Superlinear Bounds For Matrix Searching Problems

by

Maria M. Klawe

Technical Report 90-26
July, 1990

Department of Computer Science
University of British Columbia
6356 Agricultural Road
Vancouver, B.C. Canada V6T 1W5
e-mail: klawe@cs.ubc.ca

SUPERLINEAR BOUNDS FOR MATRIX SEARCHING PROBLEMS

Maria M. Klawe
Department of Computer Science, UBC
Vancouver, B.C., Canada V6T 1W5

Abstract

Matrix searching in classes of totally monotone partial matrices has many applications in computer science, operations research, and other areas. This paper gives the first superlinear bound for matrix searching in classes of totally monotone partial matrices, and also contains some new upper bounds for a class with applications in computational geometry and dynamic programming.

The precise results of this paper are as follows. We show that any algorithm for finding row maxima or minima in totally monotone partial $2n \times n$ matrices with the property that the non-blank entries in each column form a contiguous segment, can be forced to evaluate $\Omega(n\alpha(n))$ entries of the matrix in order to find the row maxima or minima, where $\alpha(n)$ denotes the very slowly growing inverse of Ackermann's function. A similar result is obtained for $n \times 2n$ matrices with contiguous non-blank segments in each row. The lower bounds are proved by introducing the concept of an independence set in a partial matrix and showing that any matrix searching algorithm for these types of partial matrices can be forced to evaluate every element in the independence set. A result involving lower bounds for Davenport-Schinzel sequences is then used to construct an independence set of size $\Omega(n\alpha(n))$ in the matrices of size $2n \times n$ and $n \times 2n$.

We also give two algorithms to find row maxima and minima in totally monotone partial $n \times m$ matrices with the property that the non- blank entries in each column form a contiguous segment ending at the bottom row. The first algorithm evaluates at most $O(m\alpha(n)+n)$ entries of the skyline matrix and performs at most that many comparisons, but may have $O(m\alpha(n)\log\log n + n)$ total running time. The second algorithm is simpler and has $O(m\log\log n + n)$ total running time.

A preliminary version of this paper appeared in the Proceedings of the First ACM/SIAM Symposium on Discrete Algorithms, 1990. The research in this paper was partially supported by an NSERC Operating Grant.

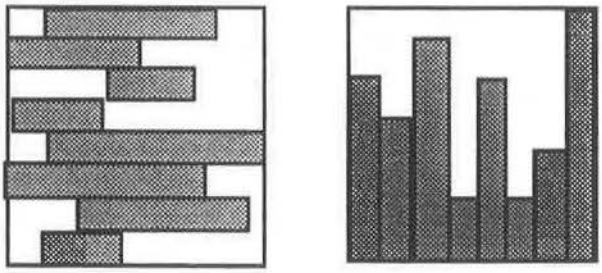
1. Introduction

The technique of matrix searching in totally monotone matrices and their generalizations is steadily finding ever more applications in a wide variety of areas of computer science and operations research, especially computational geometry and dynamic programming problems (see [AKMSW87], [AK90], [AP88], [AS87], [AS89], [EGG88], [KK90], [LS90], and [W88] for example). Although an asymptotically optimal linear time algorithm is known for the most basic problem of finding row minima and maxima in totally monotone matrices [AKMSW87], for most of the generalizations of totally monotone matrices, only superlinear algorithms are known, though until now no superlinear lower bounds have been proved. This paper gives the first superlinear bound for matrix searching in two types of totally monotone partial matrices. These types of

matrices, which we refer to as v-matrices and h-matrices, respectively, were introduced by Aggarwal and Suri [AS89] who used them to find the farthest visible pair in a simple polygon. In addition, these matrix classes are natural extensions of staircase matrices which have applications in computational geometry and dynamic programming problems. We also give new upper bounds for matrix searching in a subclass of v-matrices, which arises in problems in computational geometry and operations research.

A **partial matrix** is a matrix in which entries are either real numbers or are blank. A partial matrix $M = (M_{ij})$ is called **totally monotone** if for every $i < i', j < j'$ such that all entries of the 2×2 submatrix, $M_{ij}, M_{ij'}, M_{i'j}$, and $M_{i'j'}$, are non-blank, whenever $M_{ij} \leq M_{ij'}$ we have $M_{i'j} \leq M_{i'j'}$. A **totally monotone matrix** is a totally monotone partial matrix with no blank entries.

We will call a totally monotone partial matrix a **v-matrix** (vertical matrix) if the set of non-blank entries in each column forms a contiguous interval. Similarly, an **h-matrix** (horizontal matrix) is a totally monotone partial matrix such that the set of non-blank entries in each row forms a contiguous interval. Finally, a **skyline** matrix is a v-matrix such that every column's non-blank segment ends at the bottom row. A partial matrix is a **staircase** matrix if it is both a v-matrix and an h-matrix (this definition is slightly more general than the one given in [AK90] and [KK90], but the algorithms of those papers can be trivially extended to handle this definition of staircase matrix). Examples of an h-matrix and skyline matrix are shown in Figure 1, where the grey areas indicate the regions containing non-blank entries.



(a) h-matrix (b) skyline matrix

Figure 1

Totally monotone matrices were introduced by Aggarwal, Klawe, Moran, Shor and Wilber in [AKMSW87], who showed that several problems in computational geometry could be reduced to finding the maximum or minimum value in each row of a totally monotone matrix. We will use the term **matrix searching** to refer to the task of finding row minima or maxima in a matrix. Aggarwal et al gave an $O(m+n)$ time algorithm (which we will refer to as the SMAWK algorithm) for matrix searching in $n \times m$ totally monotone matrices, yielding faster algorithms for a broad collection of problems. Wilber [W88] used the SMAWK algorithm to get a linear time algorithm for a dynamic programming problem known as the concave least weight subsequence problem. Aggarwal and Klawe [AK90] generalized totally monotone matrices to staircase matrices, and showed that additional problems of computational geometry could be reduced to matrix searching in staircase matrices. Aggarwal and Klawe [AK90] also gave an $O(m \log\log n)$ time algorithm for searching staircase matrices of size $n \times m$, again yielding faster algorithms for several problems in computational geometry. Klawe and Kleitman [KK90] gave an $O(m\alpha(n) + n)$ time algorithm for matrix searching in staircase matrices, and extended this algorithm to handle a class of dynamic programming problems satisfying convex quadrangle inequalities which have many applications in molecular biology and other areas (see [E88], [EGG88], [EGGI90], [GG89],

[HL87], [LS90]). The function, $\alpha(n)$, denotes the very slowly growing inverse of Ackermann's function [T75]; we give the precise definition of $\alpha(n)$ in the third paragraph of section 3. In [AS89], Aggarwal and Suri introduced v-matrices and h-matrices, and used matrix searching in these matrices to give a faster algorithm for computing the farthest visible vertex pair in a simple polygon. Aggarwal and Park [AP90, section 3.1] also use matrix searching in v-matrices to give faster algorithms for solving a common problem in inventory control and production planning, the economic lot-size problem, under certain assumptions.

This paper has two main contributions. The first is a superlinear lower bound for matrix searching in v-matrices and h-matrices. This is the first superlinear lower bound for matrix searching in totally monotone matrices. The problem of extending this lower bound to staircase matrices remains open, and requires at least one more idea since we can show that our current techniques will not suffice. The second contribution is the extension of the staircase matrix searching algorithms of [KK90] and [AK90] for totally monotone staircase matrices to skyline matrices. The first yields an algorithm which evaluates at most $O(m\alpha(n)+n)$ entries of the skyline matrix. For the staircase algorithm in [KK90], Klawe and Kleitman show that it is also possible to initialize and maintain all the necessary data-structures, resulting in a total running time of at most $O(m\alpha(n)+n)$. So far we have not been able to extend this result to skyline matrices, and the best bound on total running time for a skyline algorithm we can give at this point is $O(m \log\log n + n)$ which is achieved by extending the simpler staircase algorithm of [AK90]. Like the staircase algorithms [KK90, section 4], the skyline algorithms can be modified so that they possess an evaluation ordering property. This property is a type of 'on-line' condition which is essential for applications involving dynamic programming such as the economic lot-size problem of [AP90]. The question of extending the skyline algorithms to obtain a $o(m \log n + n)$ time matrix searching algorithm for either v-matrices or h-matrices remains open.

In the next section we prove the lower bound for v-matrices and h- matrices. Section 3 contains the extension of the staircase algorithm of [KK90] to skyline matrices. Section 4 discusses the data-structures needed by our skyline algorithm, and proves that the extension of the [AK90] algorithm has $O(m \log\log n+n)$ total running time. Section 4 also defines the on-line version, and shows how the [AK90] extension can be modified to satisfy the on-line condition. The final section describes remaining open problems.

