order-2 Davenport-Schinzel sequences [38]. Thus, $\mathrm{LE}(R)$ still has expected size $O(\frac{n}{s})$. However, we can no longer use SMAWK to compute $\mathrm{LE}(R)$; instead, the discrete lower envelope computation now requires $T_{\mathrm{minima}}(m, \frac{n}{s})$ time. The total conflict list size $\sum_{\Delta\in\mathrm{VD}(R)} |L_\Delta|$ is again bounded by $O(n)$ in expectation. To compute the conflict lists of all cells in $\mathrm{VD}(R)$, it suffices to compute the conflict lists of all vertices of $\mathrm{LE}(R)$, where the conflict list of a vertex v is defined as the list of all pseudo-rays of L below v. This is because a pseudo-ray ℓ intersects a cell Δ iff ℓ is below at least one of the two vertices of Δ or the right endpoint of ℓ is in Δ. (We can easily identify the endpoints inside every cell by one left-to-right linear scan.) Now, computing the conflict lists of the vertices of $\mathrm{LE}(R)$ corresponds to a row $(\leq t_1, \ldots, t_m)$-reporting problem. As noted in the introduction, row $(\leq t_1, \ldots, t_m)$-reporting reduces to $(\leq t)$-reporting, by translating the values in each row (namely, resetting $A[i, j]$ to $A[i, j] - t_i$)—this operation preserves the Monge property. So, the conflict list computation can be done by an extra recursive call! The recursion is for a subproblem with $O(\frac{n}{s})$ expected number of points and n pseudo-rays, with $O(n)$ expected output size. (Fortunately, this extra subproblem will not hurt the recurrence too much.)

With $s = b^2$, recurrence (2.1) is changed to the following:

$$\begin{aligned} T(m, n, K) \quad \leq \quad & T_{\mathrm{minima}}(m, \tfrac{n}{b^2}) + T(O(\tfrac{n}{b^2}), n, O(n)) + \\ & \sum_i T_{O(b^3)}(m_i, n_i, K_i) + \\ & T(O(\tfrac{m+K}{b}), n, K') + O(m + n), \end{aligned} \tag{3.3}$$

for some m_i's, K_i's, and K' with $\sum_i n_i = O(n)$, $\sum_i m_i \leq m$, and $\sum_i K_i + K' \leq K$.

In addition, we still have symmetry $T(m, n, K) = T(n, m, K)$, because the transpose of a Monge staircase matrix is still a Monge staircase matrix (after reversing the order of the rows and the order of the columns).

Row minima algorithm. We solve the row minima problem, i.e., the discrete lower envelope problem, similarly by Clarkson–Shor-style divide-and-conquer. Take a random sample R of L of size $\frac{n}{s}$. Recursively compute the (discrete) lower envelope $\mathrm{LE}(R)$ and consider its vertical decomposition $\mathrm{VD}(R)$. Compute the conflict lists of all cells $\Delta \in \mathrm{VD}(R)$. As before, this reduces to a reporting problem for an expected $O(\frac{n}{s})$ number of points and n pseudo-rays, with $O(n)$ expected output size. For each cell $\Delta \in \mathrm{VD}(R)$, let m_Δ be the number of x-values of X in the x-projection of Δ; note that $\sum_{\Delta\in\mathrm{VD}(R)} m_\Delta$ is equal to m (in expectation). For each cell $\Delta \in \mathrm{VD}(R)$, compute the lower envelope of L_Δ. For example, we can adapt a naive $O(|L_\Delta|^2)$-time algorithm for this purpose. However, in the discrete setting, the computation of the intersection of two pseudo-rays cannot be done in constant time, but requires $O(\log m_\Delta)$ time by binary search (we only need to know the position of the intersection relative to m_Δ x-values). So, the naive algorithm requires $O(|L_\Delta|^2 \log m_\Delta)$ time. By a standard analysis of Clarkson and Shor [17, 34], $\mathbb{E}\left[\sum_{\Delta\in\mathrm{VD}(R)} |L_\Delta|^4\right] = O((\frac{n}{s})s^4) = O(ns^3)$. By con-

cavity of the $\log^2$ function and Jensen's inequality, $\sum_{\Delta\in \mathrm{VD}(R)} \log^2 m_\Delta \le O(|R|(1+\log^2(\sum_\Delta m_\Delta/|R|)))$. By Jensen's inequality again, $\mathbb{E}\left[\sum_{\Delta\in \mathrm{VD}(R)} \log^2 m_\Delta\right] \le O(\frac{n}{s}(1+\log^2\lceil \frac{sm}{n}\rceil))$. Hence, by the Cauchy–Schwarz inequality,

$$\begin{aligned}
&\mathbb{E}\left[\textstyle\sum_\Delta |L_\Delta|^2 \log m_\Delta\right] \\
&\le \sqrt{\mathbb{E}\left[\textstyle\sum_\Delta |L_\Delta|^4\right]\cdot \mathbb{E}\left[\textstyle\sum_\Delta \log^2 m_\Delta\right]} \\
&= O(\sqrt{ns^3\cdot \tfrac{n}{s}(1+\log^2\lceil \tfrac{sm}{n}\rceil)}) \\
&= O(sn(1+\log\lceil \tfrac{sm}{n}\rceil)).
\end{aligned}$$

Choose s to be a sufficiently large constant. We then obtain:

$$\begin{aligned}
T_{\mathrm{minima}}(m,n) \le\ & T_{\mathrm{minima}}(m,\tfrac{n}{2}) + T(O(n),n,O(n)) \\
& + O(n(1+\log\lceil \tfrac{m}{n}\rceil)),
\end{aligned} \tag{3.4}$$

Putting it all together. Expanding (3.4) and noting that $\sum_j \frac{n}{2^j}(1+\log\left\lceil \frac{2^j m}{n}\right\rceil) = O(n(1+\log\lceil \frac{m}{n}\rceil))$, we get:

$$\begin{aligned}
T_{\mathrm{minima}}(m,n) \le\ & \sum_{j=1}^{\infty} T(O(\tfrac{n}{2^j}),\tfrac{n}{2^j},O(\tfrac{n}{2^j})) + \\
& O(n(1+\log\lceil \tfrac{m}{n}\rceil)).
\end{aligned} \tag{3.5}$$

In (3.3), we can replace $T_{O(b^3)}(m_i,n_i,K_i)$ with $O((m_i+n_i)\log^{O(1)} b + K_i)$ using an aforementioned naive algorithm. Thus,

$$\begin{aligned}
T(m,n,K) \le\ & T_{\mathrm{minima}}(m,O(\tfrac{n}{b^2})) + \\
& T(O(\tfrac{n}{b^2}),n,O(n)) + \\
& O((m+n)\log^{O(1)} b + K - K') + \\
& T(O(\tfrac{m+K}{b}),n,K').
\end{aligned} \tag{3.6}$$

Using symmetry to rewrite $T(O(\frac{n}{b^2}),n,O(n))$ as $T(n,O(\frac{n}{b^2}),O(n))$ and $T(O(\frac{m+K}{b}),n,K')$ as $T(n,O(\frac{m+K}{b}),K')$ and applying (3.6) to expand these two terms, and setting $m=n$, we get:

$$\begin{aligned}
T(n,n,K) \le\ & T_{\mathrm{minima}}(n,O(\tfrac{n}{b^2})) + \\
& O((n+\tfrac{K}{b})\log^{O(1)} b + K_0'') + \\
& \sum_{j=1}^{4} T(O(\tfrac{n+K}{b}),O(\tfrac{n+K}{b}),K_j'')
\end{aligned} \tag{3.7}$$

for some $K_0'',\ldots,K_4''$ with $K_0''+\cdots+K_4'' \le K+O(n)$.

