Near-Optimal Randomized Algorithms for Selection in Totally Monotone Matrices

Timothy M. Chan*

October 30, 2020

Abstract

We revisit classical problems about searching in totally monotone matrices, which have many applications in computational geometry and other areas. In a companion paper, we gave new (near-)linear-time algorithms for a number of such problems. In the present paper, we describe new subquadratic results for more basic problems, including the following:

- A randomized algorithm to select the K-th smallest element in an $n \times n$ totally monotone matrix in $O(n^{4/3} \operatorname{polylog} n)$ expected time; this improves previous $O(n^{3/2} \operatorname{polylog} n)$ algorithms by Alon and Azar [SODA'92], Mansour et al. (1993), and Agarwal and Sen (1996).
- A near-matching lower bound of $\Omega(n^{4/3})$ for the problem (which holds even for Monge matrices).
- A similar result for selecting the k_i-th smallest in the i-th row for all i.
- In the case when all k_i's are the same, an improvement of the running time to $O(n^{6/5} \operatorname{polylog} n)$.
- Variants of all these bounds that are sensitive to K (or $\sum_i k_i$).

These matrix searching problems are intimately related to problems about arrangements of pseudo-lines. In particular, our selection algorithm implies an $O(n^{4/3} \operatorname{polylog} n)$ algorithm for computing incidences between n points and n pseudo-lines in the plane. This improves, extends, and simplifies a previous method by Agarwal and Sharir [SODA'02].

1 Introduction

Selection in totally monotone matrices. Totally monotone matrices arise in many subareas of algorithms, including computational geometry, dynamic programming speedups, shortest paths in planar graphs, and combinatorial optimization (see various surveys, e.g., [9, 25]). An $m \times n$ matrix A is *totally monotone* iff for every $i < i'$ and $j < j'$,

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

Early work on matrix searching focused on the basic problem of computing the minimum of each row: the well-known result by Aggarwal, Klawe, Moran, Shor, and Wilber [5] showed that all

*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.

row minima in a totally monotone matrix can be found in linear time. The input matrix may be given implicitly—we only assume that any matrix entry can be evaluated on demand in constant time. (In fact, their algorithm only requires comparisons of elements in a common row.)

Two important lines of research have subsequently followed, investigating (i) the row minima problem in more general types of totally monotone "partial" matrices, as well as (ii) other, more general searching problems in totally monotone matrices. The focus of this paper is on the second direction. We study the following fundamental matrix searching problems:

- *K-selection*: compute the K-th smallest element.
- *t-ranking*: count the number of elements that are at most t.
- *row $(k_1, \ldots, k_m)$-selection*: for each $i = 1, \ldots, m$, output the k_i-th smallest in the i-th row.
- *row $(t_1, \ldots, t_m)$-ranking*: for each $i = 1, \ldots, m$, count the number of elements in the i-th row that are at most t_i.

We reuse K to denote the output count in the t-ranking problem, or $\sum_i k_i$ in the row $(k_1, \ldots, k_m)$-selection problem, or the total output count in the row $(t_1, \ldots, t_m)$-ranking problem.

For example, the row minima problem corresponds to row $(1, \ldots, 1)$-selection. Ranking and selection problems are closely related (ranking easily reduces to selection by binary search, and ranking algorithms can often be modified to yield selection algorithms). Frederickson and Johnson [23, 24] gave optimal algorithms for these types of problems for sorted matrices (which are less general), but the corresponding problems for totally monotone matrices have not yet been satisfactorially solved.

Previous results. For small K, there were algorithms with running time close to $O(m+n+K)$, as first described by Kravets and Park [30] in SODA'90 and improved and extended by the author in a companion paper [13]. These algorithms actually solve stronger versions of the problems, for example, selecting the K smallest elements rather than just the K-th smallest. However, in the worst case when $K = \Theta(n^2)$, the running time is near-quadratic.

A series of papers in the early 1990s aimed for subquadratic algorithms for these problems:

- Alon and Azar [6] (in SODA'92) showed that the row $(k_1, \ldots, k_m)$-selection problem can be solved using $O(n\sqrt{m}\log n\sqrt{\log m})$ comparisons. However, the running time may be large.
- Mansour et al. [32] gave an algorithm for row $(k, \ldots, k)$-selection running in $O(n\sqrt{m\log n}+m)$ time. However, their algorithm could not be generalized to row $(k_1, \ldots, k_m)$-selection.
- Agarwal and Sen [3] described algorithms for K-selection and row $(k_1, \ldots, k_m)$-selection running in $O((m+n)\sqrt{n}\log n)$ time.

Notice that in the case of square matrices with $m = n$, all three results achieve the same bound of $\widetilde{O}(n^{3/2})$, ignoring logarithmic factors.[1] Agarwal and Sen explicitly posed the question of whether the time complexity could be improved to $\widetilde{O}(n^{4/3})$, but no improvements have been reported in two and a half decades.

[1] Throughout the paper, the $\widetilde{O}$ notation hides polylogarithmic factors.

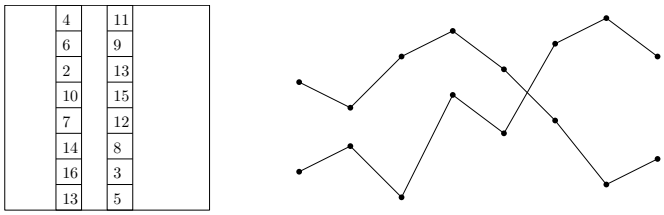

Figure 1: Columns in a totally monotone matrix map to pseudo-lines. (Figure taken from [13].)

New results. We present new algorithms for all four problems, K-selection, t-ranking, row $(k_1, \ldots, k_m)$-selection, and row $(t_1, \ldots, t_m)$-ranking, with expected running time $\widetilde{O}((mn)^{2/3}+m+n)$. The algorithms are randomized (Las Vegas). For example, in the square matrix case, the time bound is $\widetilde{O}(n^{4/3})$. This substantially improves the previous algorithms, and answers the open question by Agarwal and Sen.

Moreover, our result is near-optimal! We prove that any algorithm (randomized or deterministic) for these problems must access $\Omega((mn)^{2/3} + m + n)$ entries of the matrix in the worst case. Our lower bound holds even for Monge matrices—most totally monotone matrices arising from applications actually satisfy the stronger *Monge property*: $A[i, j] + A[i', j'] \leq A[i, j'] + A[i', j]$ for all $i < i'$ and $j < j'$.

We can further make the time bound sensitive to K, namely, $\widetilde{O}((mnK)^{1/3} + m + n)$. This K-sensitive bound smoothly interpolates between $\Omega((mn)^{2/3} + m + n)$ and the above-mentioned $\widetilde{O}(m + n + K)$ bound. We prove optimality of this result as well.

Of particular interest is row $(k, \ldots, k)$-selection, which was the case studied by Mansour et al. [32] (and also previously considered by Kravets and Park [30]). For this special case, we get a further improved bound: $\widetilde{O}(m^{2/5}n^{3/5}k^{1/5} + n)$. For example, for square matrices, and in terms of n alone, the bound is $\widetilde{O}(n^{6/5})$. While our earlier $n^{4/3}$ bound might be what an expert would expect, this $n^{6/5}$ bound is more of a surprise. This result may not be optimal though (but further improvement might require a breakthrough in combinatorial geometry).

Applications. Agarwal and Sen [3] mentioned direct applications of the K-selection problem in totally monotone matrices to selecting the K-th smallest geodesic distance for the vertices of a simple polygon (following observations by Hershberger and Suri [27]). Bein et al. [7] described another application of the selection problem to a k-link bottleneck shortest path problem for complete DAGs satisfying a certain "bottleneck Monge property". With our $\widetilde{O}(n^{4/3})$ algorithm, we immediately get improved results for these problems as well.

Connection with pseudo-lines. Totally monotone matrices can be viewed geometrically in terms of arrangements of pseudo-lines: 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 $(1, A[1, j]), (2, A[2, j]), \ldots, (m, A[m, j])$. Total monotonicity implies that these curves are indeed pseudo-lines (assuming no degeneracies), as illustrated in Figure 1. The t-ranking problem corresponds to counting the number of pairs of points (i, t) and pseudo-lines γ_j with the point (i, t) above the pseudo-line γ_j; this reduces to answering n "offline pseudo-halfplane range counting queries" on a set of m points in the plane.