2. The Lower Bound

We assume that an algorithm for matrix searching in a partial matrix is given as input the pattern of non-blank entries in the matrix. For a v-matrix this is simply the positions of the top and bottom non-blank entry in each column. We will refer to this pattern matrix indicating the positions of non-blanks as the **structure matrix** (or structure v-matrix or h-matrix as appropriate) of the partial matrix. The algorithm may query the value of any entry in the matrix at any time, and at the end must report the position of the maximum [minimum] value in each row. We will prove a lower bound on the number of entries that must be evaluated in the worst-case.

Our strategy to prove the lower bound is as follows. Given a fixed structure matrix, we define the concept

of an independence set for that structure matrix. Next we show, using a result by Wiernik which gives an $\Omega(n\alpha(n))$ lower bound on the left envelope of n line segments in the plane [W86], that there is a structure matrix of size $2n \times n$ possessing the column interval property which has an independence set of size $\Omega(n\alpha(n))$. Transposing this matrix gives a structure h-matrix of size $2n \times n$ with an independence set of size $\Omega(n\alpha(n))$. The final step is to exhibit an adversary which can respond to queries in such a way that that the matrix created is totally monotone, and such that any element of the independence set which has not yet been queried is still a candidate, but not a certainty, for the maximum in its row. In the remainder of this section we give the definition of independence set and show how Wiernik's result gives a structure matrix with the desired size of independence set. For the v-matrix case we construct the adversary directly from Wiernik's result, but this does not seem to work for the h-matrix case.

Given a set of line segments $l_1, \ldots, l_n$ in the plane, we define their **left envelope** to be the set of points $\{z : z \in l_i$ for some i, and z is the leftmost point in the intersection of $\cup_{j=1}^{n} l_i$ with the horizontal line through $z\}$. Figure 2(a) shows a set of line segments and their left envelope. It is easy to see that the left envelope is always the union of a finite set of line segments. Both Wiernik's lower bound and a recent simplification by Shor [S89] spring from the original lower bound on Davenport-Schinzel sequences of order 3 by Hart and Sharir [HS86]. A Davenport-Schinzel sequence of order 3 is a sequence of letters $s_1, \ldots, s_k$ from an alphabet of size n such that no two consecutive letters in the sequence are the same, and such that for any pair of distinct letters a, b there do not exist indices $i_1 < i_2 < i_3 < i_4 < i_5$ with $i_1 = i_3 = i_5 = a$ and $i_2 = i_4 = b$. In [HS86] Hart and Sharir proved $\Theta(n\alpha(n))$ bounds for the length of Davenport-Schinzel sequences of order 3 on n letters. It is easy to see that sequence of lines appearing in top to bottom order in the left envelope of a set of n line segments in the plane is a Davenport-Schinzel sequence of order 3 on n letters. Thus, by [HS86], the left envelope is made up of at most $O(n\alpha(n))$ segments, and Wiernik's construction shows that this upper bound is tight. It is possible to construct the desired independence set and the adversary for the v-matrix case directly from a Davenport-Schinzel sequence of order 3 on n letters in a fashion entirely analogous to what we will do with the left envelope, but since the construction seems somewhat easier to follow using the left-envelope version, we use that instead.

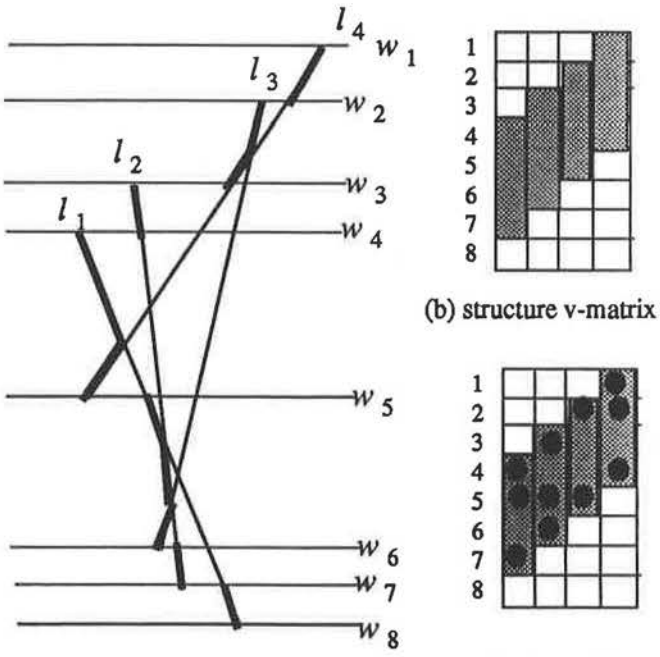
Let A be an $n \times m$ structure matrix. A subset $S \subset \{1, \ldots, n\} \times \{1, \ldots, m\}$ is said to be an **independence set for** A if
(i) for every (i, j) in S, the entry A_{ij} is non-blank and there exists some $j' \neq j$ such that (i, j') is also in S, and
(ii) for every $i < i'$ and $j < j'$ such that both (i, j') and (i', j) are in S, we have that either A_{ij} is blank or $A_{i'j'}$ is blank.

For any matrix M we will call the ordered pair (i, j) the **index** of the entry M_{ij}. Intuitively, the elements of an independence set of a structure matrix are the indices of entries which will be potential row-maxima when we construct the adversary giving the lower bound.

Let $l_1, \ldots, l_n$ be line segments in the plane such that their left envelope has $\Omega(n\alpha(n))$ segments. For each i let (x_1^i, y_1^i) and (x_2^i, y_2^i) be the top and bottom endpoints of l_i respectively, and let L_i be the infinite line extending l_i. Suppose the line segments are ordered so that whenever $i < j$, as y goes to ∞ the line

L_i is eventually to the left of L_j. We use the l_i to define a $2n \times n$ structure matrix, A, as follows. Let $\{w_1, \ldots, w_{2n}\} = \{y_j^i : j = 1, 2; i = 1, \ldots, n\}$ arranged in decreasing order. Without loss of generality we may assume that the $\{w_i\}$ are all distinct. The i-th row of the structure matrix corresponds to w_i and the j-th column corresponds to the line segment l_j. More precisely, the top non-blank entry in the j-th column of A is in the row i such that $w_i = y_1^j$ and the bottom non-blank entry is in the row $i' - 1$ where $w_{i'} = y_2^j$. A is obviously a structure v-matrix. We now show that A has an independence set of size $\Omega(n\alpha(n))$. Figure 2(b) shows the structure v-matrix corresponding to the line segments in Figure 2(a).

We start with a set T that is almost an independence set. The only way in which it may fail is that there may be some rows in which T only has one entry. Let $T = \{(i, j) :$ there is some y_0 with $w_i \geq y_0 > w_{i+1}$ such that the line segment forming the left envelope at $y = y_0$ is $l_j\}$. It is easy to see that A must be non-blank at every (i, j) in T. Suppose $i < i'$ and $j < j'$ such that both (i, j') and (i', j) are in T, and suppose both (i, j) and (i', j') are non-blank in A. Let z be the y-coordinate of the intersection of L_j and $L_{j'}$. Because of the ordering of the line- segments and the fact that $(i, j') \in T$, it is not hard to see that we must have $z > w_{i+1}$ and hence $z > w_{i'}$. Since (i', j') is non-blank, it is impossible that $(i', j) \in T$ since $l_{j'}$ lies to the left of l_j for the entire interval between $w_{i'}$ and $w_{i'+1}$. Thus at least one of (i, j) and (i', j') must be blank. Figure 2(c) shows the set T for the line segments in Figure 2(a).



(a) the left envelope

(b) structure v-matrix

(c) the set T

Figure 2

We complete the construction of the independence set, S, by removing all points from T which are the unique point in their row. We claim that S has size $\Omega(n\alpha(n))$. Since we removed at most $2n$ points, it suffices to show that the size of T is $\Omega(n\alpha(n))$. This follows immediately from the observation that in any

interval in which no line segment begins or ends, each l_j can occur in the left envelope at most once.

We now turn to the problem of constructing an adversary which will force a row-maxima finding algorithm to evaluate every entry whose index is in the independence set. We first define a v-matrix, M, whose structure matrix is A. We then prove that M is totally monotone. Next we will define a set of v-matrices M^f, such that each M^f has structure matrix A and agrees with M on all entries outside the independence set. We then prove that each M^f is totally monotone. Finally we construct an adversary for the searching algorithm such that the positions of the row-maxima cannot be known until each element whose index is in the independence set has been queried, and such that the final matrix will be M^f for some f.

Let M be the v-matrix with structure matrix A defined by M_{ij} = the maximum number of lines lying to the right of l_j at any point strictly between w_i and w_{i+1} whenever A_{ij} is non-blank. Note that M_{ij} assumes a maximal value in row i if and only if part of l_j is in the left envelope between w_i and w_{i+1} . The next lemma proves that M is totally monotone.

Lemma 2.1. M is totally monotone.