Rewriting $T_{\mathrm{minima}}(n,O(\frac{n}{b^2}))$ using (3.5), we obtain:

$$\begin{aligned}
T(n,n,K) \le\ & O((n+\tfrac{K}{b})\log^{O(1)} b + K_0''') + \\
& \sum_j T(n_j',n_j',K_j''')
\end{aligned}$$

for some n_j''s and K_j''''s with $\sum_j n_j' = O(\sum_j \frac{n/b^2}{2^j} + \frac{n+K}{b}) = O(\frac{n+K}{b})$ and $K_0'''+\sum_j K_j''' \le K+O(\sum_j \frac{n/b^2}{2^j}+n) = K+O(n)$.

Now we expand the recurrence using an increasing sequence of parameters $b_1,b_2,\ldots$ (and note that the expression $c_0(\cdots c_0((c_0(n+K)/b_1+K)/b_2+\cdots+K)/b_j$ is upper-bounded by $O(\frac{n+K}{b_j})$, assuming $b_j \ge c$ for some sufficiently large constant c depending on c_0):

$$\begin{aligned}
T(n,n,K) \le\ & O\Big((n+\tfrac{K}{b_1})\log^{O(1)} b_1 + K_1'''' + \\
& (\tfrac{n+K}{b_1}+\tfrac{K}{b_2})\log^{O(1)} b_2 + K_2'''' + \\
& (\tfrac{n+K}{b_2}+\tfrac{K}{b_3})\log^{O(1)} b_3 + K_3'''' + \cdots\Big)
\end{aligned}$$

for some K_j'''''s with $\sum_j K_j'''' \le K+O(n+\frac{n+K}{b_1}+\frac{n+K}{b_2}+\cdots) = K+O(n)$.

Finally, choosing $b_j = c^j$ (and noting that $\sum_j j^{O(1)}/c^j = O(1)$), we conclude that $T(n,n,K) = O(n+K)$. This implies that $T(m,n,K) = O(m+n+K)$.

Substituting this back into (3.5), we can also conclude that $T_{\mathrm{minima}}(m,n) = O(n(1+\log\lceil \frac{m}{n}\rceil))$.

THEOREM 3.1. *Given an $m\times n$ Monge staircase matrix, we can report all K elements that are at most a given value t (in an arbitrary order) in $O(m+n+K)$ expected time.*

Given an $m\times n$ Monge staircase matrix, we can report the row minima in $O(n(1+\log\lceil \frac{m}{n}\rceil)) \le O(m+n)$ expected time.

Remarks. Note that all the translation and transposition operations on the input matrix are to be done implicitly (we just need to remember an offset value per row and per column).

The row minima or lower envelope part of the algorithm here is similar to (and is modelled after) the randomized divide-and-conquer algorithm by Millman et al. [32] for discrete lower envelope of pseudo-lines, which also runs in $O(n(1+\log\lceil \frac{m}{n}\rceil))$ expected time. (For pseudo-lines, the conflict list computation is easier and does not require halfplane range reporting.)

Note how crucial it is that we solve both the $(\le t)$-reporting and the lower envelope problem simultaneously: the $(\le t)$-reporting algorithm requires lower envelopes of samples, and the lower envelope algorithm requires $(\le t)$-reporting to compute conflict lists.

Certain applications require solving the row minima problem for *double staircase* matrices. Aggarwal and Klawe [3] observed that such matrices can be decomposed into staircase matrices, but alternatively, it is straightforward to adapt our row minima algorithm directly to handle double staircase matrices (since such matrices are closed under transposition and the lower

envelope of the corresponding pseudo-segments still has linear complexity).

4 ($\leq t$)-Reporting for Totally Monotone Matrices

We now investigate the ($\leq t$)-reporting problem more generally for an arbitrary totally monotone (complete) matrix A. We don't have symmetry now, and so will propose a different recursive approach.

In this section, we let n denote the *maximum* number of columns in A (instead of expected). We still use m to denote the *expected* number of rows in A, and K denote the *expected* number of output elements. Redefine $T(m, n, K)$ be the expected time needed to solve the problem under these parameters.

4.1 First almost linear algorithm.

A *(pseudo-)concave chain* refers to the lower envelope of a subset of pseudo-lines in L. In the *few-concave-chains* case of the problem, the points of P lie on g concave chains. For each point p, we are given a label of the chain it is on, and its incident pseudo-line. (By a linear scan over P, we can thus obtain a sorted list of points on each chain.) Let $T_{\text{chains}}(m, n, K; g)$ denote the expected time needed to solve this special case of the problem.

Our algorithm consists of two parts: a reduction of the general problem to the few-concave-chains case, and a reduction of the few-concave-chains case back to smaller instances of the general problem.

Part I: reducing to the few-concave-chains case. For each $i = 0, \ldots, \log n - 1$, let R_i be a random sample of L where each element is chosen independently with probability $\frac{1}{2^i}$ (these $\log n$ samples are chosen independently).

For each $p \in P$, find the smallest i^* such that p is below $\text{LE}(R_{i^*})$, and lift p upward to a point $p_\uparrow$ on $\text{LE}(R_{i^*})$. (See Figure 4.) Let $P_\uparrow = \{p_\uparrow : p \in P\}$, which is a point set lying on $\log n$ concave chains. We solve the problem for $P_\uparrow$ and L. Afterwards, we can find the pseudo-lines below each $p \in P$ by a linear search over the pseudo-lines below $p_\uparrow$. (If $p_\uparrow$ does not exist, we do linear search over all n pseudo-lines.)

Expected output size. Fix a point $p \in P$ having k pseudo-lines below it. The probability that p is above $\text{LE}(R_i)$ is $1-(1-\frac{1}{2^i})^k$. Thus, the probability that $i^* = i$ is $(1-\frac{1}{2^i})^k \cdot (1-(1-\frac{1}{2^{i-1}})^k) \cdot (1-(1-\frac{1}{2^{i-2}})^k) \cdots$. Let $i_0 = \lfloor \log k \rfloor$. It follows that $\Pr[i^* = i_0] = \Omega(1)$, and for $i > i_0$, $\Pr[i^* = i] \leq O(\frac{k}{2^i})^2$ (as a loose upper bound).

Let z be the number of pseudo-lines above p and below $p_\uparrow$. Conditioned on $i^* = i$, the expected value of z is at most 2^i. Thus, $\mathbb{E}[z] \leq \mathbb{E}[2^{i^*}] \leq 2^{i_0} + \sum_{i=i_0+1}^{\infty} 2^i \cdot \Pr[i^* = i] \leq O(2^{i_0} + \sum_{i=i_0+1}^{\infty} 2^i (\frac{k}{2^i})^2) = O(k)$. It follows that the expected total output size for the problem for $P_\uparrow$ and L is $O(K)$.

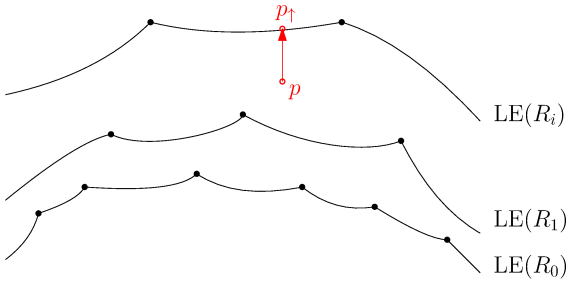

Figure 4: Lower envelopes of random samples, and lifting of a point p. (In general, the envelopes might intersect.)

As an aside, note that conditioned on $i^* = i_0$, the probability that $z \leq \alpha k$ is at least $1-(1-\frac{1}{2^{i_0}})^{\alpha k} = \Omega(\alpha)$. Since $\Pr[i^* = i_0] = \Omega(1)$, we get $\Pr[z \leq \alpha k] = \Omega(\alpha)$ unconditionally, for any $\alpha \leq O(1)$. This fact will be useful later.

