Lower Bounds for Row Minima Searching

(Extended Abstract)

Phillip G. Bradford and Knut Reinert

Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany.

{ bradford, kreinert }@mpi-sb.mpg.de

Abstract. This paper shows that finding the row minima (maxima) in an $n \times n$ totally monotone matrix in the worst case requires any algorithm to make $3n-5$ comparisons or $4n-5$ matrix accesses. Where the, so called, SMAWK algorithm of Aggarwal *et al.* finds the row minima in no more than $5n - 2\lg n - 6$ comparisons.

1 Introduction

Finding the row minima (maxima) of *monotone matrices* was introduced by Aggarwal, Klawe, Moran, Shor, and Wilber [1]. They also gave an asymptotically optimal sequential algorithm for finding the row minima in *totally monotone matrices*, among other things. Row minima problems and their variants are well motivated due to their large host of applications. Two examples are the prediction of RNA secondary structure [6] and the all–farthest–neighbor problem [1].

The *row minima problem* for an $n \times n$ matrix M, whose entries belong to some totally ordered set, is to find the minimal element in each row. To be more precise, let $\mathsf{mc}(i)$ denote the index of the leftmost column that contains a minimal element of the i-th row. Then, the row minima problem is to find $\mathsf{mc}(i)$ for $1 \leq i \leq n$.

Throughout this paper, we assume that all entries of M are distinct. Clearly, the row minima problem has time complexity $\Theta(n^2)$. It turns out, however, that many problems can be reduced to the row minima problem for a matrix M that has a special form:

Definition 1. Let M be an $n \times n$ matrix. The matrix M is *monotone* if for all $1 \leq i < j \leq n$, $\mathsf{mc}(i) \leq \mathsf{mc}(j)$.

Given an $n \times n$ monotone matrix M of n^2 distinct elements, the *monotone row minima* problem is to get the minimal value in each row of M. Aggarwal *et al.* [1] gave an $O(n \log n)$ sequential algorithm for the row minima problem on a monotone matrix and showed that it is asymptotically optimal.

Definition 2. An $n \times n$ matrix M is *totally monotone* if every 2×2 minor is monotone. That is, for all $1 \leq i < k \leq n$ and $1 \leq j < l \leq n$, if $M[i,j] > M[i,l]$, then $M[k,j] > M[k,l]$.

Given a $n \times n$ *totally* monotone matrix M the *totally monotone row minima* problem, or just the *row minima* problem, is to get the minimal value in each row of M. (The row minima and row maxima problems on totally monotone matrices are symmetric, so our exposition only deals with the row minima problem.)

Aggarwal *et al.* [1] gave a sequential $O(n)$-time algorithm for the row minima problem on a totally monotone matrix. This is the SMAWK algorithm.

The SMAWK algorithm illustrates that we do not need to generate the entire $n \times n$ totally monotone matrix M, but rather we only have to compute selected entries of M when they are needed. In this paper, we attack the natural extension to this: Exactly how many matrix elements must we generate to solve the row minima problem on a totally monotone matrix?

An exact analysis by Larmore and Schieber [6] shows that Aggarwal *et al.*'s algorithm takes at most $5n$ comparisons and Larmore [5] very recently strengthened this to $5n - 2\log n - 6$ comparisons. We show that any algorithm needs at least $3n - 5$ comparisons among totally monotone matrix elements. We also give a $4n - 5$ matrix access lower bound. So any algorithm that solves the row minima problem on a totally monotone matrix must 'look at' $4n - 5$ matrix elements.

Previous lower bounds for a variation of row minima problems were given by Aggarwal *et al.* [1], who gave an $\Omega(n \log n)$ lower bound for the more general row minima problem on *monotone matrices* and Klawe [3], who gave a $\Omega(n\alpha(n))$ lower bound for the row minima problem on *partial* totally monotone matrices, where $\alpha(n)$ is the inverse Ackermann function. Larmore [4] gives a provably optimal algorithm with unknown time complexity for the *convex strictly* monotone triangular matrix searching problem. Lower bounds for related problems have also been considered. Alon and Azar [2] show that the decision tree complexity of sorting rows in totally monotone matrices is $\Omega(n^2)$. However, to our knowledge there have been no exact lower bounds for the row minima problem on a *totally* monotone matrix.

2 The Basics of Totally Monotone Matrices

For the rest of this paper assume all matrices are totally monotone. A *dead* column is a column that has been shown not to contain any row minima through only comparisons and inferences [1]. Similarly, a dead *cell* is a single matrix element in a totally monotone matrix that has been shown to contain no row minima. Showing a matrix element is dead is *killing* a matrix cell. The left side of Figure 1 shows a dead column in a totally monotone matrix.