The observation that the columns of a totally monotone matrix may be viewed as pseudo-lines is not new (for example, see [28, 29] and our companion paper [13]), but this observation was not used in the previous selection algorithms by Alon and Azar [6], Mansour et al. [32], or Agarwal and Sen [3]. The geometric perspective allows us to tap into the rich body of techniques from computational geometry, on range searching and arrangements of lines, to attack matrix searching problems.

Some techniques for lines can be generalized to pseudo-lines without much effort. For example, existing algorithms for constructing the arrangement of n lines [19] can be adapted for pseudo-lines, taking $\widetilde{O}(n^2)$ time. Primitive operations such as computing the intersection of two pseudo-lines can be done in logarithmic time by binary search instead of constant time, but since optimizing logarithmic factors is not a main concern in the present paper, this is acceptable. So, it is not difficult to obtain an $\widetilde{O}(n^2+m)$ algorithm for the ranking problem. By a standard trick of dividing the pseudo-lines into $n/\sqrt{m}$ groups of size $\sqrt{m}$, this would already give an algorithm with running time $\widetilde{O}(\lceil n/\sqrt{m}\rceil \cdot m) = \widetilde{O}(n\sqrt{m}+m)$.

However, certain techniques are not generalizable. Even some very simple operations about points and lines are no longer doable. For example, we cannot easily form a pseudo-line through two input points. We cannot determine the orientation of three points, since a "correct" answer would require knowing the relationship of the points with respect to all pseudo-lines, which would be expensive (not $\widetilde{O}(1)$-time doable).

In traditional settings, it is well known that n halfplane or simplex range counting queries on m points in the plane can be answered in $\widetilde{O}((mn)^{2/3}+m+n)$ time [2, 14, 33, 34]. Standard solutions involve a combination of *cutting trees* and *partition trees*. Cuttings can be generalized for pseudo-lines, since they can be constructed from arrangements of random subsets of pseudo-lines. But the simplicial partition trees of Matoušek [33, 34] appear difficult to generalize (even the simpler suboptimal partition trees of Willard [44] are not obviously generalizable). In the offline setting where all the queries are given, partition trees can usually be avoided completely by switching to dual space, interchanging points and lines; interpolating between an $\widetilde{O}(n^2+m)$ solution and its dual $\widetilde{O}(m^2+n)$ solution would then yield the desired result. However, this would require a generalization of duality between points and pseudo-lines.

In SODA'02, Agarwal and Sharir [4] (extending an earlier result of Goodman [26]) actually described such a duality transform between points and pseudo-lines. Their technique would fix many of the issues mentioned above (for example, orientation of three points can be determined by examining the intersections formed by the three dual pseudo-lines). However, Agarwal and Sharir's algorithm for constructing the transform requires efficient data structures for performing certain operations on the pseudo-lines (namely, dynamic pseudo-halfplane range emptiness). For pseudo-lines that are defined by polynomials of constant degree, they showed there is such a data structure with $O(n^{\varepsilon})$ time per operation for an arbitrarily small constant $\varepsilon > 0$. However, such data structures are not available for general pseudo-line families (and pseudo-line families formed by general totally monotone matrices do not have "constant description complexity").[2]

Our new algorithm will not explicitly use duality but will work entirely in primal space. We exploit a simple observation that faces in the dual arrangement roughly correspond to equivalence classes of points in the primal (the correspondence isn't perfect but is good enough for our purposes). The existence of a duality transform is needed in the analysis of our algorithm, but not in

[2] If the matrix is Monge, some form of duality is possible, as observed in the companion paper [13], but in the present paper, our goal is in obtaining results for general totally monotone matrices.

the algorithm itself. With this approach (and some standard data structures for dynamic lower envelopes), we obtain a (relatively simple) randomized incremental algorithm with $\widetilde{O}(m^2+n)$ running time. Combined with standard cuttings, we then get the desired $\widetilde{O}((mn)^{2/3}+m+n)$ result.

As byproduct, we obtain an algorithm with the same time bound for computing incidences between points and pseudo-lines in the plane. One of the key applications of Agarwal and Sharir's duality transform [4] was in solving this incidence problem, a so-called "Hopcroft's problem" for pseudo-lines. Our new algorithm for incidence is thus (i) slightly faster (replacing n^{ε} with polylogarithmic factors in their time bound of $O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon}+m^{1+\varepsilon}+n^{1+\varepsilon})$), (ii) more general (working for any pseudo-line family, not just pseudo-lines defined by constant-degree polynomial arcs), and most importantly, (iii) simpler (not just avoiding the $O(n^{\varepsilon})$-time data structures but also bypassing the explicit construction of the duality transform). We think this new algorithmic result is of independent interest to computational geometers, regardless of the connection to matrix searching. (Agarwal and Sharir's algorithm for the duality transform is still a powerful result, but our point is that it is not necessary for solving the incidence problems.)

The problems addressed here are also related to a line of research started in the late 1990s on solving computational geometry problems *with restricted predicates* [8, 10, 31, 37]. Motivated by precision issues and cost of exact arithmetic, the goal was to understand to what extent higher-degree predicates could be avoided in solving geometric problems. Our work here shows that Hopcroft's problem and its relatives can be solved with very limited primitive operations or predicates (just deciding whether an input point is above a pseudo-line, and whether one pseudo-line is above another pseudo-line at the x-coordinate of an input point); the time complexity is unaffected by the predicate restriction (ignoring logarithmic factors). It should not be taken for granted that the same time bound is always achievable; for example, Boissonnat and Snoeyink [8] showed that the complexity of the pseudo-line segment intersection problem changes drastically when restricted to a certain natural set of predicates.

Our lower bound for selection in totally monotone or Monge matrices is also obtained using the above geometric perspective, and turns out to be a simple consequence of standard incidence bounds from combinatorial geometry (the Szemerédi–Trotter theorem) [35, 42]. Lower bounds on Hopcroft's problem or offline range searching are difficult to prove in general models of computation; for example, see Erickson's work [22] on lower bounds in a restricted model of so-called "partitioning algorithms". Our lower bound is easier to prove due to the more abstract setting, but it is still nice to see a super-linear, and near-tight, lower bound for a basic problem in a natural setting.

Finally, our unusual $\widetilde{O}(n^{6/5})$ result on the row $(k,\ldots,k)$-selection problem is obtained by combining the $\widetilde{O}(m^2+n)$ algorithm with combinatorial results on the well-known *k-level* or *k-set* problem [18, 35]. The result is new even for the case of lines (for example, we can determine which of n given points are above the k-level of n lines in 2D in $\widetilde{O}(n^{6/5})$ time). Any improvement on the k-set problem could potentially lead to improvement in our bound.

2 Ranking

We focus on the row $(t_1,\ldots,t_m)$-ranking problem (which includes the t-ranking problem) for an $m\times n$ totally monotone matrix A. Solution to the selection problems will follow once we have solved the ranking problem.

In geometric terms, the problem is equivalent to the following, which is a form of *offline pseudo-halfplane range counting*:

Given a set P of m points and a set L of n pseudo-lines in the plane, count the number of pseudo-lines below p for each point $p \in P$.

Let X be the set of all x-coordinates of P. Define the *pseudo-slope* of a pseudo-line to be the rank of its y-value at $x = \infty$ among the n pseudo-lines of L. The only allowed primitive operations are testing whether a point $p \in P$ is above a pseudo-line $\ell \in L$, and testing whether a pseudo-line $\ell \in L$ is above another pseudo-line $\ell' \in L$ at the x-coordinate of X (which correspond to comparisons of the form $A[i,j] \le A[i,j']$ or $A[i,j] \le t_i$). For simplicity, we assume no degeneracies (no point incident on a pseudo-line). It is straightforward to modify our algorithm in degenerate cases (or apply symbolic perturbation to avoid such cases).