Running time. It remains to analyze the time needed to compute i^* and $p_\uparrow$ for every $p \in P$. For each $i = 0$ to $\log n - 1$, we run SMAWK [4] on P and R_i to evaluate $\text{LE}(R_i)$ at the x-coordinates of P; for each point $p \in P$, if p is below $\text{LE}(R_i)$, we set $p_\uparrow$ to the point on $\text{LE}(R_i)$ at p's x-coordinate (found by SMAWK) and remove p from P, before proceeding to the next iteration.

SMAWK runs in time linear in $|P|$ and $|R_i|$. The total expected size of R_i over all i is $\sum_i \frac{n}{2^i} = O(n)$. Fix a point $p \in P$ that has k pseudo-lines below it. Then p participates in $O(i^* + 1)$ calls to SMAWK, and $E[i^*] \leq O(k)$, since we have earlier shown the stronger statement $E[2^{i^*}] = O(k)$. It follows that the total expected running time of all the calls to SMAWK is at most $O(m + n + K)$. Thus,

$$T(m, n, K) \leq O(m + n + K) + T_{\text{chains}}(m, n, O(K); \log n). \tag{4.8}$$

Part II: reducing the few-concave-chains case to smaller instances. For each point $p \in P$, define slope(p) to be the pseudo-slope of the pseudo-line that p is incident on; define the *range* of p to be the interval $[\text{slope}(p^-), \text{slope}(p^+))$, where p^- and p^+ are the predecessor and successor of p among the points of P on p's chain.

Divide $[1, n]$ into $\frac{n}{b}$ intervals of length b, for a parameter b to be set later. For each interval I, let L_I be the set of all pseudo-lines of L with pseudo-slopes in I, and let P_I be the set of all points of P whose ranges

intersect I. By a linear scan over the pre-sorted set P, we can generate all P_I's, each sorted by x. For each I, we solve the problem for P_I and L_I recursively.

Afterwards, for each $p \in P_I$ found to be above a pseudo-line $\ell \in L_I$ by the recursive calls, we search for all points on p's chain that are above ℓ (and have not been found before), by doing a linear scan in both directions from p.

Correctness. Consider a pseudo-line $\ell \in L_I$ for a given interval I. Suppose that some point of P on a concave chain γ is above ℓ. We claim that at least one point of P_I is above ℓ (and so the linear scans afterwards will find all points of P above ℓ). Let v be the vertex of γ that is defined by a pair of pseudo-lines (ℓ', ℓ'') with the pseudo-slope of ℓ between the pseudo-slopes of ℓ' and ℓ''. Consider the predecessor point v^- and successor point v^+ of v among the points of P on γ. Then slope(ℓ) lies in the range of v^- and the range of v^+. So, $v^-, v^+ \in P_I$. As one of v^- and v^+ must be above ℓ, the claim is proved.

Running time. Consider an interval I of size b and a concave chain γ. Consider the portion γ_I of γ that is defined by pseudo-lines with pseudo-slopes in I. The only points of P on γ that have ranges intersecting I are the points of P on γ_I, plus 2 extra points (the predecessor of the leftmost point on γ_I and the successor of the rightmost point on γ_I). Summing over all g chains and all $\frac{n}{b}$ intervals, we can then bound $\sum_I |P_I|$ by $m + \frac{2gn}{b}$.

The linear scans after the recursive calls take $O(K)$ additional time. Thus,

$$T_{\text{chains}}(m, n, K;\, g) \;\leq\; O(m+n+K) + \sum_{i=1}^{n/b} T(m_i, b, K_i), \tag{4.9}$$

for some m_i's and K_i's with $\sum_i m_i \leq m + \frac{2gn}{b}$ and $\sum_i K_i \leq K$.

Putting it all together. Set $g = \log n$ and $b = \log^2 n$. Combining (4.8) and (4.9) gives the following recurrence:

$$T(m, n, K) \leq O(m+n+K) + \sum_{i=1}^{n/\log^2 n} T(m_i, \log^2 n, K_i),$$

for some m_i's and K_i's with $\sum_i m_i \leq m + O(\frac{n}{\log n})$ and $\sum_i K_i \leq O(K)$. (For the base case, if n drops below a constant, $T(m, O(1), K) = O(m)$.)

Expanding the recurrence for $O(\log^* n)$ levels of recursion (and noting that $O(\frac{n}{\log n} + \frac{n}{\log(\log^2 n)} + \cdots) = O(n)$), we see that $T(m, n, K) = O((m+n)\log^* n + K2^{O(\log^* n)})$.

Remarks. Slight improvement in the m term is possible: We can initially reduce m to $O(K)$, as mentioned before by running SMAWK in $O(m+n)$ time, and removing points of P below LE(L). Thus, $T(m, n, K) \leq O(m+n) + T(K, n, K) = O(m + n\log^* n + K2^{O(\log^* n)})$.

Alternatively, we can stop after $c+1$ levels of recursion, and switching to the trivial bound $T(m, n, K) = O(mn)$ for the base case. This gives $T(m, n, K) = O((m+n)\log^{(c)} n + K)$ for any constant c.

Slight improvement in the m term is again possible: In Part I, the number of points actually drops to $O(n)$ in expectation, since on each edge of each chain, only two points (the leftmost and rightmost) may have nonempty ranges, and the expected total size of the chains is $O(\sum_i \frac{n}{2^i}) = O(n)$. Thus, $T(m, n, K) \leq O(m + n + K) + T(O(n), n, K) = O(m + n\log^{(c)} n + K)$.

We remark that the general idea of using logarithmically many random samples of different sizes is inspired by previous work on halfspace range reporting [11], but the way we use samples here (not requiring the standard Clarkson–Shor framework) appears original.

4.2 Refinements.

The running time of the preceding algorithm is already very close to linear, but for those who care about optimizing iterated logarithmic factors, we offer two modifications to improve the time bound further: the first to improve the n term, the second to improve the K term. Both modifications concerns Part I of the algorithm only.

First modification. We take a random sample R_i only for $i = \log s, \ldots, \log n - 1$, where s is a parameter to be set later. This way, the total size of the R_i's is reduced to $O(\sum_{i \geq \log s} \frac{n}{2^i}) = O(\frac{n}{s})$.

For the analysis of the expected output size, fix a point $p \in P$ having k pseudo-lines below it. If $k \geq 4s$, our earlier proofs that $\mathbb{E}[z] \leq \mathbb{E}[2^{i^*}] = O(k)$ and $\Pr[z \leq \alpha k] = \Omega(\alpha)$ go through unchanged. If $k < 4s$, we instead have $\mathbb{E}[z] \leq \mathbb{E}[2^{i^*}] \leq O(s + \sum_{i=\log s+2}^{\infty} 2^i \cdot (\frac{k}{2^i})^2) = O(s)$. The expected output size for the problem for $P_\uparrow$ and L is now bounded by $O(K + sm)$.

Now, p participates in $O(i^* - \log s + 1)$ calls to SMAWK, and $E[i^* - \log s] \leq E[2^{i^*}/s] = O(1 + \frac{k}{s})$. Thus, the total expected cost for SMAWK is $O(m + \frac{K}{s} + \frac{n}{s})$. (Note the sublinearity of the $\frac{n}{s}$ term; nowhere did we spend $O(n)$ time outside of the recursive calls.) We get

$$T(m, n, K) \;\leq\; O(m + \tfrac{n}{s} + K) + T_{\text{chains}}(m, n, O(K + sm);\, \log n). \tag{4.10}$$

Second modification. Next, to mitigate the constant-factor blow-up in the output size K, we repeat the procedure d times (with an independent collection of samples), for another parameter d. Let $p_\uparrow^{(j)}$ be the lifted point $p_\uparrow$ from the j-th repetition. Redefine $p_\uparrow$ as the lowest of $p_\uparrow^{(j)}$ over all j. The number of concave chains now increases by a factor of d, to at most $d \log n$, but the expected output size decreases, as we now show.