Proof. Suppose $i < i', j < j'$ such that all entries of the 2×2 submatrix, $M_{ij}, M_{ij'}, M_{i'j}$, and $M_{i'j'}$, are non-blank, and $M_{ij} \leq M_{ij'}$. We must show that $M_{i'j} \leq M_{i'j'}$. Let z be the y-coordinate of the intersection of L_j and $L_{j'}$. Since $j < j'$ we know L_j lies to the left of $L_{j'}$ as y goes to ∞. Since $M_{ij} \leq M_{ij'}$ we must have l_j lying to the right of $l_{j'}$ at some point strictly between w_i and w_{i+1}. Thus we must have $z > w_{i+1}$ and hence $z > w_{i'}$ since $w_{i+1} \geq w_{i'}$. This shows that l_j is to the right of $l_{j'}$ at every point between $w_{i'}$ and $w_{i'+1}$, and hence $M_{i'j} \leq M_{i'j'}$. ∎

For each function f from S to the non-negative real numbers, we define M^f to be the v-matrix such that $M^f_{ij} = M_{ij} + f(i,j)$ for $(i,j) \in S$ and $M^f_{ij} = M_{ij}$ otherwise. The next lemma shows that M^f is totally monotone.

Lemma 2.2. For any function f from S to the non-negative real numbers, the v-matrix M^f is totally monotone.

Proof. Suppose $i < i', j < j'$ such that all entries of the 2×2 submatrix, $M^f_{ij}, M^f_{ij'}, M^f_{i'j}$, and $M^f_{i'j'}$, are non-blank, and $M^f_{ij} \leq M^f_{ij'}$. We must show that $M^f_{i'j} \leq M^f_{i'j'}$. If none of the indices are in S this follows from Lemma 2.1, so we may assume that at least one of the indices is in S. Since S is an independence set, we cannot have both (i,j') and (i',j) in S. Also $M^f_{ij} \leq M^f_{ij'}$ implies that if (i,j) is in S then (i,j') is also. Moreover, if (i',j') is in S then either (i',j) is also, or $M^f_{i'j} \leq M^f_{i'j'}$. Thus it suffices to consider the cases $(i,j') \in S$ and $(i',j) \in S$. Suppose we have $(i,j') \in S$. This implies that $M_{ij} \leq M_{ij'}$, and hence $M_{i'j} < M_{i'j'}$ by Lemma 2.1. In addition, $M^f_{i'j} = M_{i'j}$ since $(i',j) \notin S$, and $M_{i'j'} \leq M^f_{i'j'}$ since f only assumes non-negative values. Combining this gives $M^f_{i'j} \leq M^f_{i'j'}$ as desired. Now suppose $(i',j) \in S$. Let z be the y-coordinate of the intersection of L_j and $L_{j'}$. Since $(i',j) \in S$ we must have $w_{i'} > z$, and hence l_j lies to left of $l_{j'}$ at every point between w_i and w_{i+1}, contradicting the assumption $M^f_{ij} \leq M^f_{ij'}$, and completing the proof. ∎

We are now ready to define the behaviour of an adversary for any row-maxima finding algorithm on v-matrices with structure matrix A. When the algorithm queries the entry with index (i,j), the adversary will respond with M_{ij} for $(i,j) \notin S$ and $M_{ij} + k + 1$ for $(i,j) \in S$, where k is the number of entries with indices in S that the algorithm has queried so far. By Lemma 2.2 the matrix produced by the adversary is totally monotone. Moreover, if (i,j) is the last index in S to be queried by the algorithm, the adversary could answer M_{ij} instead of $M_{ij} + |S|$ and still produce a totally monotone matrix. Since S has at least two indices in row i, the question of whether M_{ij} is a row-maxima cannot be answered without evaluating it. This shows that the algorithm must evaluate $|S| = \Omega(n\alpha(n))$ entries of the v-matrix in order to determine the positions of the row-maxima.

We now turn to the proof of the lower bound for the h-matrix case. Let A, T be the transposed versions of the structure matrix and "pre-independence set" from the proof for the v-matrix case. Clearly A is a structure h-matrix. As before let S be the set obtained by deleting any element of T which is the unique element of T in its row. It is easy to check that S is an independence set for A because property (ii) of the the definition of independence set is invariant under transposition.

We first define $S_i = \{j : (i,j) \in S\}$. Similar to the proof in the v-matrix case we will construct an h-matrix M with structure matrix A such that for each function f from S to the non-negative reals, the matrix M^f defined by $M^f_{ij} = M_{ij}$ for $(i,j) \notin S$ and $M^f_{ij} = |S_i| + f(i,j)$ is totally monotone. Given M, the adversary which forces a row-maxima finding algorithm to evaluate each entry with an index in S is completely analogous to the v-matrix case. The construction of M takes a bit more work in this case than in the v-matrix case. For each i let $A_i = \{(i,j) : A_{ij} \text{ is non-blank}\}$. We begin by defining a partial order on each A_i.

Let $j, j' \in A_i$. We define a relation $\propto_i$ on A_i by $j \propto_i j'$ if any of the following hold:
(i) $(i,j) \notin S$ and $(i,j') \in S$.
(ii) Neither (i,j) nor (i,j') are in S, $j < j'$ and for some $h < i$ we have $(h,j') \in S$ and A_{hj} non-blank.
(iii) Neither (i,j) nor (i,j') are in S, $j' < j$ and for some $i' > i$ we have $(i',j') \in S$ and $A_{i'j}$ non-blank.

Let $\prec_i$ be the transitive closure of $\propto_i$, i.e. $j \prec_i j'$ if for any $k \geq 1$ there exist $j_0, j_1, \ldots, j_k \in A_i$ with $j = j_0 \propto_i j_1 \propto_i \ldots \propto_i j_k = j'$.

Remark 2.3. Whenever $p, q \in A_i$ with $p \prec_i q$ we have $(i,p) \notin S$.

Proof. This follows immediately from the observation that whenever $p, q \in A_i$ with $p \propto_i q$ we have $(i,p) \notin S$. ∎

Lemma 2.4. Suppose A is a structure h-matrix. Then $\prec_i$ is a partial order on A_i.

Proof. Since $\prec_i$ is obviously transitive, it suffices to show that we cannot have $j \prec_i j$ for any j in A_i. Suppose the contrary. Let $k \geq 1$ and $j_0, j_1, \ldots, j_k \in A_i$ such that $j = j_0 \propto_i j_1 \propto_i \ldots \propto_i j_k = j$, and suppose that j and k are chosen so that k is minimal, i.e. whenever $j'_0 \propto_i j'_1 \propto_i \ldots \propto_i j'_{k'} = j'_0$ we have $k' \geq k$. It is easy to check from the definition of $\propto_i$ that we never have $j' \propto_i j'$ for any $j' \in A_i$. It is also not hard to see

that we cannot have $j \propto_i j' \propto_i j$ for any pair j, j' in A_i. To see this, suppose without loss of generality that $j < j'$. Then in order to have $j \propto_i j' \propto_i j$, we must have some $h < i$ such that $(h, j') \in S$ and A_{hj} non-blank, and some $i' > i$ such that $(i', j) \in S$ and $Ai'j'$ non-blank. However, this contradicts the independence of S since we have $h < i', j < j'$ with both (h, j') and (i', j) in S and both Ahj and $A_{i'j'}$ non-blank. Thus we may assume that $k \geq 3$, and that $j_0 < j_s$ for $s = 1, ..., k-1$. Also, note that (i, j_s) is not in S for $0 \leq s \leq k$. This is obvious for $0 \leq s \leq k-1$ by Remark 2.3, and also for $s = k$ since $j_k = j_0$. Thus whenever $j_s < j_{s+1}$ there is some $h_s < i$ such that $(h_s, j_{s+1}) \in S$ and $A_{h_s j_s}$ is non-blank, and whenever $j_s > j_{s+1}$ there is some $i_s > i$ such that $(i_s, j_{s+1}) \in S$ and $A_{i_s j_s}$ is non-blank.

Choose r such that $|j_r - j_{r+1}|$ is maximal. Without loss of generality we assume that $j_r < j_{r+1}$ (the proof for the other case is symmetric). Let t such that $j_t > j_s$ for $s \neq t$. It is not hard to prove that for any q with $j_0 < q \leq j_t$, there is some s and some s' such that $j_s < q \leq j_{s+1}$ and $j_{s'+1} < q \leq j_{s'}$. For example, taking s to be maximal such that $j_w < q$ for each $w \leq s$, and s' to be minimal such that $j_x < q$ for each $x > s'$ will do. Thus there is some y such that $j_{y+1} < j_{r+1} \leq j_y$. Now since $|j_r - j_{r+1}|$ is maximal, we must have $j_r \leq j_{y+1}$. We have $h_r < i < i_y$ and both (h_r, j_{r+1}) and (i_y, j_{y+1}) in S and both $A_{h_r j_r}$ and $A_{i_y j_y}$ are non-blank. Moreover, since A has the row interval property and $j_r \leq j_{y+1} < j_{r+1} \leq j_y$, we must have that $A_{h_r j_{y+1}}$ and $A_{i_y j_{r+1}}$ are non-blank. Now this contradicts the independence of S, completing the proof. ∎

If for $i = 1, \ldots, n$ we have a linear order $<_i$ on each A_i, we define the **canonical partial matrix** generated by the $\{<_i\}$ to be the matrix M with M_{ij} = the position of j in the $<_i$ ordering of A_i if A_{ij} is non-blank, and blank otherwise. We will say that a set $\{\prec_i : \prec_i$ is a partial order on $A_i\}$ is **consistent** if whenever $j, j' \in A_i$ with $j < j'$ and $j \prec_i j'$, for every i' with $i < i'$ and $j, j' \in A_{i'}$ we have $j \prec_{i'} j'$.