In the left side of Figure 1, we have $A < A'$; therefore, due to the total monotonicity of M we do not have to consider any elements directly above A' as a candidate row minima. Likewise, in the left side of Figure 1, we have $B' > B$; therefore, due to the total monotonicity of M we don't have to consider any elements directly below B' as candidate row minima. That is:

Lemma 3 (Aggarwal *et al.* [1]). If M is a totally monotone matrix with the situation depicted in the left of Figure 1, that is $A < A'$ and $B' > B$, then the shaded column is dead.

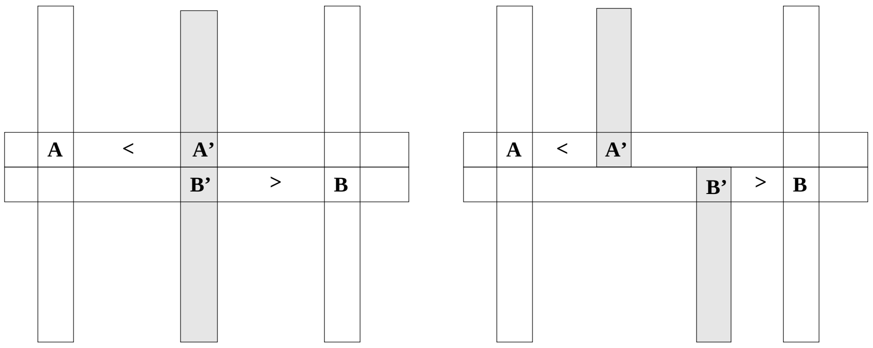

Fig. 1. Left: The Middle Column is Dead Since it Can't Contain any Row Minima in a Totally Monotone Matrix. Right: Two half-dead columns.

This lemma lies at the foundation of the SMAWK algorithm.

The right side of Figure 1 shows two *half-dead* columns. More formally, given a matrix M a column c is half-dead column from row r up to row 1 iff there is a column $c' < c$ such that $M[i, c'] < M[i, c]$, for $i : r \geq i \geq 1$. Symmetrically, a column c is half-dead column from row r down to row n iff there is a column c' where $c < c'$ is such that $M[i, c] > M[i, c']$, for $i : n \leq i \leq r$.

3 The Lower Bound

In this section we give our main result: a $3n - 5$ lower bound on the number of comparisons needed for finding the row minima of a totally monotone matrix. We begin with a trivial $2n - 2$ lower bound to find a certificate to verify a solution to the row minima problem, then this is extended giving our $3n - 5$ lower bound.

We use an adversary based argument. The adversary fixes the input for any algorithm. Of course, the input must be consistent, in that it must represent an actual $n \times n$ totally monotone matrix. The adversary starts by choosing the input such that every value in row i is smaller than the smallest value in row $i + 1$. This is so comparisons of elements between different rows cannot help any algorithm. This is done without loss of generality since choosing the values in different rows of M as just stated is consonant with Definition 2.

Given an $n \times n$ totally monotone matrix M, a *certificate* for the row minima problem is a set of comparisons that indicate that all but n cells in M are dead. Certainly, finding the actual solution of the row minima problem is at least as hard as finding a certificate for a solution. Hence we will show that our adversary can make it costly for any algorithm to find a certificate.

3.1 Foundations for the Adversary

First we introduce some terminology which we illustrate in Figure 2. Along the main diagonal we have a band of width seven. The cells in the matrix which are