Fix a point $p \in P$ having k pseudo-lines below it. Let $z^{(j)}$ be the number of pseudo-lines above p and below $p_\uparrow^{(j)}$. Let z be the number of pseudo-lines above p and below $p_\uparrow$, i.e., $z = \min_j z^{(j)}$. If $k < 4s$, we already have $\mathbb{E}[z] \le \mathbb{E}[z^{(1)}] = O(s)$. Assume $k \ge 4s$. As we have analyzed earlier, $\mathbb{E}[z^{(j)}] \le ck$ for some constant c, and $\Pr[z^{(j)} \le \alpha k] = \Omega(\alpha)$ for any $\alpha \le O(1)$. Thus, $\Pr[z > \frac{2^i k}{d}] \le (1 - \Omega(\frac{2^i}{d}))^d = e^{-\Omega(2^i)}$ for $2^i \le O(d)$. On the other hand, $\Pr[z > 2^i ck] \le (\frac{1}{2^i})^d$ by Markov's inequality. So, $\mathbb{E}[z] \le O(\frac{k}{d} + \sum_{i \ge 1} \frac{2^i k}{d} e^{-\Omega(2^i)} + \sum_{i \ge 1} (\frac{1}{2^i})^{d-1} k) \le O(\frac{k}{d} + \frac{k}{2^d}) = O(\frac{k}{d})$.

It follows that the expected output size for the problem for $P_\uparrow$ and L is $(1 + O(\frac{1}{d}))K + O(sm)$. Thus,

$$T(m,n,K) \le O(d(m + \tfrac{n}{s} + \tfrac{K}{s}) + K) + T_{\text{chains}}(m, n, (1 + O(\tfrac{1}{d}))K + O(sm);\, d \log n). \tag{4.11}$$

Putting it all together. Set $g = d \log n$, $b = \log^2 n$, $s = (\log^* n)^4$, and $d = (\log^* n)^2$. Combining (4.11) and (4.9) yields the following new recurrence (loosely upper-bounding $d \log n$ by $\log^2 n$):

$$T(m,n,K) \le O(m(\log^* n)^2 + \tfrac{n}{(\log^* n)^2} + K) + \sum_{i=1}^{n/\log^2 n} T(m_i, \log^2 n, K_i), \tag{4.12}$$

for some m_i's and K_i's with $\sum_i m_i \le m + O(\frac{n(\log^* n)^2}{\log n})$ and $\sum_i K_i \le (1 + O(\frac{1}{(\log^* n)^2}))K + O(m(\log^* n)^4)$.

To solve the recurrence, define $n_1 = n$ and $n_j = \log^2 n_{j-1}$, and let $h = O(\log^* n)$ be the smallest index such that n_h is below a constant. Note that $\prod_{j=1}^h (1 + O(\frac{1}{(\log^* n_j)^2})) \le e^{O(\sum_{j'} (1/j')^2)} = O(1)$. At the j-th level of the recursion, the sum of the local m values is bounded by $m + O(\sum_{j'=1}^j \frac{n(\log^* n_{j'})^2}{\log n_{j'}}) = m + O(\frac{n(\log^* n_j)^2}{\log n_j})$ (as the series is super-geometric), and the sum of the local K values is $O(K + \sum_{j'=1}^j (m + \frac{n(\log^* n_{j'})^2}{\log n_{j'}})(\log^* n_{j'})^4) = O(K + m(\log^* n)^5 + \frac{n(\log^* n_j)^6}{\log n_j})$. The total cost at the j-th level is $O((m + \frac{n(\log^* n_j)^2}{\log n_j})(\log^* n_j)^2 + \frac{n}{(\log^* n_j)^2} + (K + m(\log^* n)^5 + \frac{n(\log^* n_j)^6}{\log n_j})) = O(m(\log^* n)^5 + \frac{n}{(\log^* n_j)^2} + K)$. Since $\sum_{j=1}^h \frac{1}{(\log^* n_j)^2} \le O(\sum_{j'} (1/j')^2) = O(1)$, the total over all h levels is $T(m,n,K) = O(m(\log^* n)^6 + n + K \log^* n)$.

Final improvement. Finally, we can improve the m term by using our earlier Clarkson–Shor-style divide-and-conquer approach. A recurrence similar to (2.1) still holds, since it does not use symmetry. Choose $b = (\log^* n)^6$. Instead of recursion, use the above new algorithm, which allows us to replace $T_{O(b^3)}(m_i, n_i, K_i)$ with $O(m_i(\log^* b)^6 + n_i + K_i \log^* b)$, and $T(O(\frac{m+K}{b}), n, K')$ with $O(\frac{m+K}{b}(\log^* n)^6 + n + K' \log^* n)$.

Then (2.1) gives $T(m,n,K) = O(m(\log^* \log^* n)^6 + n + K \log^* n)$.

As noted before, we can initially reduce m to $O(K)$ after spending $O(m+n)$ time. We get our final time bound $T(m,n,K) = O(m + n + K \log^* n)$.

THEOREM 4.1. *Given an $m \times n$ totally monotone matrix, we can report all K elements that are at most a given value t in $O(m + n + K \log^* n)$ expected time, or alternatively in $O(m + n \log^{(c)} n + K)$ expected time for any constant c.*

Remarks. The only primitive operations needed by the algorithms in this section are comparisons of the form $A[i,j] \le A[i,j']$ or $A[i,j] \le t$.

We leave open the question of whether the remaining $\log^*$ factor can be further reduced. If there were an $O(m(\log^* n)^{O(1)} + n + K)$ algorithm, then bootstrapping with (2.1) would give a time bound with a doubly iterated logarithm. But we don't even know of an $O(m \log^{O(1)} n + K)$ algorithm, or for that matter, an $O(m^{O(1)} + n + K)$ algorithm.

It is possible to design a randomized algorithm that has optimal but unknown time complexity for the $(\le t)$-reporting problem for totally monotone matrices, similar to results by Larmore [30] on row minima for totally monotone staircase matrices, or Pettie and Ramachandran [35] on minimum spanning trees. The algorithm in Section 4.1 reduces the problem to instances of size at most $\log^{(c)} n$, after a constant number of rounds, and for such extremely small instances, we can build an optimal decision tree by "brute force".

5 Consequences

In this section, we describe applications or variants of our $(\le t)$-reporting algorithms to solve a number of related problems.

5.1 $(\le K)$-selection. There is a simple general randomized reduction of the $(\le K)$-selection problem to $(\le t)$-reporting (this reduction does not require geom-

etry, and has been observed before in other contexts, e.g., in [12]):

First, we may reduce m to be at most K, by running a row minima algorithm, selecting the K-th smallest row minimum t_0, and keeping only rows whose minima are at most t_0.

Now, pick a random sample of n entries from the entire $m \times n$ matrix, and let t be the $\lceil \frac{2K}{m} \rceil$-th smallest of the sample, which can be found in $O(n)$ time. Now, run our $(\leq t)$-reporting algorithm, with a time limit of $T(m, n, 6K)$. If the output contains between K and $6K$ elements, we report the first K smallest elements from the output in $O(m + K)$ additional time.

The expected rank of t is $\lceil \frac{2K}{m} \rceil m$, which is between $2K$ and $3K$ (since $m \leq K$). Straightforward calculations show that the rank of t is between K and $6K$ with probability $\Omega(1)$. Thus, an $O(1)$ expected number of trials suffices. The total expected running time is $O(T(m, n, 4K))$. Consequently, by using Theorems 4.1 and 3.1, we obtain the following:

THEOREM 5.1. *Given an $m \times n$ totally monotone matrix and a number K, we can report the K smallest elements (in an arbitrary order) in $O(m + n + K \log^* n)$ expected time.*

Given an $m \times n$ Monge (complete or staircase) matrix and a number K, we can report the K smallest elements (in an arbitrary order) in $O(m + n + K)$ expected time.