Remark 2.5. If the linear orderings $\{<_i\}$ are consistent then the canonical partial matrix generated by the $\{<_i\}$ is totally monotone.

Proof. This follows immediately from the definition of total monotonicity. ∎

Suppose $\prec_i$ is a partial order on A_i and j, j' are incomparable elements of A_i with $j < j'$. We define the partial order $\prec_i^+ (j, j')$ on A_i to be the extension of $\prec_i$ obtained by adding the relation $j \prec_i^+ j'$, and similarly define the partial order $\prec_i^- (j, j')$ on A_i to be the extension of $\prec_i$ obtained by adding the relation $j' \prec_i^- j$. If $P = \{\prec_s : s = 1, \ldots, n\}$ we use $P_i^+(j, j')$ and $P_i^-(j, j')$ to denote the sets obtained by replacing $\prec_i$ in P by $\prec_i^+ (j, j')$ and $\prec_i^- (j, j')$ respectively.

Lemma 2.6. If $P = \{\prec_s : s = 1, \ldots, n\}$ is consistent and j, j' are incomparable elements of A_i with $j < j'$, then at least one of $P_i^+(j, j')$ and $P_i^-(j, j')$ must be consistent.

Proof. Suppose not. Since $P_i^+(j, j')$ is not consistent, there is some $i' > i$ with $j, j' \in A_{i'}$ and $j' \prec_{i'} j$. Similarly, since $P_i^-(j, j')$ is not consistent, there is some $h < i$ with $j, j' \in A_h$ and $j \prec_h j'$, but this contradicts the consistency of P. ∎

Corollary 2.7. If $P = \{\prec_s : s = 1, \ldots, n\}$ is consistent then there is a consistent set P' of linear orderings extending P.

Proof. This follows immediately from Lemma 2.6. ∎

Lemma 2.8. Suppose A is a structure h-matrix, and $j_1, \ldots, j_k \in A_i$ with $j_1 \propto_i \ldots \propto_i j_k$. Then if $(j_s - j_{s-1})(j_k - j_{k-1}) < 0$ for each $s = 2, \ldots, k-1$, then j_k cannot lie between j_{s-1} and j_s for $s = 2, \ldots, k-1$.

Proof. First suppose $j_k < j_{k-1}$. This implies that $j_{s-1} < j_s$ for each $s = 2, \ldots, k-1$. Thus it suffices to show $j_k < j_{s-1}$ for each $s = 2, \ldots, k$. The proof is by backwards induction on s. This holds for $s = k$ since we assumed $j_k < j_{k-1}$, so suppose $3 \leq s \leq k$ and $j_k < j_{s-1}$. Since $j_{s-2} \propto_i j_{s-1}$ there is some $h < i$ such that $j_{s-2}, j_{s-1} \in A_h$ and $(h, j_{s-1}) \in S$. Similarly as $j_{k-1} \propto_i j_k$ there is some $i' > i$ such that $j_{k-1}, j_k \in A_{i'}$ and $(i', j_k) \in S$. If $j_k \geq j_{s-2}$ then $j_k \in A_h$ because A has the row interval property and $j_{s-2} \leq j_k < j_{s-1}$. This contradicts the independence of S. The argument for the case $j_k > j_{k-1}$ is symmetric. ∎

Lemma 2.9. Suppose A is a structure h-matrix, and $j_1, \ldots, j_k \in A_i$ with $j_1 \propto_i \ldots \propto_i j_k$, where $k \geq 2$. Let p be minimal such that $(j_s - j_{s-1})(j_k - j_{k-1}) > 0$ for all s with $p \leq s \leq k$. Then j_s lies between j_{p-1} and j_p for $s = 1, \ldots, p$.

Proof. The proof is by induction on k. It is obviously true for $k = 2$ so assume $k > 2$ and that the hypothesis holds for $k-1$. Let q be minimal such that $(j_s - j_{s-1})(j_{k-1} - j_{k-2}) > 0$ for all s with $q \leq s \leq k-1$. Without loss of generality suppose $j_k < j_{k-1}$. If $p \leq k-1$ it is easy to see that statement holds since clearly $p = q$. Thus suppose $p = k$. This implies that $(j_s - j_{s-1})(j_k - j_{k-1}) < 0$ for $s = q, \ldots, k-1$. Now by Lemma 2.8, we have that $j_k < j_{s-1}$ for $s = q, \ldots, k$ and the interval $[j_k, j_{k-1}]$ contains the interval $[j_{s-1}, j_s]$ for $s = q, \ldots, k-1$. This completes the proof as the interval $[j_{q-1}, j_q]$ contains all the j_s for $s = 1, \ldots, q$ by the induction hypothesis. ∎

Corollary 2.10. Suppose A is a structure h-matrix, $j_1, \ldots, j_k \in A_i$ with $j_1 \propto_i \ldots \propto_i j_k$, where $k \geq 2$. Then j_k is either the maximum or minimum of $\{j_s : 1 \leq s \leq k\}$.

Proof. Let p be as in Lemma 2.9. Without loss of generality suppose $j_p < j_{p+1} < \ldots < j_{k-1} < j_k$. If $p = 1$ we are done so assume $p > 1$. Then $j_p < j_{p-1}$ so by Lemma 2.9 it suffices to show that $j_{p-1} < j_k$. If $j_k < j_{p-1}$ then j_{p+1} lies between j_p and j_{p-1} but this is impossible by Lemma 2.8 since $(j_p - j_{p-1})(j_{p+1} - j_p) < 0$. ∎

Lemma 2.11. Suppose $x, y \geq 2, a_1 < \ldots a_x, b_y < \ldots < b_1, a_1 < b_y$ and $a_x < b_1$. Then there exist u, v with $2 \leq u \leq x, 2 \leq v \leq y$ such that $a_{u-1} \leq b_v < a_u \leq b_{v-1}$.

Proof. Choose $u, v \geq 2$ such that $b_v < a_u$ and such that $a_u - b_v$ is minimal. It is always possible to do this since $b_y < a_x$ and $x, y \geq 2$, and clearly by the minimality of $a_u - b_v$ we have $a_{u-1} \leq b_v < a_u \leq b_{v-1}$. ∎

Theorem 2.12. Suppose A is a structure h-matrix. Then the set $\{\prec_i\}$ is consistent.

Proof. Suppose there exist $i < i', j < j'$ such that $j, j' \in A_i \cap A_{i'}$ and $j \prec_i j', j' \prec_{i'} j$. Then there exist $k, k', j_1, \ldots, j_k, j'_1, \ldots, j'_{k'}$ such that $j = j_1 \propto_i \ldots \propto_i j_k = j'$ and $j' = j'_1 \propto_{i'} \ldots \propto_{i'} j'_{k'} = j$. Moreover, since $j < j'$ by Corollary 2.10 we have $j' > j_s$ for $s = 1, \ldots, k-1$ and $j < j'_s$ for $s = 2, \ldots, k'$. Let p be minimal such that $(j_s - j_{s-1})(j_k - j_{k-1}) > 0$ for all s with $p \leq s \leq k$, and let p' be minimal such that $(j'_s - j'_{s-1})(j'_{k'} - j'_{k'-1}) > 0$ for all s with $p' \leq s \leq k'$. By Lemma 2.9 we have $j_{p-1} \leq j, \quad j_{p-1} < j_p <$

$j_{p+1} < \ldots < j_{k-1} < j_k = j'$, $j' \leq j'_{p'-1}$ and $j = j'_{k'} < j'_{k'-1} < \ldots < j'_{p'} < j'_{p'-1}$. Now by Lemma 2.11 there exist $u, v \geq 2$ such that $j_{u-1} \leq j'_v < j_u \leq j'_{v-1}$. Now since $j_{u-1} < j_u$ and $j_{u-1} \propto_i j_u$, there is some $h < i$ such that $j_{u-1}, j_u \in A_h$ and $(h, j_u) \in S$. Moreover, as A is a structure h-matrix, j'_v must be in A_h also. Likewise, as $j'_{v-1} > j'_v$ and $j'_{v-1} \propto_{i'} j'_v$, there is some $r > i'$ such that $j'_{v-1}, j'_v \in A_r$ and $(r, j'_v) \in S$. Finally, as A is a structure h-matrix, j_u must be in A_r also. Combining all this we have $h < i < i' < r$, $j'_v < j_u$, $(h, j_u) \in S$, $(r, j'_v) \in S$ and both (h, j'_v) and (r, j_u) non-blank, contradicting the independence of S.∎

By Corollary 2.7, there is a consistent set, $\{<_i\}$, of linear orderings which extend the set $\{\prec_i\}$ of partial orders. Let $M = (M_{ij})$ be the canonical partial matrix generated by the $\{<_i\}$. By Remark 2.5, M is a totally monotone. Thus M is an h-matrix with A as its structure matrix. Recall that for any function f from S to the positive reals, we define the matrix M^f by $M^f_{ij} = M_{ij}$ for $(i,j) \notin S$ and $M^f_{ij} = |S_i| + f(i,j)$.