It is useful to imagine altering each pseudo-line ℓ in the following manner: between two consecutive x-values $x_j, x_{j+1} \in X$, make ℓ stay horizontal from $x = x_j$ to $x = x_{j+1} - \delta i$, and from $x = x_{j+1} - \delta i$ to $x = x_{j+1}$, with a vertical jump at $x = x_{j+1} - \delta i$, where i is the pseudo-slope of ℓ and $\delta > 0$ is an infinitesimal. The alteration allows us to explicitly compute the intersection of two pseudo-lines in $O(\log m)$ time without needing to change the required primitive operations (since we can find the two consecutive x-values between which the intersection is located, by binary search, and then the intersection point itself can be determined from the pseudo-slopes of the two pseudo-lines).

We first review known data structures for dynamic lower (or upper) envelopes that we will use. (These are easier than the dynamic pseudo-halfplane range emptiness data structures needed by Agarwal and Sharir [4]—the data structures below work for general pseudo-lines.)

Lemma 2.1. *Let X be a set of m x-values and L be a set of n pseudo-lines. There is a dynamic data structure for maintaining a subset $C \subseteq L$ of pseudo-lines, supporting the following operations:*

- *insert a pseudo-line to C;*
- *delete a pseudo-line from C;*
- *given a query value $x \in X$, find the lowest (or highest) pseudo-line of C at x-coordinate x.*
- *given a query value $x \in X$, find the leftmost vertex of the lower (or upper) envelope of C with x-coordinate $> x$.*

Each insertion or deletion takes $O(\log^2 n \log m)$ time, and each query takes $O(\log n)$ time.

Proof. (Sketch) This follows from a dualized version of Overmars and van Leeuwen's *hull tree* structure [40]. (Agarwal et al. [1] described the dualized method specifically for pseudo-lines, but the solution is simpler in our setting, where we know all pseudo-lines and points in advance and we don't care about extra logarithmic factors.) For the sake of completeness, we include a quick sketch:

Let $\mathrm{LE}(C)$ denote the lower envelope of C. (We can handle the upper envelope similarly.) For each dyadic interval I, let C_I be the subset of all pseudo-lines of C with pseudo-slope in I. Divide I into two dyadic subintervals I_1 and I_2 (with I_1 left of I_2). of half the length. Store the intersection point z_I of $\mathrm{LE}(C_{I_1})$ and $\mathrm{LE}(C_{I_2})$. (The intersection point is unique since the pseudo-lines in C_{I_1} have smaller pseudo-slopes than C_{I_2}.)

For the first type of query, if the given value x is less than the x-coordinate of z_I, then we recursively find the lowest pseudo-line of C_{I_2} at the x-coordinate x; otherwise, we recursively find the lowest pseudo-line of C_{I_1} at x. The query time is $O(\log n)$.

For the second type of query, if the given value x is less than the x-coordinate of z_I, then we recursively query in C_{I_2}, but if the returned answer is to the right of z_I, return z_I instead; otherwise, we recursively query in C_{I_1}.

To insert/delete a pseudo-line ℓ in C_I: if the pseudo-slope of ℓ is in I_2, we recursively insert/delete ℓ in C_{I_2}; otherwise, we recursively insert/delete ℓ in C_{I_1}. We then recompute z_I by binary search over X, using $O(\log m)$ calls to the query algorithm, in $O(\log n \log m)$ time. The overall update cost is $O(\log^2 n \log m)$. □

Corollary 2.2. *Let X be a set of m x-values and L be a set of n pseudo-lines. There is a dynamic data structure for maintaining a collection $\mathcal{C}$ of disjoint subsets of L, supporting the following operations:*

- *given a query value $x \in X$, report the lowest (or highest) pseudo-line of C at x-coordinate x;*
- *given a point q, split a subset $C \in \mathcal{C}$ into two subsets $C_{\prec q} = \{\ell \in C : \ell \text{ is below } q\}$ and $C_{\succ q} = \{\ell \in C : \ell \text{ is above } q\}$.*

The total time over all splits is $O(n \log^3 n \log m)$, and each query takes $O(\log m)$ time.

Proof. We use a standard amortization trick, of splitting a set by repeatedly deleting elements of the smaller from the larger set.

To split C for a given point q, first initialize $k = 1$. Find the k lowest and highest pseudo-lines of C at the x-coordinate of q, by making $O(k)$ queries and deletions (and re-insertions). If q is below the k-th lowest pseudo-line, then move the pseudo-lines below q to a new subset $C_{\prec q}$, by $O(k)$ deletions and insertions. Otherwise, if q is above the k-th highest pseudo-line, move the pseudo-lines above q to a new subset $C_{\succ q}$, by $O(k)$ deletions and insertions. Otherwise, double k and repeat. This way, the split is accomplished by making a total of $O(\min\{|C_{\prec q}|, |C_{\succ q}|\})$ queries, insertions, and deletions. We can charge each query/insertion/deletion operation to a member of the smaller of the two sets $C_{\prec q}$ or $C_{\succ q}$. Each time a pseudo-line ℓ is charged, the subset containing ℓ decreases by at least a factor of 2. Thus, the total number of queries, insertions, and deletions over a sequence of splits is $O(n \log n)$, which has total cost $O(n \log^3 n \log m)$ by Lemma 2.1. □

2.1 Warm-up: $\widetilde{O}(m^3 + n)$ algorithm

We are now ready to describe a simple *incremental* algorithm to solve the row ranking problem.

We say that a point q is *completely above* (resp. *completely below*) a set C of pseudo-lines if q is above (resp. below) all pseudo-lines in C. We say that q *conflicts with* C if q is neither completely above nor completely below C. For two pseudo-lines $\ell, \ell' \in L$, we say that ℓ and ℓ' are *equivalent* w.r.t. a point set Q iff $\{q \in Q : \ell \text{ is above } q\}$ and $\{q \in Q : \ell' \text{ is above } q\}$ are identical. (See Figure 2(a).)

The idea is to incrementally maintain the equivalence classes of L as we insert points one at a time. More precisely, let $P = \{p_1, \ldots, p_n\}$. In the i-th iteration, we maintain the collection $\mathcal{C}$ of all equivalence classes of L w.r.t. $\{p_1, \ldots, p_i\}$, which are stored in the data structure from Corollary 2.2. The counts can be computed along the way.

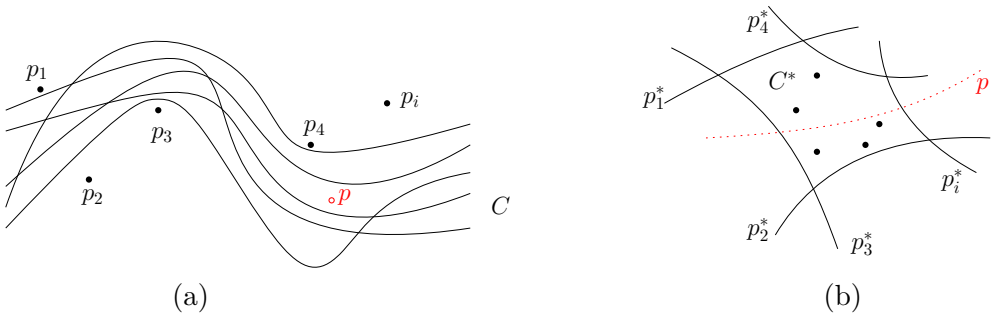

Figure 2: (a) An equivalence class C of pseudo-lines w.r.t. $\{p_1, \ldots, p_i\}$, and a point p in conflict with C. (b) A cell in the dual arrangement in the zone of p corresponding to C.

Pseudocode.

0. let $p_1, \ldots, p_m$ be the points of P (in any order), and initialize $\mathcal{C} = \{L\}$
1. for $i = 1$ to m do
2. for each $C \in \mathcal{C}$ do
3. if p_i conflicts with C then split C into $C_{\prec p_i}$ and $C_{\succ p_i}$ in $\mathcal{C}$
4. for each $C \in \mathcal{C}$ do
5. if p_i is completely above C then add $|C|$ to COUNT$[p_i]$

Correctness of the algorithm is easy to see.

Running time. To analyze the running time of the algorithm, we need to bound the number of equivalence classes. To this end, we use Agarwal and Sharir's duality transform [4]: they showed that given a set P of points and a set L of pseudo-lines, there exists a mapping of points $p \in P$ to pseudo-lines p^*, and of pseudo-lines $\ell \in L$ to points ℓ^*, such that p is above ℓ iff p^* is above ℓ^*.