5.2 Row $(\leq k_1, \ldots, k_m)$-selection for totally monotone matrices. We now consider the row $(\leq k_1, \ldots, k_m)$-selection problem for a totally monotone matrix, which is trickier than $(\leq K)$-selection. Let $T_{\text{select}}(m, n, K)$ be the expected time needed to solve this problem where m is the (expected) number of rows, n is the (maximum) number of rows, and K is the (expected) sum $\sum_i k_i$.

In geometric terms, the problem reduces to the following:

> Given a set X of values of (expected) size m, a set L of pseudo-lines of (maximum) size n in the plane, and a number k_x for each $x \in X$, report the first k_x lowest pseudo-lines of L at x-coordinate x for each $x \in X$. We let K be the (expected) sum $\sum_{x \in X} k_x$.

First approach, via sampling. We first give a general randomized reduction from row $(\leq k_1, \ldots, k_m)$-selection to $(\leq t_1, \ldots, t_m)$-reporting, which as mentioned reduces to $(\leq t)$-reporting (this reduction does not require geometry):

Take a sample R of $\frac{n}{2}$ columns. Find the $\min\left\{\frac{3}{4}k_i + c \log n,\, k_i\right\}$-th smallest t_i for the i-th row among the columns in R, by recursion. Run our algorithm for row $(\leq t_1, \ldots, t_m)$-reporting, in $O(T(m, n, K))$ time. Search for the answers among the output entries.

Observe that in the i-th row, the k_i-th smallest among all columns is at most the $(3k_i/4 + c \log n)$-th smallest among the columns in R with high probability (say $1 - n^{-3}$) by a Chernoff bound, for a sufficiently large constant c. (If failure is detected, we can switch to a brute-force quadratic-time algorithm.) Thus, we obtain:

$$\begin{aligned} T_{\text{select}}(m, n, K) \quad \leq \quad & T_{\text{select}}(m, \tfrac{n}{2}, \tfrac{3K}{4} + O(m \log n)) + \\ & O(T(m, n, K) + m + n + K). \end{aligned}$$

By using Theorem 4.1, the recurrence solves to $T_{\text{select}}(m, n, K) = O(m \log n \log^* n + n + K \log^* n)$. The n and K terms are fine; however, the m term has an extra logarithmic factor.

Second approach, via Clarkson–Shor. We now improve the m term by modifying the Clarkson–Shor-style divide-and-conquer algorithm from Section 2. (In contrast, the $(\leq k)$-reporting algorithm from Section 4.1 does not seem adaptable.)

Take a random sample R of L of size $\frac{n}{s}$. Compute the lower envelope $\text{LE}(R)$ and its vertical decomposition $\text{VD}(R)$, in $O(m + \frac{n}{s})$ time as before. For each cell $\Delta \in \text{VD}(R)$, compute its conflict list L_Δ; as before, this takes time $O(\sum_{\Delta \in \text{VD}(R)} |L_\Delta|)$, which has expected value $O((\frac{n}{s})s) = O(n)$.

Let X' be the subset of all $x \in X$ with $k_x > b$. For each $\Delta \in \text{VD}(R)$ with $|L_\Delta| \leq bs$, we recursively solve the subproblem for L_Δ and the x-values in $X - X'$ that lie in the x-projection of Δ. Let X'' be the subset of all x-values in $X - X'$ whose answers found lie above $\text{LE}(R)$. Let X''' be the subset of all x-values in X that lie in the x-projection of cells $\Delta \in \text{VD}(R)$ with $|L_\Delta| > bs$. We recursively solve the subproblem for L and $X' \cup X'' \cup X'''$.

We have $|X'| \leq \frac{K}{b}$. For each $x \in X - X'$, the probability that $x \in X''$, i.e., that the k_x-th lowest point on x is above $\text{LE}(R)$, is at most $\frac{b}{s}$. Thus, the expected size of X'' is $O((\frac{m}{b})s)$. Furthermore, by Clarkson and Shor's analysis, the expected value of $|L_\Delta|$ for the cell Δ intersecting a fixed x is $O(s)$. Thus, the probability that $|L_\Delta| > bs$ is $O(\frac{1}{b})$, and so the expected size of X''' is $O(\frac{m}{b})$.

Choose $s = b^2$. The expected size of $X' \cup X'' \cup X'''$ is then $O(\frac{m+K}{b})$, which is $O(\frac{K}{b})$ since $K \geq m$. Observe that only the x-values of X'' may participate in two recursive calls, and the expected value of $\sum_{x \in X''} k_x$ is at most $O(\frac{Kb}{s}) = O(\frac{K}{b})$. We thus obtain the following

recurrence:

$$T_{\text{select}}(m,n,K) \leq \sum_i T_{\text{select}}(m_i,n_i,K_i) + T_{\text{select}}(O(\tfrac{K}{b}),n,K') + O(m+n),$$

for some m_i's, n_i's, and K_i's with $\sum_i n_i = O(n)$, $\max_i n_i = O(b^3)$, $\sum_i m_i \leq m$, and $\sum_i K_i + K' \leq K + O(\frac{K}{b})$ and $K' \leq K$.

By the first approach, we can replace $T_{\text{select}}(m_i,n_i,K_i)$ with $O(m_i \log^{O(1)} n_i + n_i + K_i \log^* n_i)$. Letting $K'' = K - K'$, we get:

$$T_{\text{select}}(m,n,K) \leq O(m\log^{O(1)} b + n + (K'' + \tfrac{K}{b})\log^* b) + T_{\text{select}}(O(\tfrac{K}{b}),n,K-K'').$$

Now we expand the recurrence using an increasing sequence $b_1, b_2, \ldots$:

$$\begin{aligned} &T_{\text{select}}(m,n,K) \\ &\leq O\Big((m\log^{O(1)} b_1 + n + K_1''\log^* b_1) \\ &\quad + (\tfrac{K}{b_1}\log^{O(1)} b_2 + n + (K_2'' + \tfrac{K}{b_1})\log^* b_2) + \cdots \\ &\quad + (\tfrac{K}{b_{h-1}}\log^{O(1)} b_h + n + (K_h'' + \tfrac{K}{b_{h-1}})\log^* b_{h-1}) \\ &\quad + \tfrac{K}{b_h} n\Big) \end{aligned}$$

for some K_j'''s with $\sum_j K_j'' \leq K$.

Choose $b_1 = 2$ and $b_j = 2^{b_{j-1}^{\delta}}$ for a sufficiently small constant $\delta > 0$, and let $h = O(\log^* n)$ be the smallest index such that $b_h \geq n$. We conclude that $T_{\text{select}}(m,n,K) = O(m + (n+K)\log^* n) = O((n+K)\log^* n)$ (since $m \leq K$).

THEOREM 5.2. *Given an $m \times n$ totally monotone matrix and numbers $k_1, \ldots, k_m \geq 1$, we can report the k_i smallest elements (in an arbitrary order) in the i-th row, for all $i = 1, \ldots, m$, in $O((n+K)\log^* n)$ expected time.*

Remarks. We leave as an open question whether the time bound could be improved to $O(n + K\log^* n)$, to match the complexity of $(\leq t)$-reporting.

Alternatively, if we use the $O(m + n\log^{(c)} n + K)$ algorithm for $(\leq k)$-reporting as a start, then the first approach gives $T_{\text{select}}(m,n,K) = O(m\log n + n\log^{(c)} n + K)$. In the second approach, by beginning with $b_1 = \log^{(c)} n$, we have $h = O(1)$ and get $T_{\text{select}}(m,n,K) = O((m+n)\log^{(c)} n + K)$.

5.3 Row minima for totally monotone v-matrices. As another application of our $(\leq k)$-reporting algorithm, we consider the row minima problem for an $m \times n$ totally monotone v-matrix.

In geometric terms, the problem corresponds to computing a discrete lower envelope of n pseudo-line segments (or *pseudo-segments*):

> For a set X of m values and a set L of n pseudo-segments in the plane, evaluate the lower envelope of L at the x-coordinates of X.