Theorem 2.13. The partial matrix M^f is totally monotone.

Proof. Suppose $i < i', j < j'$ such that all entries of the 2×2 submatrix, $M^f_{ij}, M^f_{ij'}, M^f_{i'j}$, and $M^f_{i'j'}$, are non-blank, and $M^f_{ij} \leq M^f_{ij'}$. We must show that $M^f_{i'j} \leq M^f_{i'j'}$. There is nothing to prove if none of the indices are in S since M is totally monotone. Also note that if exactly one of indices in a row is in S then the relationship between the two entries in that row is the same in M and in M^f. It is not hard to see that this implies we may restrict our attention to the two cases $(i,j),(i,j') \in S$ and $(i',j),(i',j') \notin S$; and $(i,j),(i,j') \notin S$ and $(i',j),(i',j') \in S$. Suppose $(i,j),(i,j') \in S$ and $(i',j),(i',j') \notin S$. Then by the definition of $\alpha_{i'}$ we have $j \alpha_{i'} j'$ and hence $M_{i'j} < M_{i'j'}$. The argument for the other case is analogous. ∎

As we noted at the beginning of the proof for the h-matrix lower bound, given Theorem 2.13 the remainder of the proof of the lower bound for h-matrices can now be completed along entirely analogous lines as the proof for the v-matrix case.

3. The Matrix Searching Algorithm for Skyline Matrices

In this section we extend the almost linear time row-minima finding algorithm of [KK90] to skyline matrices. The algorithm can trivially be converted to a row-maxima finding algorithm for skyline matrices. In the case that a row has more than one entry with the minimum value, we follow the usual matrix searching convention of using the term row-minimum to mean the leftmost entry with the minimum value.

Recall that a **skyline** matrix is a v-matrix in which every non-blank column segment ends at the bottom row. The extension of the staircase algorithm follows from the following observation. Suppose that, given some particular type of partial totally monotone matrix, we choose a parameter t and let $q_c(t,n,m)$ denote the worst-case number of comparisions between matrix entries needed to find the row- minima of any of this type of partial matrix with at most n rows, m columns, and parameter value at most t. Then if this function $q_c(t,n,m)$ satisfies the three key propositions in [KK90], namely Lemma 2.1, Corollary 2.4 and Theorem 2.6, it will also satisfy Theorem 2.9, i.e. $q_c(n,n,m) = O(m\alpha(n) + n)$. Moreover, if as in [KK90] each of the

propositions is proved by means of an algorithm, these algorithms can be combined as in [KK90] to provide a row-minima finding algorithm using $O(m\alpha(n)+n)$ comparisons.

For simplicity and completeness we restate the three key propositions from [KK90]. As in [KK90] we define the functions $L_i(n)$ for $i = -1, 0, 1, 2, \ldots$ recursively as follows. $L_{-1}(n) = n/2$, and for $i \geq 0$, $L_i(n) = \min_s \{L_{i-1}^s(n) \leq 1\}$. Thus $L_0(n) = \lceil \log n \rceil$, $L_1(n)$ is essentially $\log^*(n)$, $L_2(n)$ is essentially $\log^{**}(n)$ etc. We now define $\alpha(n) = \min\{s : L_s(n) \leq s\}$.

Proposition 3.1 (Lemma 2.1 in [KK90]). For any positive integer a we have

$$q_c(n,n,m) \leq q_c(n/a,n,m) + O(am+n).$$

Proposition 3.2 (Corollary 2.4 in [KK90]). For any positive integer a we have

$$q_c(n,n,m) \leq q_c(n/a,n/a,m) + O(am+n).$$

Proposition 3.3 (Theorem 2.6 in [KK90]). There is a constant c_1 such that for $s \geq 0$ we have $q_c(n,n,m) \leq c_1(m+nL_s(n)) + \max\{\sum_{i=1}^k q_c(n_i/L_{s-1}(n_i), n_i, m_i) : \sum_{i=1}^k n_i \leq nL_s(n) \text{ and } \sum_{i=1}^k m_i \leq m+nL_s(n)\}$.

For M a skyline matrix, we will say that row i is a **top row** if it is the top row of some column's non-blank segment. A skyline matrix M is said to be of shape (t,n,m) if it has at most t top rows, at most n rows and at most m columns. We will denote the worst case number of comparisons between matrix entries needed to find the row-minima of a skyline matrix of this shape by $q_c(t,n,m)$. Proving that the three propositions above hold for this definition of the function q_c, provides an $O(m\alpha(n)+n)$ on the number of comparisons needed to find row-minima in skyline matrices. The proofs of Propositions 3.1 and 3.3 are very similar to those in [KK90] for staircase matrices, but the proof of 3.2 is a bit more subtle than the proof of the corresponding result for staircase matrices. Thus we sketch the proof of 3.1 to indicate how the arguments must be modified for skyline matrices, give a complete proof of 3.2, and omit the proof of 3.3.

In order to translate the proofs for staircase matrices to skyline matrices we need to give the appropriate definitions of various terms in this setting.

For each column j of M let $t(j)$ be the top row of the non-blank segment For each i, the i-th slice of M is the set of columns $\{j : t(j) = i\}$. in column j. For $i \leq k$ and $j \leq l$, we will use the notation $M[i,k;j,l]$ to denote the skyline matrix obtained by taking the intersection of rows $i,\ldots,k$ of M with columns $j,\ldots,l$ of M. For any positive integer a, we define the **stepsize a approximation** of M to be the submatrix of M obtained by, for each j with $t(j) \leq a\lfloor n/a \rfloor$, truncating the non-blank entries in column j so that the non-blank segment begins at row $a\lceil t(j)/a \rceil$, and for each j with $t(j) > a\lfloor n/a \rfloor$ replacing all the entries in column j with blanks. We denote the stepsize a approximation of M by M_a. An example is illustrated in Figure 3. Clearly M_a is a skyline matrix of shape $(\lfloor n/a \rfloor, n, m)$. We define the **$a$-border matrices of M**, which we denote by $M(a,i)$ for $i = 1,\ldots,\lceil n/a \rceil$, by $M(a,i)$ is the skyline matrix with at most $a-1$ rows whose non-blank entries are the non-blank entries of M in rows $(i-1)a+1,\ldots,\min(ia-1,n)$ which are blank in M_a. An example of a-border matrices is also illustrated in Figure 3. We will denote the set of a-border matrices of M by $\Gamma(M,a)$. It is easy to see that the skyline matrices $M_a, M(a,1),\ldots,M(a,\lceil n/a \rceil)$ disjointly cover the non-blank entries of M.

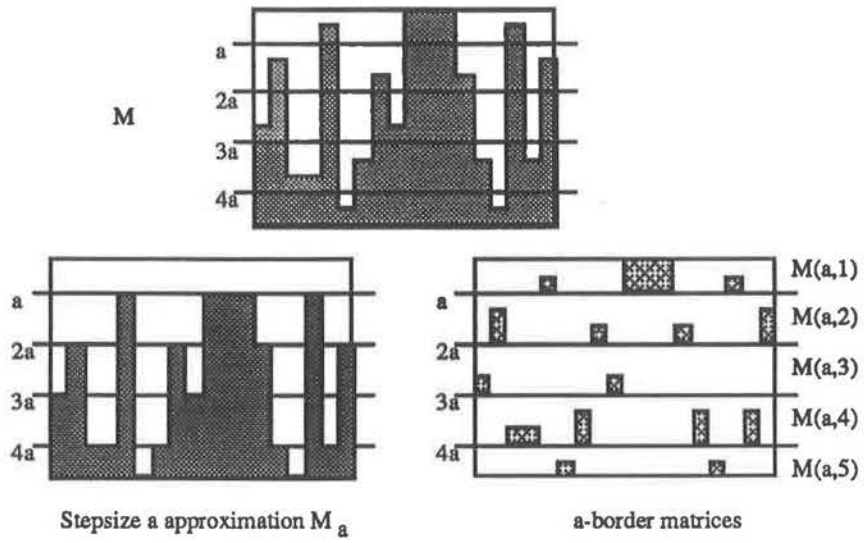


Figure 3

Given these definitions, the proof of the first proposition is identical to the proof of Lemma 2.1 in [KK90].