Let $\mathcal{A}_i$ be the arrangement of the dual pseudo-lines $p_1^*, \ldots, p_i^*$. Observe that each equivalence class C w.r.t. $\{p_1, \ldots, p_i\}$ maps to a face in the dual arrangement $\mathcal{A}_i$: if the points ℓ^* and $(\ell')^*$ are in the same face of $\mathcal{A}_i$, then $\{p^* \in \{p_1^*, \ldots, p_i^*\} : p^* \text{ is below } \ell^*\}$ and $\{p^* \in \{p_1^*, \ldots, p_i^*\} : p^* \text{ is below } (\ell')^*\}$ are identical, i.e., ℓ and ℓ' are equivalent w.r.t. $\{p_1, \ldots, p_i\}$. On the other hand, if the points ℓ^* and $(\ell')^*$ are in different faces of $\mathcal{A}_i$, then there is a pseudo-line $p^* \in \{p_1^*, \ldots, p_i^*\}$ separating the two points, so $\{p^* \in \{p_1^*, \ldots, p_i^*\} : p^* \text{ is below } \ell^*\}$ and $\{p^* \in \{p_1^*, \ldots, p_i^*\} : p^* \text{ is below } (\ell')^*\}$ are not identical, i.e., ℓ and ℓ' are not equivalent w.r.t. $\{p_1, \ldots, p_i\}$.

Any arrangement of i pseudo-lines has $O(i^2)$ faces, and so $\mathcal{A}_i$ has $O(i^2)$ faces, and so the number of equivalence classes at the i-th iteration is $O(i^2)$. The tests in lines 3 and 5 require a query (at the x-coordinate of p_i) in the data structure from Corollary 2.2. The entire algorithm makes $O(\sum_{i=1}^m i^2) = O(m^3)$ queries, costing $\widetilde{O}(m^3)$ time. The total cost of the split operations is $\widetilde{O}(n)$. The total time bound is $\widetilde{O}(m^3 + n)$.

Remarks. The mapping of equivalence classes in primal space to faces in dual space is not necessarily surjective, as some faces in the dual arrangement may be empty of points. Explicitly generating all faces seems to require Agarwal and Sharir's duality transform algorithm (which works only for restricted families of pseudo-lines). But the above demonstrates that working with

equivalence classes instead of faces is sufficient for our problem, although the resulting near-cubic algorithm is slower than the standard quadratic algorithm for constructing the faces in an arrangement of lines [19]. We will remedy the situation and improve the running time to near-quadratic in the next subsection.

2.2 $\widetilde{O}(m^2 + n)$ algorithm

We next improve the running time by using a *randomized* incremental approach, well-known in computational geometry [17, 16, 38]—in other words, we insert points in a random order. Randomization was not needed in the standard incremental algorithm for constructing an arrangement of lines [19], but will be crucial here.

As in the previous subsection, we maintain the equivalence classes $\mathcal{C}$ as we insert points. The bottleneck in the previous algorithm is in the loop in lines 2–3 for finding all equivalence classes that a point conflicts with (there may be $O(i^2)$ classes in total, but only $O(i)$ classes may conflict with a point). Our approach is to explicitly maintain for each point $p \in P$ a list of all classes $C \in \mathcal{C}$ that p conflicts with—the resulting bipartite graph between points and classes is called the *conflict graph*. (Similar conflict graphs were used in the original randomized incremental algorithms by Clarkson and Shor [16] for many geometric problems.)

Pseudocode.

0. let $p_1, \ldots, p_m$ be the points of P in a *random* order, and initialize $\mathcal{C} = \{L\}$
1. for $i = 1$ to m do
2. for each $C \in \mathcal{C}$ that p_i conflicts with, do
3. split C into $C_{\prec p_i}$ and $C_{\succ p_i}$ in $\mathcal{C}$
4. for each $p \in P$ that conflicts with C, and for each $C' \in \{C_{\prec p_i}, C_{\succ p_i}\}$ do
5. test whether p conflicts with C'
6. if p is completely above C' then add $|C'|$ to COUNT$[p]$

In line 2, the conflict graph enables us to loop through all classes that p_i conflicts with, in time linear in the number of such classes (using linked lists). However, after a class C is split in line 3, the conflict graph needs to be updated, which is the purpose of the loop in lines 4–5.

At the end, COUNT$[p]$ gives us the correct answer, since a pseudo-line below p will eventually belong to a class completely below p and will be counted the first time this happens.

Expected running time. Let $\mu_i^-(p, C)$ be 1 if p conflicts with C and the class C is destroyed (i.e., gets split) in the i-th iteration, and 0 otherwise. Let $\mu_i^+(p, C)$ be 1 if p conflicts with C and the class C is created in the i-th iteration, and 0 otherwise.

Lines 5–6 require a query in the data structure from Corollary 2.2. The number of queries in the i-th iteration is $O(\sum_{p\in P} \sum_C \mu_i^-(p, C))$. The total is $O(\sum_{i=1}^m \sum_{p\in P} \sum_C \mu_i^-(p, C))$. Clearly, $\sum_{i=1}^m \mu_i^-(p, C) \le \sum_{i=1}^m \mu_i^+(p, C)$, since a class destroyed must have been created in some earlier iteration. Thus, the total number of queries is at most $O(\sum_{i=1}^m \sum_{p\in P} \sum_C \mu_i^+(p, C))$.

We use backwards analysis [38, 41] to bound $\mathbb{E}[\sum_C \mu_i^+(p, C)]$ for each fixed i and each fixed $p \in P$, as follows:

As in the earlier analysis, we use Agarwal and Sharir's duality transform [4]. Let $\mathcal{A}_i$ be the arrangement of the dual pseudo-lines $p_1^*, \ldots, p_i^*$. Let $C^* = \{\ell^* : \ell \in C\}$. If p conflicts with a class

C at the end of the i-th iteration, then the pseudo-line p^* is above some point in C^* and below some point in C^*, and so it must intersect the face of $\mathcal{A}_i$ containing C^* (the converse may not be true but is not important). Furthermore, if C is created in the i-th iteration, then p_i^* must appear on the boundary of the face of $\mathcal{A}_i$ containing C^*.

Thus, if $\mu_i^+(p, C)$ is true, then p^* intersects the face of $\mathcal{A}_i$ containing C^*, and p_i^* appears on the boundary of this face. (See Figure 2(b).)

So, $\sum_C \mu_i^+(p, C)$ is upper-bounded by the number of appearances of p_i^* on the boundaries of the faces of $\mathcal{A}_i$ intersected by the pseudo-line p^*, i.e., the number of appearances of p_i^* in the *zone* of the pseudo-line p^* in $\mathcal{A}_i$. By the Zone Theorem [20], the sum of the number of edges in the faces of the zone is $O(i)$. Conditioned to a fixed subset $\{p_1^*, \ldots, p_i^*\}$ (and thus a fixed $\mathcal{A}_i^*$), we can think of p_i^* as a random element that is equally likely to be any element in this subset, with probability $1/i$. It follows that $\mathbb{E}[\sum_C \mu_i^+(p, C)] = O(i/i) = O(1)$. Since this expectation bound is independent of the subset, it holds unconditionally.

Therefore, the entire algorithm makes $O(\mathbb{E}[\sum_{i=1}^m \sum_{p \in P} \sum_C \mu_i^+(p, C)]) = O(m^2)$ queries, costing $\widetilde{O}(m^2)$ time, in expectation. The total cost of the split operations is $\widetilde{O}(n)$. The total expected time bound is $\widetilde{O}(m^2 + n)$.

Remarks. As in other randomized incremental algorithms, the conflict graph may alternatively be replaced by a *history dag* (or, in this case, a "history tree") [38].

There are similarities with a randomized algorithm by Mulmuley and Sen [39], although their goal was in developing dynamic point location data structures in hyperplane arrangements for random update sequences. It is also interesting to compare the above algorithm with the standard (non-randomized) incremental algorithm for constructing an arrangement of (pseudo-)lines [19]. To find all the faces intersected by the current line, the standard algorithm navigates from faces to adjacent faces, viewing the arrangement as a planar graph, which we cannot do without explicitly constructing the duality map (faces that are empty of points are essential for navigation). In contrast, our algorithm basically navigates from parent faces to child faces in the history tree. The analysis of the standard algorithm uses the Zone Theorem in a straightforward way; the randomized analysis of our algorithm uses the Zone Theorem in a subtler way.