We assume that the x-coordinates of X and of the endpoints of L have been pre-sorted, and the pseudo-segments of L has been pre-sorted by *pseudo-slope*. More precisely, each pseudo-segment is given a distinct number called the pseudo-slope, with the property that if segments ℓ and ℓ' intersect and ℓ has larger pseudo-slope than ℓ', then ℓ is below ℓ' to the left of the intersection. (A lemma from [13] states that such a numbering exists iff the pseudo-segments are *extendible*.) As is known [24, 38], the lower envelope of n pseudo-segments has at most $O(n\alpha(n))$ vertices.

We need two subroutines:

1. A naive lower envelope algorithm with $O(n^2(1 + \log\lceil\frac{m}{n^2}\rceil))$ running time:

   Draw vertical lines at the endpoints to divide the plane into $O(n)$ slabs. In each slab formed by two consecutive vertical lines, run SMAWK to compute lower envelope (since the pseudo-segments may be treated as pseudo-lines within the slab). The total time for these n subproblems is $O(\sum_{i=1}^{O(n)} n(1 + \log\lceil\frac{m_i}{n}\rceil))$ for some m_i's with $\sum_{i=1}^{O(n)} m_i = m$. The sum is at most $O(n^2(1+\log\lceil\frac{m}{n^2}\rceil))$ by concavity of the logarithm.

2. An algorithm for solving the reporting problem for points and pseudo-segments in $O(m^3 + n + K\log^* n)$ expected time—given a set P of m points and a set L of n pseudo-segments, report all K pairs $(p,\ell) \in P \times L$ with p above ℓ:

   Draw vertical lines at the endpoints to divide the plane into $O(n)$ slabs. For each pair of slabs σ_1 and σ_2, run our reporting algorithm from Theorem 4.1 on the points between σ_1 and σ_2 and the pseudo-segments with left endpoints in σ_1 and right endpoints in σ_2 (for such a subproblem, the pseudo-segments may be treated as pseudo-lines). There are $O(m^2)$ subproblems; each point participates in $O(m^2)$ subproblems, but each pseudo-segment participates in one. By Theorem 4.1, the total expected time is $O(m^3 + n + K\log^* n)$.

Choose a hierarchy of random samples $R_1 \subset R_2 \subset \cdots \subset R_\ell = L$, where each element of L is in R_i with probability $\frac{1}{n_i}$, and $n_1 = n$, $n_{i+1} = n_i^{3/4}$, and $n_\ell = O(1)$ (with $\ell = O(\log\log n)$). At the i-th

iteration, we assume that we have already computed the lower envelope $\mathrm{LE}(R_i)$, its vertical decomposition $\mathrm{VD}(R_i)$, and the conflict lists of the cells of $\mathrm{VD}(R_i)$. For each cell $\Delta \in \mathrm{VD}(R_i)$, we compute the lower envelope $\mathrm{LE}(R_{i+1} \cap L_\Delta)$ inside Δ by the above subroutine 1. By gluing all these lower envelopes, we obtain $\mathrm{LE}(R_{i+1})$.

The total time for these steps is

$$O\left(\left[\textstyle\sum_i \sum_{\Delta \in \mathrm{VD}(R_i)} |R_{i+1} \cap L_\Delta|^2 \cdot (1 + \log\left\lceil \frac{m_\Delta}{|R_{i+1} \cap L_\Delta|^2} \right\rceil)\right]\right),$$

where $\sum_{\Delta \in \mathrm{VD}(R_i)} m_\Delta = m$ for each i. By concavity of the logarithm, the sum is at most $O(S(1 + \log\left\lceil \frac{m \log\log n}{S} \right\rceil))$, where $S = \sum_i S_i$ and $S_i = \sum_{\Delta \in \mathrm{VD}(R_i)} |R_{i+1} \cap L_\Delta|^2$. Now, $\mathbb{E}[S_i] = \mathbb{E}\left[\sum_{\Delta \in \mathrm{VD}(R_i)} (\frac{|L_\Delta|}{n_{i+1}})^2\right]$. By a standard analysis of Clarkson and Shor [17, 34], this expectation is $O((\frac{n}{n_i}\alpha(\frac{n}{n_i}))(\frac{n_i}{n_{i+1}})^2) = O((\frac{n}{n_i}\alpha(n))\sqrt{n_i}) = O(n\alpha(n)/\sqrt{n_i})$. It follows that $\mathbb{E}[S] = O(\sum_i n\alpha(n)/\sqrt{n_i}) = O(n\alpha(n))$. So, $S = O(n\alpha(n))$ with probability $\Omega(1)$. We can repeat an $O(1)$ expected number of times to ensure success. The time bound is $O(n\alpha(n)(1 + \log\left\lceil \frac{m \log\log n}{n\alpha(n)} \right\rceil)) \le O(n\alpha(n) \log\lceil \frac{m}{n} \rceil + n\alpha(n) \log\log\log n)$.

Before proceeding to the next iteration, we still need to compute the conflict lists of the cells of $\mathrm{VD}(R_{i+1})$. It suffices to compute the conflict list of every vertex v of $\mathrm{LE}(R_{i+1})$, where the conflict list of v is defined as the list of all pseudo-segments below v. This is because a pseudo-line segment ℓ intersects a cell Δ iff ℓ is below at least one of the two vertices of Δ or at least one of the endpoints of ℓ is inside Δ. For each cell $\Delta \in \mathrm{VD}(R_i)$, we want to find the pseudo-segments of L_Δ below each vertex of $\mathrm{LE}(R_{i+1} \cap L_\Delta)$ inside Δ. By the above subroutine 2, this takes $O(|R_{i+1} \cap L_\Delta|^3 + |L_\Delta| + K_\Delta \log^* n)$ expected time per cell Δ, where K_Δ is the total size of the conflict lists of the vertices of $\mathrm{LE}(R_{i+1})$ inside Δ. The total expected cost over all cells is proportional to

$$\mathbb{E}\left[\textstyle\sum_{\Delta \in \mathrm{VD}(R_i)} ((\frac{|L_\Delta|}{n_{i+1}})^3 + |L_\Delta|) + \sum_{\Delta' \in \mathrm{VD}(R_{i+1})} |L_{\Delta'}| \log^* n\right].$$

By Clarkson–Shor, this is $O((\frac{n}{n_i}\alpha(\frac{n}{n_i}))(\frac{n_i}{n_{i+1}})^3 + n_i) + (\frac{n}{n_{i+1}}\alpha(\frac{n}{n_{i+1}}))n_{i+1} \log^* n) = O((\frac{n}{n_i}\alpha(n))(n_i^{3/4} + n_i) + n\alpha(n) \log^* n) = O(n\alpha(n) \log^* n)$. The total over all $\ell = O(\log\log n)$ iterations is $O(n\alpha(n) \log^* n \log\log n)$.

The overall time bound is $O(n\alpha(n) \log^* n \log\log n + n\alpha(n) \log\lceil \frac{m}{n} \rceil)$. In the case of $n > m$, we can divide into $\frac{n}{m}$ subproblems of size m and obtain an $O(\frac{n}{m}(m\alpha(m) \log^* m \log\log m)) = O(n\alpha(m) \log^* m \log\log m)$ time bound.

THEOREM 5.3. *Given an $m \times n$ totally monotone v-matrix, we can find all row minima in $O(n\alpha(m) \log^* m \log\log m + n\alpha(m) \log\lceil \frac{m}{n} \rceil) \le O(n\alpha(m) \log^* m \log\log m + m)$ expected time.*

Remarks. The $\log^*$ factor may be removed in the Monge case, by using the reporting algorithm from Theorem 2.1 in the implementation of subroutine 2.

The $n \log\log m$ barrier seems harder to break. This appears to require implementing subroutine 2 in, say, $O(m \log^{O(1)} n + n + K)$ time instead of $O(m^{O(1)} + n + K)$.