Proposition 3.1. For any positive integer a we have

$$q_c(n,n,m) \leq q_c(n/a,n,m) + O(am+n).$$

Proof. Let M be a skyline matrix of shape (n,n,m). Since the stepsize a approximation M_a is of shape $(n/a,n,m)$, it can be processed in $q_c(n/a,n,m)$ time. Since the total number of non-blank entries in the a-border matrices $M(a,i)$ for $i = 1,\ldots,\lfloor n/a \rfloor$, is less than am we can process these border matrices in $O(am)$ time. Finally in $O(n)$ comparisons we can compare the row-minima found in M_a with the row-minima found in the a-border matrices, and hence determine the row-minima of M. ∎

The proof of the second proposition requires a little more care than the corresponding proposition for staircase matrices.

Proposition 3.2. For any positive integer a we have

$$q_c(n,n,m) \leq q_c(n/a,n/a,m) + O(am+n).$$

Proof. Let N be a skyline matrix of shape (n,n,m). By the proof of Proposition 1 it suffices to show that finding the row- minima of the stepsize approximation matrix, N_a, can be reduced to finding the row-minima of a skyline matrix of shape $(n/a,n/a,m)$ in $O(m+n)$ time. Let $M = N_a$, and for each j let $t(j)$ be the row such that the non-blank segment of column j of M begins at row $t(j)$. Let S be the $\lfloor n/a \rfloor \times m$ skyline matrix where $S_{i,j} = M_{ai,j}$ if $t(j) \leq (i-1)a$ and is blank otherwise. Let $s(i)$ be the column containing the minimum value in row i of S, and let $d(i)$ such that $M_{ai,d(i)}$ is minimal among the $M_{ai,j}$ such that $t(j) = ai$.

Finally let $j(i)$ be the column containing the minimum value in row ai of N. Note that $j(i)$ must be either $s(i)$ or $d(i)$. Note that S is a skyline matrix of shape $(n/a, n/a, m)$. Thus it suffices to show that given the $\{s(i)\}$, the row minima of the rows of M_a can be found in $O(m+n)$ time.

We first note that we can find the $d(i)$ in at most $O(m)$ time since we only need to look at one entry in each column. Let $r = \lfloor n/a \rfloor$. For each $i = 1, \ldots, r+1$ let $J(i)$ be the set of columns $\{j : s(i) \leq j \leq j(i-1), t(j) \leq a(i-1)$, and $j \leq s(k)$ for each $k < i$ such that $t(j) \leq a(k-1)\}$, where we adopt the convention that $j(0) = m$ and $s(r+1) = 1$. Let $A(i)$ be the matrix consisting of the intersection of rows $a(i-1)+1, \ldots, \min(n, ai-1)$ of M with the columns in $J(i)$. An example is shown in Figure 4.

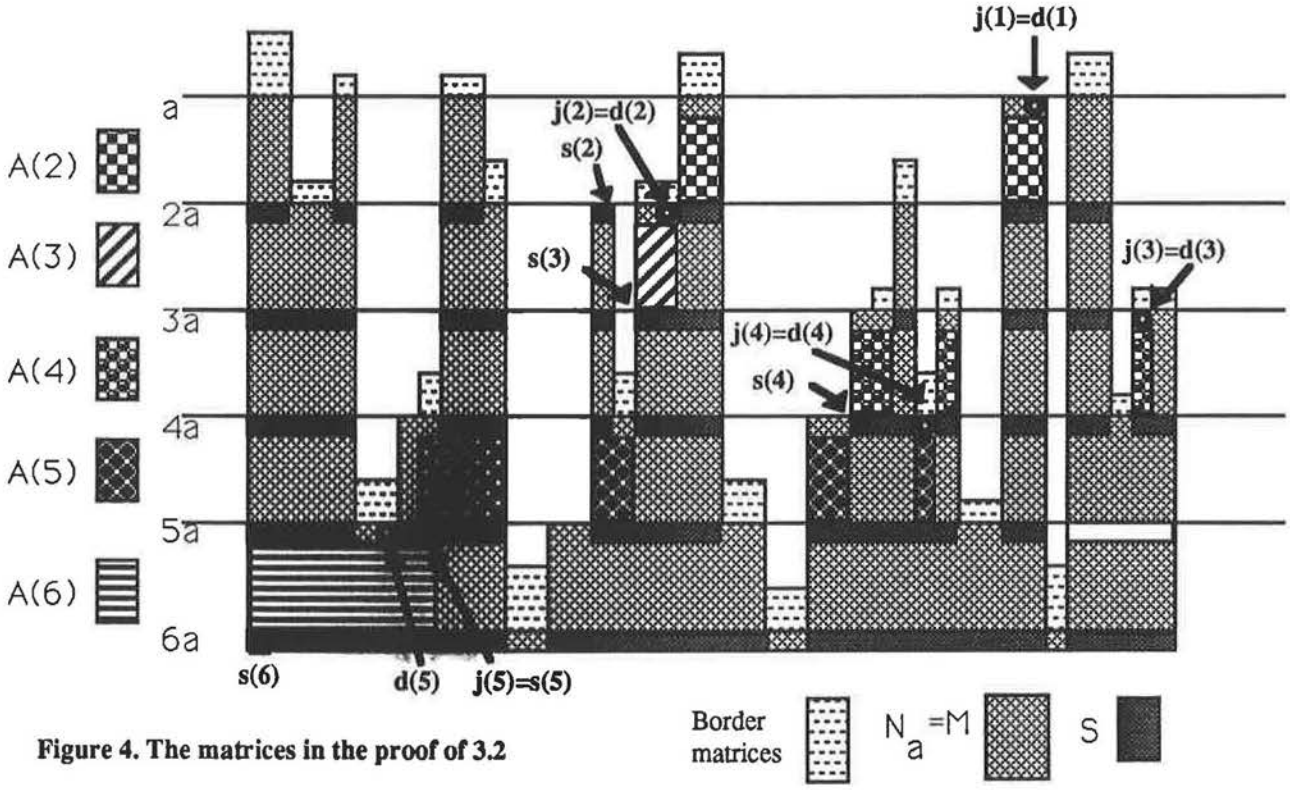


Figure 4. The matrices in the proof of 3.2

We claim that each $A(i)$ is a totally monotone matrix containing all the row minima in rows $a(i-1)+1, \ldots, \min(n, ai-1)$ of M, and $\sum_{i=1}^{r+1} |J(i)| \leq m + r + 1$. We now prove this claim. It is easy to see that each $A(i)$ is a totally monotone matrix since j in $J(i)$ implies that column j has no blank entries in rows $a(i-1)+1, \ldots, \min(n, ai-1)$. We now show that $A(i)$ contains all the row minima in rows $a(i-1)+1, \ldots, \min(n, ai-1)$ of M. In other words, for each j not in $J(i)$, and $a(i-1)+1 \leq h \leq \min(n, ai-1)$, we must show that $M_{h,j}$ is not a row minima. If $t(j) > a(i-1)$ then $t(j) \geq ai$ so $M_{h,j}$ is blank, and hence not a row minima. Thus we may assume $t(j) \leq a(i-1)$. Now by total monotonicity, if $j < s(i)$ then we must have $M_{h,j} > M_{h,s(i)}$ and if $j > j(i-1)$ then we must have $M_{h,j(i-1)} \leq M_{h,j}$, so in either case $M_{h,j}$ is not a (leftmost) row minima. Finally suppose $j > s(k)$ for some $k < i$ such that $t(j) \leq a(k-1)$. Again by total monotonicity we have $M_{h,j} \geq M_{h,s(k)}$ so again $M_{h,j}$ is not a row minima.

We now show that $\sum_{i=1}^{r+1} |J(i)| \leq m + r + 1$. It suffices to show that for $i < i'$, the sets $J(i) \backslash \{s(i)\}$ and

$J(i')$ are disjoint. Suppose j belongs to both $J(i)$ and $J(i')$. Then since $i < i'$ and $t(j) \leq a(i-1)$ because j is in $J(i)$, we must have $j \leq s(i)$ because j is in $J(i')$. However j in $J(i)$ implies $j \geq s(i)$ and hence $j = s(i)$.

The proof is completed by observing that applying the linear time SMAWK algorithm of [AKMSW90] to find the row minima of the $A(i)$ requires at most $O(n + \sum_{i=1}^{r+1} |J(i)|) = O(m+n)$ time. ∎

4. Data-Structures and the On-line Version

As in [KK90] the proofs of the three propositions in the preceding section actually provide a row-minima finding algorithm for skyline matrices which uses $O(m\alpha(n)+n)$ comparisons between matrix entries. However, in order to bound the total running time of this algorithm, we need to consider the time needed by the algorithm to initialize and maintain appropriate data-structures. In [KK90] it is shown that for staircase matrices this can be done in time linear in the number of comparisions, thus yielding an $O(m\alpha(n)+n)$ bound on the total running time. Unfortunately the situation for skyline matrices seems to be more complicated, and so far the best bound on total running time that we have been able to achieve is $O(m \log\log n + n)$ using a simpler row-minima finding algorithm for which the bound on the number of comparisions is also $O(m \log\log n + n)$. We now sketch how this bound is obtained, beginning with a description of the data-structures used. At the end of this section we sketch how to modify this simpler row-minima finding algorithm so that it satisfies the on-line constraint needed for dynamic programming applications, while retaining the $O(m \log\log n + n)$ bound on the total running time. By modifying the more complicated algorithm, one can obtain an on-line algorithm using $O(m\alpha(n)+n)$ comparisons, but the best total running time we can achieve for that algorithm at this point is only $O(m\alpha(n)\log\log n + n)$.