Note that by dividing points into groups of size $\sqrt{n}$, the above result automatically implies an algorithm with expected running time $\widetilde{O}(\lceil m/\sqrt{n} \rceil \cdot n) = \widetilde{O}(m\sqrt{n} + n)$. This bound is already new, but we will improve it further.

2.3 $\widetilde{O}((mn)^{2/3} + m + n)$ algorithm

For the final algorithm, we combine the $\widetilde{O}(m^2+n)$ algorithm with the standard geometric technique of *cuttings*, which are generalizable to pseudo-lines without any extra effort. Below, a *cell* refers to a "pseudo-trapezoid", which has two vertical sides, and a top and bottom side that are parts of the input pseudo-lines. We assume that the intersection of two pseudo-lines can be computed in $\widetilde{O}(1)$ time, which is true after altering the pseudo-lines as described previously.

Lemma 2.3. (Cutting Lemma) *Let L be a set of n pseudo-lines, Δ_0 be an initial cell, and $a \leq n$. We can cut Δ_0 into $O(a^2)$ cells each intersecting $O(\frac{n}{a} \log a)$ pseudo-lines of L.*

In fact, the expected number of cells is bounded by $O(a + \nu_{\Delta_0}(\frac{a}{n})^2)$, where ν_{Δ_0} denotes the number of intersections in Δ_0 among the pseudo-lines of L. The construction takes $\widetilde{O}(na^{O(1)})$ expected time.

Proof. (Sketch) With randomization, a standard, simple way to construct cuttings [15, 38] is to just take a random sample R of size a, and construct the vertical decomposition of the arrangement formed by R. This construction immediately generalizes to pseudo-lines. (We cannot use the canonical triangulation, but the vertical decomposition is fine, and yields pseudo-trapezoidal cells.) □

Corollary 2.4. *Let L be a set of n pseudo-lines, and $r \le n$. We can cut the plane into $O(r^2)$ cells, each intersecting at most n/r pseudo-lines of L. Furthermore, we can compute the cell, the list L_Δ of all pseudo-lines of L intersecting each cell Δ, and the number $\textsc{count}_\Delta$ of pseudo-lines of L below each cell Δ, in total expected time $\widetilde{O}(nr)$. Given a set P of m points, we can compute $P \cap \Delta$ for all cells Δ, in $O(m \log r)$ additional time.*

Proof. (Sketch) This follows from Chazelle's *hierarchical cutting* method [14]. The description can be simplified with randomization, and for the sake of completeness, we include a quick sketch:

Let b be a sufficiently large constant. Construct a constant-degree tree of cells, where the root cell is the entire plane. We maintain the invariant that each cell Δ at level j intersects $|L_\Delta| \le n/b^j$ pseudo-lines of L. By applying the Cutting Lemma with $a = (c_0 b \log b)\frac{|L_\Delta|}{n/b^j}$ for a sufficiently large constant c_0, we can subdivide a given cell Δ at level j into an expected $O(a + \nu_\Delta(\frac{a}{|L_\Delta|})^2) = O(b \log b + (b^{2j+2} \log^2 b)\frac{\nu_\Delta}{n^2})$ number of child cells, each intersecting at most $c_0'\frac{|L_\Delta|}{a} \log a \le n/b^{j+1}$ pseudo-lines for some constant c_0'. Let t_j be the number of cells at level j. Then $t_{j+1} \le O(b \log b)t_j + O(b^{2j+2} \log^2 b)$, implying that $t_{j+1} = O(b^{2j+2} \log b) = O(b^{2j})$ for a sufficiently large constant b. We output the cells at level $\ell = \lceil \log_b r \rceil$. The total number of cells is thus $O(r^2)$, and each intersects at most n/r pseudo-lines.

For each child cell Δ' of Δ, we can compute $L_{\Delta'}$ and $\textsc{count}_{\Delta'}$ from L_Δ and $\textsc{count}_\Delta$ in $O(|L_\Delta|) = O(n/b^j)$ time. The total expected time is $\widetilde{O}(\sum_{j \le \ell} b^{2j} \cdot n/b^j) = \widetilde{O}(nr)$. The leaf cell containing a given point p can be located in $O(\log r)$ time by following a path in the tree. □

We now apply the above corollary. Further subdivide each cell so that each cell contains at most $\lceil m/r^2 \rceil$ points of P; the number of cells needed remains $O(r^2)$. For each cell Δ, solve the subproblem for the $O(m/r^2)$ points of $P \cap \Delta$ and the $O(n/r)$ pseudo-lines of L_Δ, by using the algorithm in Section 2.2. Afterwards, for each $p \in P_\Delta$, we add $\textsc{count}_\Delta$ to the current count for p.

The total expected running time is

$$\widetilde{O}(r^2 \cdot ((m/r^2)^2 + n/r) + nr + m) \;=\; \widetilde{O}(m^2/r^2 + nr + m).$$

Setting $r = \lceil m^{2/3}/n^{1/3} \rceil$ gives $\widetilde{O}((mn)^{2/3} + m + n)$, assuming that $m \le n^2$. For $m > n^2$, we can divide into groups of n^2 points and solve the problem for each group in $\widetilde{O}(n^2)$ time, yielding $\widetilde{O}(m)$ total time.

Theorem 2.5. *Given an $m \times n$ totally monotone matrix and values $t_1, \ldots, t_m$, we can count the number of elements at most t_i in the i-th row, for all $i = 1, \ldots, m$, in $\widetilde{O}((mn)^{2/3} + m + n)$ expected time.*

Remarks. The number of hidden logarithmic factors is small (in the low single-digit), but we have not attempted to optimize it and so will left it unspecified.

It is straightforward to modify our algorithms to compute all incidences between n points and m pseudo-lines in $\widetilde{O}((mn)^{2/3} + m + n)$ expected time, using the same primitive operations. We

just have to pay more careful attention to degeneracies. (For example, in defining equivalence of two pseudo-lines ℓ and ℓ' w.r.t. Q, we should additionally insist that $\{q \in Q : q \text{ is on } \ell\}$ and $\{q \in Q : q \text{ is on } \ell'\}$ are identical.)

Cuttings can be constructed deterministically [14], but the toughest part to derandomize is the $\widetilde{O}(m^2 + n)$ randomized incremental algorithm. (In the Monge case, we might be able to avoid this part, however, by duality or symmetry [13].) The more straightforward $\widetilde{O}(m^3 + n)$ algorithm is already deterministic, so it should be possible to obtain some improved (but not necessarily near-optimal) deterministic algorithms with our approach.

3 Consequences

3.1 K-sensitive running time

We can make the running time of our row $(t_1, \ldots, t_m)$-ranking algorithm sensitive to K, the sum of the row ranks of the t_i's, i.e., the sum of the levels of the points in P. Here, the *level* of a point p is the number of pseudo-lines of L below p.