For large n, it is possible to achieve linear running time by a more naive approach: Namely, divide the plane into b vertical slabs each containing $\frac{m}{b}$ x-values. If a pseudo-segment spans multiple slabs, divide it into a left, middle, and right piece where each left/right piece is contained in a slab, and all the middle pieces have x-coordinates from b distinct values. We recursively solve the problem inside each slab, ignoring the middle pieces. For the middle pieces, we put pieces with the same x-projection in the same class, and then compute the discrete lower envelope of each class by SMAWK (since within the same class, the pseudo-segments behave like pseudo-lines). The total cost of these $O(b^2)$ calls to SMAWK is $O(b^2 m + n)$ (since each middle piece belongs to just one class). We can combine the envelopes in $O(b^2 m)$ additional time. The total time over all $O(\log_b n)$ levels of recursion is $O((b^2 m + n) \log_b n)$. Setting $b = n^{\delta/2}$ yields $O(n + m^{1+\delta})$ for any constant $\delta > 0$. This upper bound may not be too exciting, but it disproves a conjecture of Klawe [26] that her $\Omega(n\alpha(n))$ lower bound for $m = \Theta(n)$ could be strengthened to $\Omega(n\alpha(m))$ for $n > m$.

Acknowledgement. I thank Jeff Erickson for a conversation about SMAWK and staircase matrices, which led to the start of this work.

References

[1] Peyman Afshani, Timothy M. Chan, and Konstantinos Tsakalidis. Deterministic rectangle enclosure and offline dominance reporting on the RAM. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 77–88, 2014. doi:10.1007/978-3-662-43948-7_7.

[2] Pankaj K. Agarwal and Micha Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM J. Comput.*, 34(3):526–552, 2005. doi:10.1137/S0097539703433900.

[3] Alok Aggarwal and Maria M. Klawe. Applications of generalized matrix searching to geometric algorithms. *Discret. Appl. Math.*, 27(1-2):3–23, 1990. doi:10.1016/0166-218X(90)90124-U.

[4] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

[5] Alok Aggarwal and James K. Park. Notes on searching in multidimensional monotone arrays. In *Proc. 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 497–512, 1988. doi:10.1109/SFCS.1988.21966.

[6] Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weight k-link path graphs with the concave Monge property and applications. *Discret. Comput. Geom.*, 12:263–280, 1994. doi:10.1007/BF02574380.

[7] Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *Proc. 3rd ACM Symposium on Computational Geometry (SoCG)*, pages 278–290, 1987. doi:10.1145/41958.41988.

[8] Wolfgang W. Bein, Mordecai J. Golin, Lawrence L. Larmore, and Yan Zhang. The Knuth–Yao quadrangle-inequality speedup is a consequence of total monotonicity. *ACM Trans. Algorithms*, 6(1):17:1–17:22, 2009. doi:10.1145/1644015.1644032.

[9] Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discret. Appl. Math.*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

[10] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012. doi:10.1137/090766863.

[11] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000. doi:10.1137/S0097539798349188.

[12] Timothy M. Chan. On enumerating and selecting distances. *Int. J. Comput. Geom. Appl.*, 11(3):291–304, 2001. doi:10.1142/S0218195901000511.

[13] Timothy M. Chan. On levels in arrangements of curves. *Discret. Comput. Geom.*, 29(3):375–393, 2003. doi:10.1007/s00454-002-2840-2.

[14] Timothy M. Chan. Near-optimal randomized algorithms for selection in totally monotone matrices. Manuscript, 2020.

[15] Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discret. Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/s00454-016-9784-4.

[16] Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985. doi:10.1007/BF01934990.

[17] Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. doi:10.1007/BF02187740.

[18] Karen L. Daniels, Victor J. Milenkovic, and Dan Roth. Finding the largest area axis-parallel rectangle in a polygon. *Comput. Geom.*, 7:125–148, 1997. doi:10.1016/0925-7721(95)00041-0.

[19] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming II: convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992. doi:10.1145/146637.146656.

[20] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 8(1):415–428, 2012. doi:10.4086/toc.2012.v008a019.

[21] Zvi Galil and Kunsoo Park. Dynamic programming with convexity, concavity, and sparsity. *Theor. Comput. Sci.*, 92(1):49–76, 1992. doi:10.1016/0304-3975(92)90135-3.

[22] Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in Monge and partial Monge matrices are equivalent to predecessor search. *ACM Trans. Algorithms*, 16(2):16:1–16:24, 2020. doi:10.1145/3381416.

[23] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

[24] Sergiu Hart and Micha Sharir. Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6(2):151–178, 1986. doi:10.1007/BF02579170.

[25] Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and partial Monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

[26] Maria M. Klawe. Superlinear bounds for matrix searching problems. *J. Algorithms*, 13(1):55–78, 1992. doi:10.1016/0196-6774(92)90005-W.

[27] Maria M. Klawe and Daniel J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM J. Discret. Math.*, 3(1):81–97, 1990. doi:10.1137/0403009.

[28] Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Trans. Algorithms*, 6(2):30:1–30:18, 2010. doi:10.1145/1721837.1721846.

[29] Dina Kravets and James K. Park. Selection and sorting in totally monotone arrays. *Math. Syst. Theory*, 24(3):201–220, 1991. doi:10.1007/BF02090398.

[30] Lawrence L. Larmore. An optimal algorithm with unknown time complexity for convex matrix searching. *Inf. Process. Lett.*, 36(3):147–151, 1990. doi:10.1016/0020-0190(90)90084-B.

[31] Michael McKenna, Joseph O'Rouke, and Subhash Suri. Finding the largest rectangle in an orthogonal polygon. In *Proc. 23rd Allerton Conference on Communication, Control and Computing*, 1985.

[32] David L. Millman, Steven Love, Timothy M. Chan,

and Jack Snoeyink. Computing the nearest neighbor transform exactly with only double precision. In *Proc. 9th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 66–74, 2012. doi:10.1109/ISVD.2012.13.

[33] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n/\log\log n)$ time. In *Proc. 18th European Symposium on Algorithms (ESA), Part II*, pages 206–217, 2010. doi:10.1007/978-3-642-15781-3_18.

[34] Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1994.

[35] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002. doi:10.1145/505241.505243.

[36] Baruch Schieber. Computing a minimum weight k-link path in graphs with the concave Monge property. *J. Algorithms*, 29(2):204–222, 1998. doi:10.1006/jagm.1998.0955.

[37] Raimund Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 37–68. Springer, 1993.

[38] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

[39] F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proc. 12th ACM Symposium on Theory of Computing (STOC)*, pages 429–435, 1980. doi:10.1145/800141.804691.

A Applications

We mention a few applications of our row minima algorithm for Monge staircase matrices, to various problems in computational geometry and dynamic programming:

- Given two disjoint convex n-gons P and Q in the plane, we want to find the nearest (or farthest) invisible vertex of Q for each vertex of P. Aggarwal and Klawe [3] observed that this problem can be reduced to row minima in Monge staircase matrices. We can now solve the problem in $O(n)$ expected time.

- Given an orthogonally convex n-gon P in the plane, we want to find the largest-area axis-aligned rectangle contained in P. Daniels et al. [18] used Klawe and Kleitman's algorithm to solve this problem in $O(n\alpha(n))$ time (the Monge property was observed earlier by McKenna et al. [31]). We can now solve the problem in $O(n)$ expected time.[5] Daniels et al. [18] then showed that for an x-monotone polygon (where every vertical line intersect the polygon at most twice), the same problem can be solved in $O(n\alpha(n)\log n)$ time, by divide-and-conquer. Our improvement yields an $O(n \log n)$ expected time bound.