The data-structures used are fairly similar to those described in [KK90] for staircase matrices. We represent each skyline matrix as a sub-skyline matrix of some large totally monotone matrix U, using the following set of data-structures.

Integer variables $\#ROW_M$ and $\#COL_M$ containing the number of rows and columns respectively.

An array, R_M, of length $\#ROW_M$ where $R_M[i]$ contains the number of the row of U which contains the i-th row of M, and a flag indicating whether or not the i-th row of M is a top row of M.

An array C_M of length $\#COL_M$ which contains the column information for M. More precisely, the i-th item in C_M contains the number of the column of U which is the i-th column of M and the number $t(i)$ of its top-row. For simplicity we will abuse our notation and use $C_M[i]$ to denote the number of the column of U which is the i-th column of M. We will always have $C_M[1] < \ldots < C_M[m]$, where m is the number of columns of M.

A set of doubly-linked lists linking together, in left-to-right order, the columns with the same top-row. We will refer to the linked list of columns with $t(i) = j$ as the j-th top-row list.

A pointer array T_M of length $\#ROW_M$ with $T_M[j]$ containing a pointer to the first item in the j-th top-row list if j is a top-row , and null otherwise.

The output information will be stored in a pointer array of length $\#ROW_M$ named MIN_M. For each i, the entry $MIN_M[i]$ will contain a pointer to the $j(i)$-th item of C_M, where the $j(i)$-th column of M contains the minimum value of the i-th row of M.

The main problems which arise in trying to maintain the data-structures during the operation of the algorithm are as follows. First, when we form border matrices, we can use the top-row lists to determine which columns are in each border matrix, but we will need to resort the sets of columns in each border matrix to obtain the correct ordering for the column array. This can be done in time linear in the number of rows and columns if we first determine to which border matrix of each column will belong, and then bucket sort the columns (taken in their left to right order) into buckets according to border matrix. For on-line variants it may or may not be possible to do this, depending on whether the structure of the matrix (i.e. the values of top-rows) is specified as part of the initial input. For the case of the algorithm below, we use a multi-radix sort since we will be able to assume that the number of columns in each border matrix is bounded by a polynomial in its number of rows. A similar problem arises in constructing the top-row lists for stepsize approximation matrices, but can be handled in the same manner.

A more serious problem involves how to determine the sets of columns, $J(i)$, needed in the algorithm specified in the proof of Proposition 3.2. The best approach we know of involves the use of priority queues, and would require $O(m \log\log n + n)$ time to obtain the $J(i)$, resulting in a total running time of $O(m\alpha(n)\log\log n + n)$. In order get rid of the $\alpha(n)$ factor in the bound on the running time, we revert to a simpler algorithm with a bound of $O(m \log\log n + n)$ on the number of comparisons made, and show that for this algorithm the time needed to initialize and maintain the data-structures is of the same order as the number of comparisons made. The algorithm we use is essentially the skyline variant of the $O(m \log\log n+n)$ algorithm for staircase matrices given in [AK90], in the same sense as the algorithm discussed in section 3 is the skyline variant of the staircase algorithm in [KK90]. The algorithm is obtained by applying the algorithm in the proof of the following proposition recursively. Let $q(n, n, m)$ denote the total time needed to find the row-minima of a skyline matrix of shape (n, n, m).

Proposition 4.1. We have $q(n, n, m) \leq O(m+n) + \sum_{1 \leq i \leq \sqrt{n}+2} q(\sqrt{n}, \sqrt{n}, m_i)$ where $\sum_{1 \leq i \leq \sqrt{n}+2} m_i \leq m$.

Proof. Let N be a skyline matrix of shape (n, n, m). We first note that if $m \geq n^2$ then we easily can find the row minima of N in $O(m + n)$ time. This is done by partitioning the columns of N according to their top-row, yielding at most n rectangular matrices. Now we apply SMAWK to each of these rectangular matrices to find their row-minima and finally obtain the row-minima of N by comparing the row-minima from the rectangular sub-matrices. Since the columns in the sub-matrices are disjoint the total time is $O(n^2 + m) = O(m)$.

Thus we may assume $m \leq n^2$. For this case the algorithm is basically similar to that in the proof of Proposition 3.2. Let $a = \lfloor\sqrt{n}\rfloor$. Given the row-minima of the a-border matrices and of the stepsize approximation matrix, N_a, in $O(n)$ time we can determine the row-minima of N. It is straightforward to create the data-structures for the a-border matrices in $O(m + n)$ time from the data-structures of N except for the problem noted above of obtaining the left-to-right order of the columns in each border matrix. However since $m \leq n^2$ we can use an $O(\sqrt{n})$ time multi-radix sorting algorithm to sort the columns of each

border matrix according to their position in N. Similarly the data-structures of N_a can be created from the data-structures for N in $O(m+n)$ time, using a multi-radix sort to obtain the top-row lists.

Since each a-border matrix is of shape $(\sqrt{n}, \sqrt{n}, m_i)$, since there are at most $\sqrt{n}+2$ of them, and since their columns are disjoint, it suffices to show that the total time needed to find the row-minima of N_a is $O(m+n)$. Let M, $t(j)$, S, $s(i)$, $j(i)$, $J(i)$, and $A(i)$ be as in the proof of 3.2. As we noted in proving 3.2, given the $s(i)$ the total additional time needed to find the $j(i)$ is $O(m+n)$. Moreover, given $A(i)$ and the appropriate data- structures we can use SMAWK to find the row-minima of the $A(i)$ and hence of M in $O(m+n)$ time. As described in [AK90], the necessary data-structure for applying SMAWK to each $A(i)$ is a doubly-linked list of the columns of $A(i)$ in left-to-right order. Since we can also use a multi-radix sort to get the left-to-right order of the columns of the $A(i)$, it suffices to show how to obtain the sets of columns, namely the $J(i)$. Thus we must show how to find the $s(i)$ in $O(m+n)$ time, and given the $s(i)$ and $j(i)$, how to obtain the $J(i)$ in $O(m+n)$ time.

It is trivial to construct the data-structures for S from the data- structures for $M = N_a$ in $O(m+n)$ time. Moreover, finding the $s(i)$ is easy since like the case where $m \geq n^2$ we can partition S into $O(\sqrt{n})$ column-disjoint totally monotone matrices (i.e. according to their top-row) and apply SMAWK to each of these matrices. Now for each row of S we compare the $O(\sqrt{n})$ possible candidates for row- minima to find the minimum value. Since S has $O(\sqrt{n})$ rows the total time used is $O(n+m)$.

Thus we have reduced the problem to the final task of producing the $J(i)$ given the data-structures for M and pointers to the $s(i)$ and the $j(i)$ for each i. Recall that $J(i) = \{j : s(i) \leq j \leq j(i-1), t(j) \leq a(i-1),$ and $j \leq s(k)$ for each $k < i$ such that $t(j) \leq a(k-1)\}$, and let $TR(k) = \{j : t(j) = a(k-1)\}$. Thus $T(k)$ is the set of columns with top-row equal to k in S. Let $J(i)_k = J(i) \cap TR(k)$. We have $J(i) = \bigcap_{2 \leq k \leq i} J(i)_k$, and we obtain $J(i)$ in increasing order of i by examining the top-row lists of S to find the members of each $J(i)_k$ for $2 \leq k \leq i$. In order to do this efficiently we keep track of the variable $s^*(i) = \min\{s(j) : 1 \leq j < i\}$ and maintain a pointer into each of the top-row lists of S to the largest numbered (i.e. rightmost) column in each top-row list which is smaller than (i.e. to the left of) $s^*(i)$. In order to find the columns in $J(i)_k$ we begin at the column pointed to in the k-th top-row list and walk back, inserting columns into $J(i)_k$ until we encounter a column with a smaller index than $s(i)$. We then reset the pointer to that column. This procedure may take $O(\sqrt{n}+m)$ time for the cases $i = 1, 2$ because of the need to set the pointers into the top-row lists, but for each $i > 2$ it is easy to see that the time used to create $J(i)$ is $O(\sqrt{n} + |J(i)|$. Since $\sum_i |J(i)| = O(m+n)$ as proved in 3.2, the total time is thus $O(m+n)$ as desired.∎

We now turn to defining an on-line version of matrix searching for skyline matrices that is needed for dynamic programming applications [AP90].