Assume that an upper bound K_0 of K is given. Let $k_* = \frac{K_0}{m}$. Take a random sample R of size $\frac{n}{4k_*}$ and compute the lower envelope $\mathrm{LE}(R)$ and its vertical decomposition $\mathrm{VD}(R)$. The decomposition has $O(\frac{n}{k_*})$ cells, and each cell intersects $\widetilde{O}(k_*)$ pseudo-lines of L w.h.p. (by Clarkson and Shor's analysis [16, 38]). For each cell $\Delta \in \mathrm{VD}(R)$, let L_Δ be its conflict list (the subset of all pseudo-lines of L intersecting Δ). To compute the conflict lists, for each pseudo-line ℓ, we find a vertex of $\mathrm{LE}(R)$ that is above ℓ, by binary search, and then find all cells $\Delta \in \mathrm{VD}(R)$ intersected by ℓ, by a linear search in both directions starting at that initial vertex. The total running time is $\widetilde{O}(n + \sum_\Delta |L_\Delta|) = \widetilde{O}(n + (\frac{n}{k_*})k_*) = \widetilde{O}(n)$ w.h.p. We further subdivide the cells so that each cell contains at most $m/(\frac{n}{k_*})$ points; the number of cells remains $O(\frac{n}{k_*})$. For each cell Δ, we solve the subproblem for $P \cap \Delta$ and L_Δ by the algorithm from the previous section. The total time is

$$\widetilde{O}((\tfrac{n}{k_*}) \cdot ((m/(\tfrac{n}{k_*})) \cdot k_*)^{2/3} + k_* + m/(\tfrac{n}{k_*})) \;=\; \widetilde{O}(n^{1/3}m^{2/3}k_*^{1/3} + m + n) \;=\; \widetilde{O}((mnK_0)^{1/3} + m + n).$$

The above handles all points of P that are below $\mathrm{LE}(R)$. We repeat the process for all remaining points above $\mathrm{LE}(R)$. Observe that at least half of the points of P have level at most $2k_*$. For each point with level at most $2k_*$, the probability that it is above $\mathrm{LE}(R)$ is at most $2k_* \cdot n/(4k_*) = 1/4$. Thus, the expected number of points of P below $\mathrm{LE}(R)$ is at least $n/4$. Consequently, the process stops after an $O(\log n)$ expected number of iterations. Therefore, the expected running time of the algorithm is $\widetilde{O}((mnK_0)^{1/3} + m + n)$; by repeating logarithmically many times, the time bound holds w.h.p. To finish, we try $K_0 = 1, 2, 4, 8, \ldots$ until the algorithm successfully runs to completion (which is true w.h.p. when $K_0 \geq K$). The total time bound remains $\widetilde{O}((mnK)^{1/3} + m + n)$.

Theorem 3.1. *Given an $m \times n$ totally monotone matrix and values $t_1, \ldots, t_m$, we can count the number of elements at most t_i in the i-th row, for all $i = 1, \ldots, m$, in $\widetilde{O}((mnK)^{1/3}+m+n)$ expected time, where K is the sum of the counts.*

Remark. A similar K-sensitive bound, $\widetilde{O}(n^{2/3}K^{1/3} + n)$, was known before for the problem of selecting the K-th smallest distance (under the Euclidean metric) for a set of n points in the plane [11].

3.2 Column $(t, \ldots, t)$-ranking

Our algorithms can be modified to solve the *column* $(t, \ldots, t)$*-ranking* problem: counting the number of elements at most a given value t, in each column. In geometric terms, the problem reduces to counting the the number of points of P above ℓ for each pseudo-line ℓ—this is offline pseudo-halfplane range counting. We briefly describe the changes:

In the $\widetilde{O}(m+n^2)$ algorithm, for each equivalence class C, we maintain the number $\textsc{count}[C]$ of points completely above C. In line 3, we initialize $\textsc{count}[C_{\prec p_i}]$ and $\textsc{count}[C_{\succ p_i}]$ to $\textsc{count}[C]$. In line 6, if p is completely above C', we increment $\textsc{count}[C']$. At the end, the count for a pseudo-line ℓ is the count for ℓ's equivalence class.

In the $\widetilde{O}((mn)^{2/3}+m+n)$ algorithm: In Corollary 2.4, we also compute the number $\textsc{count}[\ell]$ of points of P in the cells completely above ℓ. To do so, if Δ' is a child of Δ and $\ell \in L_\Delta$ is completely below Δ', we add $|P \cap \Delta'|$ to $\textsc{count}[\ell]$.

In the K-sensitive algorithm, the changes are straightforward.

Theorem 3.2. *Given an $m \times n$ totally monotone matrix and a value t, we can count the number of elements at most t in the j-th column, for all $j = 1, \ldots, n$, in $\widetilde{O}((mnK)^{1/3} + m + n)$ expected time, where K is the sum of the counts.*

3.3 Row successors

In the *row successor* problem, we are given values $t_1, \ldots, t_m$, and want to find the smallest element greater than t_i in the i-th row, for all $i = 1, \ldots, m$. In geometric terms, the problem corresponds to *offline vertical ray shooting*: for each $p \in P$, find the first pseudo-line of L hit by a vertical upward ray from p. We observe that our algorithms can be modified to solve this problem:

In the $\widetilde{O}(m + n^2)$ algorithm, in line 6, if p is completely below C', we can find the lowest pseudo-line ℓ of C' at the x-coordinate of p by querying the data structure in Corollary 2.2; if ℓ is lower than the current answer for p, reset the current answer for p to ℓ.

In the $\widetilde{O}((mn)^{2/3} + m + n)$ algorithm, no major changes are required (since the answer for p may be found in the cell containing p). The K-sensitive algorithm also requires no major changes.

3.4 Row $(k_1, \ldots, k_m)$-selection

There are several ways to adapt our ranking algorithm to solve the row $(k_1, \ldots, k_m)$-selection problem.[3] If we don't mind some extra logarithmic factors, there is a general randomized reduction from row $(k_1, \ldots, k_m)$-selection to row ranking and row successors (this reduction does not require geometry, and is based on standard sampling ideas, though this particular variant might be new):

Choose a hierarchy of random samples $R_1 \subset R_2 \subset \cdots \subset R_\ell = \{1, \ldots, n\}$ with $\ell = \log n$, where R_j has size 2^j. Let A_j be the submatrix formed by the columns in R_j.

Consider a fixed $j \in \{1, \ldots, \ell\}$. For each i with $k_i \in [\frac{n}{2^j}, \frac{n}{2^{j-1}})$, first compute the $(c \log n)$-th smallest element $\bar{t}_i$ in the i-th row of A_j, for a sufficiently large constant c—we do this simultaneously for all i, for example, by $O(\log n)$ calls to the row successors algorithm for A_j. By a Chernoff bound, the answer for the i-th row (i.e., the k_i-th smallest in the i-th row of A) is upper-bounded by $\bar{t}_i$, and furthermore, the rank of $\bar{t}_i$ in the i-th row of A is at most $\widetilde{O}(k_i)$.

[3] It is tempting to try parametric search [36], but one issue is that an efficient parallelization of the ranking algorithm is required (so our randomized incremental algorithm would need change). Another issue is that the row selection problem is seeking multiple values, not one...

We now proceed in ℓ rounds and maintain a value t_i for each row i. Before round j, we assume that the answer for the i-th row lies between t_i and the successor of t_i in the i-th row of A_{j-1}. Then w.h.p., the number of elements in the i-th row of A_j between these two values is $O(\log n)$. We start with t_i and generate the $O(\log n)$ successors of t_i, but not exceeding the upper bound $\bar{t}_i$, in the i-th row of A_j—we do this simultaneously for all i, by $O(\log n)$ calls to the row successors algorithm for A_j. We compute the ranks of these elements in the i-th row of A—again we do this simultaneously for all i, by $O(\log n)$ calls to the row ranking algorithm for A. (Because of the upper bound $\bar{t}_i$, the total rank in each call is $\widetilde{O}(K)$.) As a result, we know where the answer lies among these $O(\log n)$ candidates, and can then update t_i so that the answer for the i-th row lies between t_i and the successor of t_i in the i-th row of A_j.

At the end of the ℓ rounds, we know the answer for every row i. In total, we have made $O(\log^2 n)$ calls to the row successors algorithm and the row ranking algorithm.

Theorem 3.3. *Given an $m \times n$ totally monotone matrix and numbers $k_1, \ldots, k_m$, we can find the k_i-th smallest element in the i-th row, for all $i = 1, \ldots, m$, in $\widetilde{O}((mnK)^{1/3} + m + n)$ expected time, where $K = \sum_i k_i$.*

3.5 K-selection

There is a similar general randomized reduction from K-selection to row ranking and row successors:

Choose a hierarchy of random samples $R_1 \subset R_2 \subset \cdots \subset R_\ell = \{1, \ldots, n\}$ with $\ell = \log n$, where R_j has size 2^j. Let A_j be the submatrix formed by the columns in R_j.

First compute the $(\frac{cK \log n}{m})$-th smallest element $\bar{t}$ in a random subset of n elements in the matrix, in $O(n)$ time. By a Chernoff bound, the answer (i.e., the K-th smallest in A) is upper-bounded by $\bar{t}$, and furthermore, the rank of $\bar{t}$ in the i-th row of A is at most $\widetilde{O}(K)$.