- Given a complete DAG with n vertices whose edge weights satisfy the convex Monge property, and given two vertices s and t and a number k, we want to find the shortest path from s to t using exactly k links. Straightforward dynamic programming reduces the problem to k instances of the row minima problem in convex Monge, upper triangular matrices. Our new algorithm implies an $O(nk)$ expected time bound, improving a previous bound of $O(nk\alpha(n))$ [9]. (Note that the problem in the concave Monge case has received much more attention [6, 36].)

- In a seminal work, F. Yao [39] studied the problem of evaluating a recurrence of the following form: for any $1 \le i < j \le n$,

$$c(i,j) = w(i,j) + \min_{i<k\le j}(c(i,k-1) + c(k,j)),$$

  where the values $c(i,i)$ are given. Naive dynamic programming requires $O(n^3)$ time. Yao described an $O(n^2)$-time algorithm to compute all $c(i,j)$ values when w satisfies the concave Monge property and, in addition, $w(i,j) \le w(i',j')$ whenever $[i,j] \subset [i',j']$. Aggarwal and Park [5] noted an $O(n^2\alpha(n))$-time algorithm for the less studied case when w satisfies the convex Monge property and, in addition, $w(i,j) \ge w(i',j')$ whenever $[i,j] \subset [i',j']$. We can now improve the time bound in this convex case to $O(n^2)$. This can be most easily seen by following an approach of Bein et al. [8], who reduced the problem to $O(n)$ instances of row minima in a Monge, upper triangular matrix. Bein et al. described the reduction for the concave case, but the same approach works in the convex case.

There were also a few algorithms in the literature on planar or surface-embedded graphs (e.g., [28, 10, 33]) that used Klawe and Kleitman's row minima algorithm as an intermediate step, which now may be replaced by our new algorithm; however, the final running time in these graph algorithms appears to be dominated by other steps. There were also a number of important applications of Klawe and Kleitman's algorithm to speed up certain types of dynamic programming that arise from computational biology [19], but these require

[5] On a related note, Aggarwal and Suri [7] studied the problem of finding the largest empty axis-aligned rectangle for a set of n points in the plane, and gave an $O(n \log^2 n)$-time algorithm, using row minima in Monge staircase matrices and other types of Monge partial matrices as subroutines. Our result does not seem to immediately improve their overall time bound, but perhaps with more effort, some improvement might be possible...

matrix searching in an "online setting", for which our new algorithm does not seem applicable.

Our ($\leq K$)-selection algorithm has at least one interesting application in computational geometry:

- Given a convex n-gon P in the plane, we want to report the K farthest pairs of vertices (in an arbitrary order). For general planar point sets, the problem requires $O(n \log n + K)$ time [12]. Kravets and Park [29] used their ($\leq K$)-selection algorithm to obtain an $O(n + K \log \frac{n^2}{K})$-time algorithm for the problem for a convex polygon. Our result implies an optimal $O(n + K)$-time randomized algorithm.

B A Simple Randomized Alternative to SMAWK

In this appendix, we briefly sketch a simple $O(m + n)$-time randomized algorithm for the original problem of computing row minima in a complete $m \times n$ totally monotone matrix, which may serve as an alternative to SMAWK [4]. Now, SMAWK is already a simple (but tricky) algorithm. In some sense, the new algorithm is conceptually more straightforward, and thus may have some pedagogical value. (It is not easier to implement, however.) Millman et al. [32] have already given an alternative randomized linear-time algorithm via sampling, but the following randomized incremental algorithm is a little simpler.

Though the main idea has its roots from known randomized incremental algorithms for planar lower envelopes of lines (i.e., convex hulls in the dual) [17, 34], we will desribe it in matrix terms. Let A be the input $m \times n$ concave totally monotone matrix. By padding with extra rows or columns, we may assume that $m = n$. For each $i = 1, \ldots, n$, we want to compute the index j_i that minimizes $A[i, j_i]$. Note that $j_1 \leq j_2 \leq \cdots \leq j_n$, because of total monotonicity. Set $j_0 = 0$ and $j_{n+1} = n + 1$. In addition, for each $j = 1, \ldots, n$, we will also compute the index i_j for which $j_{i_j} \leq j < j_{i_j+1}$.

Algorithm. We will describe the randomized incremental algorithm "backwards", to make the analysis easier: Namely, the algorithm randomly picks a row i^* and a column j^*, delete the row and the column, solve the problem recursively for the resulting $(n-1) \times (n-1)$ matrix, then add back the row and the column and update the answers. (See Figure 5.)

To add back row i^*: We just compute j_{i^*} by finding the minimum of $A[i, j]$ over all $j = j_{i^*-1}, \ldots, j_{i^*+1}$ (by naive linear search), and update i_j appropriately for all $j = j_{i^*-1}, \ldots, j_{i^*+1}$.

To add back column j^*: We find the the smallest $i' \leq i_{j^*-1}$ with $A[i', j] < A[i', j_{i'}]$ (by naive linear search), and the largest $i'' > i_{j^*-1}$ with $A[i'', j] < A[i'', j_{i''}]$ (by another naive linear search). We then reset j_i to j^* for all $i = i', \ldots, i''$, and update i_j appropriately for all $j = j_{i'}, \ldots, j_{i''}$.

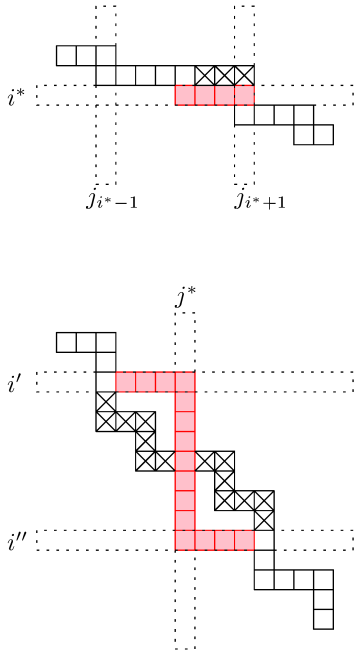

Figure 5: How the answer staircase S changes when inserting a row i^* (top) or when inserting a column j^* (bottom).

Analysis. We bound the expected running time by using a standard backwards analysis [34, 37]. Define the *answer staircase* S to be an orthogonal polygonal path through positions $(0, 0), (0, j_1), (1, j_1), (2, j_1), (2, j_2), \ldots, (n, j_n), (n, n)$, with respect to the *final* answers for the entire $n \times n$ matrix A.

The cost of adding back row i^* is proportional to the sum of the lengths of the horizontal edges in S contained in rows $i^* - 1$ and i^*. Since the sum of these lengths over all i^* is $O(n)$, the expected cost for a random i^* is $O(1)$.

The cost of adding back column j^* is proportional to the sum of the lengths of the vertical edge in S contained in column j^* and its two adjacent horizontal edges in S. (If column j^* contains just a single position of S and not a vertical edge, the cost is $O(1)$.) Since the sum of these lengths over all j^* is $O(n)$ (as each edge is counted $O(1)$ times), the expected cost for a random j^* is $O(1)$.

Thus, the total expected cost for an $n \times n$ matrix satisfies the recurrence $T(n) \leq T(n-1) + O(1)$, yielding $T(n) = O(n)$. For an $m \times n$ input matrix, the expected running time is thus $O(m + n)$.

Remarks. In actual implementation, the active rows (resp. columns) need to be kept in a sorted, doubly linked list, to avoid shifting indices when deleting rows (resp. columns); for example, $i^* \pm 1$ should actually be the successor/predecessor of i^* in the linked list. Because of all the pointer manipulations and bookkeeping of the extra indices j_i's, the above algorithm probably would not compete as well with SMAWK in practice (which has very efficient existing implementations). Also, the above algorithm does not seem to work in the online setting, unlike SMAWK.

The random-sampling-based algorithm by Millman et al. [32] (which achieves a better $O(n(1 + \log \lceil \frac{m}{n} \rceil))$ expected cost in the non-square case) similarly removes columns at random but does not remove rows; however, it is less simple and requires extra binary searches.