For each j with $1 \leq j \leq m$ let $b(j) = \max\{0, t(i) : 1 \leq i \leq n, t(i) < t(j)\}$. We will call row $b(j)$ the **foundation row** for column j. . Note that if we adopt the convention that the 0-th row is a top row which contains only blanks, then the foundation row for a column is the highest numbered top row with which the column's intersection is blank. Recall that the i-th slice is the set of columns $\{j : t(j) = i\}$. Note that the i-th slice is non-empty if and only if row i is a top row. We will call a row-minima finding algorithm for skyline matrices **on-line** if it satisfies the following constraint:
For each j with $b(j) > 0$, the algorithm always determines the minima of rows $1, \ldots, b(j)$ before querying

values of the non-blank entries in column j.

In general we assume that an on-line algorithm does have the information specifying the structure matrix at the beginning of the computation, and so the only restriction is on the order of evaluation of entries in the skyline matrix. This assumption is consistent with the requirements of the applications to dynamic programming.

In [KK90] it was shown how the $O(m\alpha(n)+n)$ algorithm could be modified to satisfy the on-line constraint above, but in order to make the modification work it was necessary to prove the algorithm satisfied an additional constraint. Klawe and Kleitman referred to an on-line row-minima finding algorithm for skyline matrices as **ordered** if it also satisfied the constraint that it used no information about the columns in the i-th slice until after the minimum in row $i-1$ had been found. Thus when an ordered algorithm begins processing the staircase matrix M, the only initial information it has is the set of rows of M (i.e. the information in R_M), and the column information for the first slice. The techniques from [KK90] can be used to modify the skyline algorithm discussed in section 3 to obtain an ordered skyline algorithm using $O(m\alpha(n) + n)$ comparisons, but since the bound on total running time for this algorithm is worse, we concentrate on showing how to modify the simpler algorithm of this section to obtain an on-line algorithm with $O(m \log\log n + n)$ total running time. Since the ideas are also largely based on [KK90], we merely provide a sketch, emphasizing the points where the skyline case needs more care.

Let us call the algorithm obtained by recursively applying the procedure described in the proof of 4.1 the AK algorithm. In order to make the AK algorithm an on-line algorithm we must change the structure of its recursive calls. Let us refer to the finding of the row-minima of a skyline matrix as the processing of that skyline matrix. In order to convert the AK algorithm into an on-line algorithm we must interleave the processing of S (i.e. finding the $s(i)$ and the $(j(i))$ with the recursive calls to process the border matrices, and the processing of the $A(i)$ by the SMAWK algorithm. It is fairly straightforward to see how to do this, but it may be helpful to examine the example shown in Figure 4 while following the discussion below.

We begin by running the on-line version of the AK-algorithm on the top border matrix. After this has finished we examine the appropriate entries in row a of N to find $d(1) = j(1)$. We can do this since at this point we have found all the row minima for rows less than a. We next start processing S to find $s(2)$. Specifically we run SMAWK on the rectangular matrix consisting of the columns in S which have $2a$ as their top row. Since each of these has its top row in N less than or equal to a, its foundation row is less than a and hence we may evaluate any entry in this rectangular matrix at this time. After finding $s(2)$, we can obtain the set $J(2)$, and hence the matrix $A(2)$. Since each column in $J(2)$ has its foundation row less than a we can evaluate any entry in $A(2)$ at this point, and hence can use SMAWK to find its row minima. Next we process the border matrix which is second from the top using a slight modification of the on-line version of the AK-algorithm. The modification is that whenever we have computed the minimum value of a row in this second border matrix, we immediately compare it with the corresponding minimum for $A(2)$ to determine the minimum value for that row in N. We are now ready to determine $d(2)$ and hence $j(2)$, then continue processing S to find $s(3)$, then $A(3)$, then the third border matrix, etc. We continue in this way until we have determined all the row-minima of N.

5. Open Problems

There are many interesting problems in matrix searching which remain open (see [AP88] for example). In this section we restrict ourselves to problems related to upper and lower bounds for matrix searching in partial matrices. The first obvious group of problems concerns closing the gap between the current upper and lower bounds for the partial matrices discussed in this paper. Specifically for staircase and skyline matrices we have $O(m\alpha(n)+n)$ upper bounds ([KK90] and this paper respectively) for the number of evaluations made in searching matrices of size $n \times m$ but only linear lower bounds. For skyline matrices, it would be interesting to find an algorithm with an $O(m\alpha(n)+n)$ bound on the total running time (as instead of on the number of comparisons). For v-matrices and h- matrices there are fairly straightforward $O(m \log n + n)$ upper bounds [AS89] and lower bounds of $\Omega(n\alpha(n))$ (this paper). It would be interesting to improve these lower bounds to $\Omega(m\alpha(n))$ for the case $m > n$. Another problem which seems to be difficult is to find a better upper bound for horizontal skyline matrices, i.e. h- matrices in which each row's non-blank segment starts in the first column.

A completely different direction involves Monge matrices [AP88]. These are matrices which satisfy the condition, for every $i < i', j < j'$ such that all entries of the 2×2 submatrix, $M_{ij}, M_{ij'}, M_{i'j}$, and $M_{i'j'}$, are non-blank, we have $M_{ij} + M_{i'j'} \geq M_{ij'} + M_{i'j}$. It is easy to see that Monge implies totally monotone but the reverse is not true. In most applications of totally monotone matrices, the matrix in question is actually Monge, so it would be worthwhile to get a superlinear lower bound for matrix searching of Monge matrices.

Finally, there are a number of open problems concerning the techniques used to prove the lower bound for h-matrices. First, is it possible to find a simpler construction of the matrix M directly from the line segments and their left envelope as was done in the v-matrix case? Next, for any structure matrix A with an independence set S, one can define the relations $\{\propto_i\}$ as was done in section 2. It is easy to find structure matrices in which some of the $\{\prec_i\}$ are not partial orders. It seems natural to try to characterize the family of structure matrices for which $\{\prec_i\}$ is a consistent set of partial orders. This paper proves that structure h-matrices have this property, and we believe that a similar proof can be given for structure v-matrices though it seems to be slightly more difficult. We conjecture that in fact this will hold for any structure matrix in which for every non-blank entry, the set of non-blank entries in either its row or column form a contiguous segment.

References

[AKMSW87] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric applications of a matrix searching algorithm, Algorithmica 2(1987), pp. 195-208.

[AK90] A. Aggarwal and M. Klawe, Applications of generalized matrix searching to geometric algorithms, Discrete Applied Math 27(1990), pp.3-23.

[AP88] A. Aggarwal and J. Park, Notes on searching in multidimensional arrays, Proc. 29th Ann. IEEE Symposium on Found. Comp. Sci. (1988), pp.497-512.

[AP90] A. Aggarwal and J. Park, Improved Algorithms for Economic Lot- Size Problems, submitted to the Journal of Operations Research.

[AS87] A. Aggarwal and S. Suri, Fast algorithms for computing the largest empty rectangle, Proc. 3rd Ann. Symp. Comp. Geom.(1987), pp.278-290.

[AS89] A. Aggarwal and S. Suri, Computing the farthest visible pair in a simple polygon, preprint.

[E88] D. Eppstein, Sequence comparison with mixed convex and concave costs, to appear in J. Algorithms.

[EGG88] D. Eppstein, Z. Galil and R. Giancarlo, Speeding up dynamic programming, Proc. 29th Ann. IEEE Symposium on Found. Comp. Sci. (1988), pp.488-496.

[EGGI90] D. Eppstein, Z. Galil, R. Giancarlo and G. Italiano, Sparse dynamic programming, Proceedings of the First ACM/SIAM Symposium on Discrete Algorithms, 1990, pp. 513-522.

[GG89] Z. Galil and R. Giancarlo, Speeding up dynamic programming with applications to molecular biology, Theor. Comp. Sci. 64 (1989), pp. 107- 118.

[GP90] Z. Galil and K. Park, A linear time algorithm for concave one- dimensional programming, to appear in IPL.

[HS86] S.Hart and M.Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, Combinatorica 6 (1986), pp. 151-177.

[HL87] D.S. Hirschberg and L.L. Larmore, The least weight subsequence problem,SIAM J. Computing 16, (1987), pp. 628-638.

[KK90] M. Klawe and D.J. Kleitman, An almost linear time algorithm for generalized matrix searching, SIAM J. Discrete Math., Vol. 3, No. 1(1990) pp. 81-97.

[LS90] L. Larmore and B. Schieber, On-line dynamic programming with applications to the prediction of RNA secondary structure, Proceedings of the First ACM/SIAM Symposium on Discrete Algorithms, 1990, pp. 503- 512.

[S89] P. Shor, Geometric Realizations of Superlinear Davenport-Schinzel Sequences, preprint 1989.

[T75] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, JACM 22 (1975), pp. 215-225.

[W86] A. Wiernik, Planar realizations of nonlinear Davenport-Schinzel sequences by segments, Proc. 27th Ann. IEEE Symposium on Found. Comp. Sci. (1986), pp. 97-106.

[W88] R. Wilber, The concave least weight subsequence revisited, J. Algorithms 9 (1988), pp.418-425.