We now proceed in ℓ rounds and maintain a value t_i for each row i. Before round j, we assume that the answer lies between t_i and the successor of t_i in the i-th row of A_{j-1}. Then w.h.p., the number of elements in the i-th row of A_j between these two values is $O(\log n)$. We start with t_i and generate the $O(\log n)$ successors of t_i, but not exceeding the upper bound $\bar{t}$, in the i-th row of A_j—we do this simultaneously for all i, by $O(\log n)$ calls to the row successors algorithm for A_j. Relative to all these $O(m \log n)$ elements, we determine where the answer lies; this can be done by binary search with $O(\log(mn))$ calls to the ranking algorithm. (Because of the upper bound $\bar{t}$, the rank in each call is $\widetilde{O}(K)$.) We can then update t_i so that the answer lies between t_i and the successor of t_i in A_j.

At the end of the ℓ rounds, we know the answer for every row i. In total, we have made $O(\log^2 n)$ calls to the row successors algorithm and the row ranking algorithm.

Theorem 3.4. *Given an $m \times n$ totally monotone matrix and a number K, we can find the K-th smallest element in $\widetilde{O}((mnK)^{1/3} + m + n)$ expected time.*

3.6 Row $(k, \ldots, k)$-selection

We now describe a better time bound for row $(k_1, \ldots, k_m)$-selection in the case when $k_1 = \cdots = k_m = k$.

In geometric terms, the problem corresponds to the following:

> Given a set X of m values and a set L of n pseudo-lines in the plane, determine the k-th lowest pseudo-line at each x-coordinate $x \in X$.

In other words, we want to find the intersection of the k-level with the vertical lines at the x-coordinates of X, where the *k-level* consists of all points that are on one pseudo-line and above exactly $k-1$ pseudo-lines.

We need two known facts about k-levels. The first is an efficient output-sensitive algorithm. Here, we assume that the intersection of two pseudo-lines can be computed in $\widetilde{O}(1)$ time, which is true after altering the pseudo-lines as described before.

Lemma 3.5. *The k-level of a set L of n pseudo-lines can be constructed in $\widetilde{O}(n+h)$ time, where h is the number of vertices of the k-level.*

Proof. This follows by adapting a known algorithm by Edelsbrunner and Welzl [21], which reduces the k-level construction problem to dynamic lower/upper envelopes. We sweep the plane from left to right by a vertical line $x = x_m$. At any time, we maintain the k-th lowest pseudo-line ℓ of L at $x = x_m$, the subset L^- of the k lowest pseudo-lines of L at $x = x_m$, and the subset L^+ be the subset of the $n-k+1$ highest pseudo-lines of L at $x = x_m$ (the only element in both L^- and L^+ is ℓ). We store L^- and L^+ in the dynamic data structure in Lemma 2.1. In each iteration, we find the next vertex of the upper envelope of L^- to the right of $x = x_m$ and the next vertex of the lower envelope of L^+ to the right of $x = x_m$, by querying the data structure. We change x_m to the smaller x-coordinate of these two vertices, and then make local changes to L^- and L^+ ($O(1)$ insertions/deletions) before proceeding to the next iteration. The number of insertion and deletion operations is $O(h)$. □

Second, we need the current best known bound on the combinatorial complexity of the k-level (which is famously still an open problem), or slightly more generally, the $(k \pm s)$-levels:

Lemma 3.6. *For n pseudo-lines in the plane and $s \le k$, the total number of vertices in the j-level for $j = k-s, \ldots, k+s$ is at most $O(nk^{1/3}s^{2/3})$.*

Proof. Dey [18] proved this bound for lines in his celebrated paper; Tamaki and Tokuyama [43] provided the generalization to pseudo-lines. □

From these two facts, we can immediately solve our problem just by constructing the k-level in $\widetilde{O}(nk^{1/3})$ expected time, which is already an improvement for certain input parameters. The bound could in fact be better if the worst-case k-level complexity turns out to be smaller. But even with the current combinatorial bounds above, we can obtain a strict improvement, by a more focused divide-and-conquer around the neighborhood of the k-level, as we now describe:

Let $s \le k$ be a parameter to be set later. First choose a random number $s' \in \{1, \ldots, s\}$, and compute the $(k+s')$-level $\mathcal{L}^+$ and the $(k-s')$-level $\mathcal{L}^-$. The expected size of these two levels is $O(nk^{1/3}s^{2/3}/s) = O(nk^{1/3}/s^{1/3})$, and the expected construction time is thus $\widetilde{O}(nk^{1/3}/s^{1/3})$ by Lemma 3.5. We can repeat an $O(1)$ expected number of times to guarantee this size bound.

Divide the plane into $z = \Theta(nk^{1/3}/s^{4/3})$ vertical slabs each containing $O(s)$ vertices of $\mathcal{L}^+$ and $\mathcal{L}^-$. Let L_σ be the pseudo-lines that participate in defining vertices of $\mathcal{L}^+$ and $\mathcal{L}^-$ in σ; we have $|L_\sigma| = O(s)$. Let L'_σ be the pseudo-lines that are between $\mathcal{L}^-$ and $\mathcal{L}^+$ at the left wall of σ; we have $|L'_\sigma| \le O(s)$. We can compute L'_σ for all σ by sweeping the plane from left to right and maintaining the subset of $O(s)$ lines between $\mathcal{L}^-$ and $\mathcal{L}^+$ at the sweep line (whenever we hit a vertex of $\mathcal{L}^+$ or $\mathcal{L}^-$, we make local changes to L'_σ). Inside σ, the k-level of L corresponds to the k_σ-level of $L_\sigma \cup L'_\sigma$ for some k_σ. For example, we can take any fixed point p on $\mathcal{L}^-$ and set k_σ to be the level of p in $L_\sigma \cup L'_\sigma$ minus the level of p in L (the latter number is $k-s'$).

Subdivide slabs further so that each slab contains $O(m/z)$ x-values of X. The number of slabs remains $O(z)$.

For each slab σ, solve the subproblem for the $O(m/z)$ x-values of X in σ and the $O(s)$ pseudo-lines in $L_\sigma \cup L'_\sigma$, by the algorithm from Section 2.2.

The total expected running time is

$$\widetilde{O}(z \cdot ((m/z)^2 + s)) \;=\; \widetilde{O}(m^2/z + zs) \;=\; \widetilde{O}((m^2 s^{4/3})/(nk^{1/3}) + nk^{1/3}/s^{1/3}).$$

Setting $s = \min\{\lceil (nk^{1/3}/m)^{6/5} \rceil,\, k\}$ gives $\widetilde{O}(m^{2/5}n^{3/5}k^{1/5} + m + n)$, assuming that $m \le nk^{1/3}$. If $m > nk^{1/3}$, we can directly solve the problem using Lemma 3.5 in $\widetilde{O}(nk^{1/3}) \le \widetilde{O}(m)$ time.

Theorem 3.7. *Given an $m \times n$ totally monotone matrix and a number k, we can find the k-th smallest element in the i-th row, for all $i = 1, \ldots, m$, in $\widetilde{O}(m^{2/5}n^{3/5}k^{1/5} + m + n)$ expected time.*

Remarks. This result appears new even in the case of lines.

A similar strategy of doing divide-and-conquer around a neighborhood of the k-level was also employed in an algorithm in the appendix of [12] but for a different problem (computing all local minima of the k-level of n lines in the plane in $\widetilde{O}((nk)^{3/5} + n)$ time).

The same idea can be used to prove that given m lines and n points at level k in the plane, the number of point-line incidences is $O(m^{2/5}n^{3/5}k^{1/5} + m + n)$. An interesting combinatorial question (which might potentially be easier than the original k-level problem) is whether this incidence bound could be improved to near-linear.

4 Lower Bound

In this final section, we prove our lower bound for the t-ranking problem for an $m \times n$ totally monotone matrix A. The proof works even if A is Monge. We will lower-bound the number of accesses to the elements of A (which would automatically lower-bound the number of comparisons, for whatever the class of comparisons we allow).

We start with a known construction of a set P of m points and a set L of n lines in the plane that have $\Omega((mn)^{2/3} + m + n)$ incidences [35, 42]. We may assume that no two points of P have the same x-coordinates, no lines are vertical, and no two lines of L have the same slope, because we can apply a random affine transformation, which preserves incidences and guarantees that the condition is true with probability 1.

Sort the points $p_1, \ldots, p_m$ of P in increasing x-order and the lines $\ell_1, \ldots, \ell_n$ of L in decreasing order of slopes. Let the i-th point be (x_i, y_i) and the j-th line be $y = m_j x + b_j$. Define

$$A[i, j] = m_j x_i + b_j - y_i.$$

We can check that the Monge property is strictly satisfied: for any $i < i'$ and $j < j'$, $A[i, j] + A[i', j'] - A[i, j'] - A[i', j] = (m_{j'} - m_j)(x_{i'} - x_i) < 0$. This implies that A is (strictly) totally monotone.

Let δ be the minimum of $|(m_{j'} - m_j)(x_{i'} - x_i)|$ over all $i < i'$ and $j < j'$. Set $t = \delta/4$.

Observe that if (p_i, ℓ_j) is an incidence pair, then $A[i, j] = 0$. We claim that any correct deterministic algorithm must evaluate $A[i, j]$ for all incidence pairs (p_i, ℓ_j). If this is not true for some

incidence pair (p_i, ℓ_j), then the adversary could reset $A[i,j]$ from 0 to $\delta/2$; the Monge property would still be satisfied. But the rank of t would decrease by 1, and so the answer cannot be determined yet. This proves an $\Omega((mn)^{2/3} + m + n)$ lower bound. (If degeneracies are not allowed, we can first replace $A[i,j]$ by $A[i,j] + \delta_{ij}$ for some random $\delta_{ij} \in (0, \delta/100)$, which preserves the Monge property.)

Similarly, any randomized algorithm with $\Omega(1)$ correctness probability must evaluate $A[i,j]$ for at least a fraction of the incidence pairs (p_i, ℓ_j).

We can make the bound sensitive to K, by noting the existence of a set P and a set L with $\Omega((mnK)^{1/3} + m + n)$ incidences, where the sum of the levels of the points of P is bounded by K. To this end, we construct $\frac{n}{k_*}$ groups, where $k_* = \frac{K}{m}$, and each group consists of $m/(\frac{n}{k_*})$ points and k_* lines with $((m/(\frac{n}{k_*})) \cdot k_*)^{2/3} + m/(\frac{n}{k_*}) + k_*)$ incidences. We place each group in a bounding box, where the lines touch both the left and right sides of the box. Apply an affine transformation to turn each box into a very thin rectangle, so that we get $\frac{n}{k_*}$ thin rectangles in concave position, with sufficient spacing in between. This will ensure that no line from one group may intersect or go below the rectangle of another group. The sum of the ranks of the points is at most $O((\frac{n}{k_*})k_*^2) = O(K)$. The total number of incidences (and thus a lower bound on the number of evaluations) is

$$\Theta((\tfrac{n}{k_*}) \cdot ((m/(\tfrac{n}{k_*})) \cdot k_*)^{2/3} + k_* + m/(\tfrac{n}{k_*})) \;=\; \Theta(n^{1/3}m^{2/3}k_*^{1/3} + m + n) \;=\; \Theta((mnK)^{1/3} + m + n).$$

A similar argument applies to related problems including K-selection, row $(k_1, \ldots, k_m)$-selection, and row successors.

Theorem 4.1. *For any m, n, K with $K \leq mn/c$ for some constant c, any (deterministic or randomized) algorithm that counts the number of elements at most t in an $m \times n$ Monge matrix must access $\Omega((mnK)^{1/3} + m + n)$ elements, for some input with count K.*

Any algorithm that finds the K-th smallest element in an $m \times n$ Monge matrix must access $\Omega((mnK)^{1/3} + m + n)$ elements.

References

[1] Pankaj K. Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer. Maintaining the union of unit discs under insertions with near-optimal overhead. In *Proc. 35th International Symposium on Computational Geometry (SoCG)*, pages 26:1–26:15, 2019. doi:10.4230/LIPIcs.SoCG.2019.26.

[2] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999. URL: http://jeffe.cs.illinois.edu/pubs/survey.html.

[3] Pankaj K. Agarwal and Sandeep Sen. Selection in monotone matrices and computing kth nearest neighbors. *J. Algorithms*, 20(3):581–601, 1996. doi:10.1006/jagm.1996.0028.

[4] 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.

[5] 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.

[6] Noga Alon and Yossi Azar. Comparison-sorting and selecting in totally monotone matrices. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 403–408, 1992. URL: http://dl.acm.org/citation.cfm?id=139404.139484.

[7] Wolfgang W. Bein, Peter Brucker, Lawrence L. Larmore, and James K. Park. The algebraic Monge property and path problems. *Discret. Appl. Math.*, 145(3):455–464, 2005. doi:10.1016/j.dam.2004.06.001.

[8] Jean-Daniel Boissonnat and Jack Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. *Comput. Geom.*, 16(1):35–52, 2000. doi:10.1016/S0925-7721(99)00057-7.

[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] Timothy M. Chan. Reporting curve segment intersections using restricted predicates. *Comput. Geom.*, 16(4):245–256, 2000. doi:10.1016/S0925-7721(00)00012-2.

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

[12] Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.

[13] Timothy M. Chan. (Near-)linear-time randomized algorithms for row minima in Monge partial matrices and related problems. Manuscript, 2020.

[14] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discret. Comput. Geom.*, 9:145–158, 1993. doi:10.1007/BF02189314.

[15] Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discret. Comput. Geom.*, 2:195–222, 1987. doi:10.1007/BF02187879.

[16] 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.

[17] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[18] Tamal K. Dey. Improved bounds for planar k-sets and related problems. *Discret. Comput. Geom.*, 19(3):373–382, 1998. doi:10.1007/PL00009354.

[19] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15(2):341–363, 1986. doi:10.1137/0215024.

[20] Herbert Edelsbrunner, Raimund Seidel, and Micha Sharir. On the Zone Theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993. doi:10.1137/0222031.

[21] Herbert Edelsbrunner and Emo Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15(1):271–284, 1986. doi:10.1137/0215019.

[22] Jeff Erickson. New lower bounds for Hopcroft's problem. *Discret. Comput. Geom.*, 16(4):389–418, 1996. doi:10.1007/BF02712875.

[23] Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982. doi:10.1016/0022-0000(82)90048-4.

[24] Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.*, 13(1):14–30, 1984. doi:10.1137/0213002.

[25] 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.

[26] Jacob E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discret. Math.*, 32(1):27–35, 1980. doi:10.1016/0012-365X(80)90096-5.

[27] John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM J. Comput.*, 26(6):1612–1634, 1997. doi:10.1137/S0097539793253577.

[28] 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.

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

[30] 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.

[31] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998. doi:10.1137/S0097539796305365.

[32] Yishay Mansour, James K. Park, Baruch Schieber, and Sandeep Sen. Improved selection in totally monotone arrays. *Int. J. Comput. Geom. Appl.*, 3(2):115–132, 1993. doi:10.1142/S0218195993000087.

[33] Jirí Matoušek. Efficient partition trees. *Discret. Comput. Geom.*, 8:315–334, 1992. doi:10.1007/BF02293051.

[34] Jirí Matoušek. Range searching with efficient hierarchical cuttings. *Discret. Comput. Geom.*, 10:157–182, 1993. doi:10.1007/BF02573972.

[35] Jirí Matoušek. *Lectures on Discrete Geometry*. Springer, 2002.

[36] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

[37] 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.

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

[39] Ketan Mulmuley and Sandeep Sen. Dynamic point location in arrangement of hyperplanes. *Discret. Comput. Geom.*, 8:335–360, 1992. doi:10.1007/BF02293052.

[40] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

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

[42] Endre Szemerédi and William T. Trotter. Extremal problems in discrete geometry. *Combinatorica*, 3(3):381–392, 1983. doi:10.1007/BF02579194.

[43] Hisao Takeshi and Takeshi Tokuyama. A characterization of planar graphs by pseudo-line arrangements. *Algorithmica*, 35(3):269–285, 2003. doi:10.1007/s00453-002-0999-9.

[44] Dan E. Willard. Polygon retrieval. *SIAM J. Comput.*, 11(1):149–165, 1982. doi:10.1137/0211012.