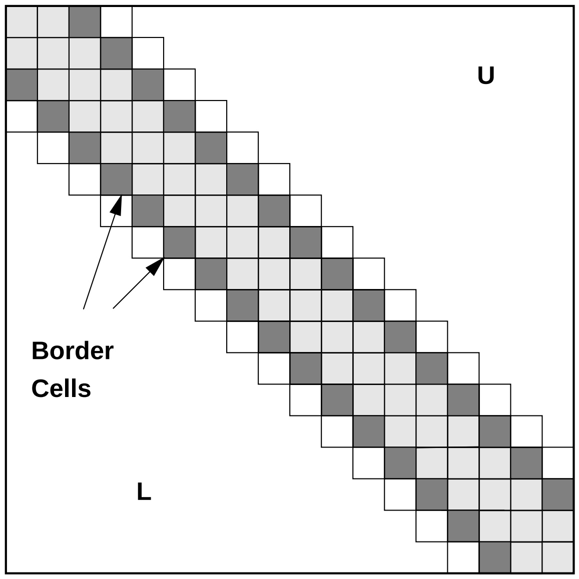

Fig. 2. A Banded Matrix with (2×2)–Boxes

to the upper right of the band are *U-cells*. Similarly the cells which are to the lower left of the band are *L-cells*. Cells within the band are *B-cells*. Killing a border cell for the first time is a *border kill*. Along the main diagonal there are intersecting (2×2)*–Boxes* which play a crucial role in our argument. Cells in (2×2)–Boxes are (2×2)*–Box-cells*. The intersecting (2×2)–Boxes in Figure 2 are in *canonical* form, since their diagonal is the same as the diagonal of M and they share one (2×2)–Box-cell. If a comparison kills a cell in a (2×2)–Box, then this comparison is a (2×2)*–Box-kill*. We imagine that each row minimum is always in some (2×2)–Box, though the (2×2)–Boxes can be moved up or down one row by the adversary to make more comparisons necessary.

The cells lying in the band but bordering all of the intersecting (2×2)–Boxes are *border cells*. The border cells are the darkest cells in the band. If a (2×2)–Box is moved up or down one row, then 'its' border cells move with it. The (2×2)–Boxes will always remain in this band of width seven.

Figure 3 shows our adversary's association of border cells with (2×2)-Box-cells. This association will turn out to be central for our adversary.

Two (2×2)–Boxes are *intersecting* if they both share one or two matrix elements. Two (2×2)–Boxes are *neighboring* if one is above the other and they share at least one matrix cell border. A *connected monotone falling chain* or a *chain* is a list of intersecting or neighboring (2×2)–Boxes that include both cells $M[1,1]$ and $M[n,n]$ and in this list of (2×2)–Boxes there is a path of (2×2)–Box-cells from $M[1,1]$ down to $M[n,n]$.

Note that the chain may contain dead cells.

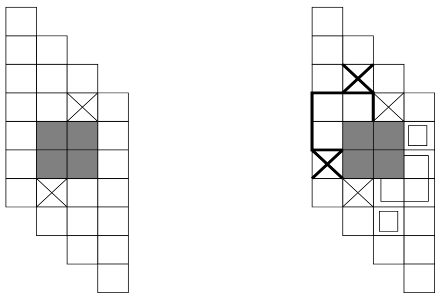

Fig. 3. Some (2×2)–Boxes and their associated border cells. Two intersecting (2×2)–Boxes never have coinciding border cells since the adversary only moves (2×2)–Boxes up or down.

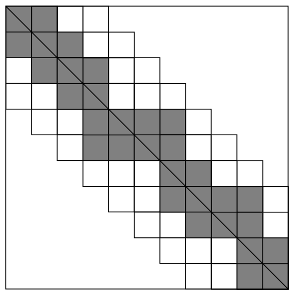

Fig. 4. The grey cells form a *chain* of (2×2)–Boxes. Such a chain would only come to light after an algorithm makes certain comparisons.

Lemma 4. For a $n\times n$ totally monotone matrix, there are exactly $3n-2$ (2×2)–Box-cells in any chain of (2×2)–Boxes.

The basic intuition behind our adversary is that each (2×2)–Box alone requires in the worst case two comparisons to certify its row minima. A $(2\times b)$–Box is a sub-matrix with 2 rows and b columns. A $(2\times b)$–Box requires in the worst case $2(b-1)$ comparisons to certify its row minima. That is, in the worst case monotonicity does not help to find the minima in a $(2\times b)$–Box.

In canonical form (see Figure 2) even though the (2×2)–Boxes intersect, their intersection does not diminish the number of comparisons necessary for certifying minima in them. Since there are initially $n-1$ of these (2×2)–Boxes so at the start when they run right down the diagonal we get a $2n-2$ cost for the entire certificate. However, we will see that we can charge to almost every (2×2)–Box a third comparison via moving some of the (2×2)–Boxes up or down. In particular, if comparisons are only done among matrix elements in the chain in canonical form, then a simple adversary can keep many border cells (hence half-columns) alive that may potentially contain row minima. Furthermore, if we can move (2×2)–Boxes in the canonical chain up or down one row depending on the comparisons of an algorithm, then some of those comparisons in the original canonical chain will have been for naught while at the same time we still have to check for row minima in the moved (2×2)–Boxes. However, moving (2×2)–Boxes up or down can form a variety of 'box-types' in the chain. The adversary must ensure that these box-types have worst-case 'sufficiently expensive' certificates. Suppose it takes t comparisons to create a box-type T consisting of u (2×2)–Boxes. Assume these (2×2)–Boxes contain no dead cells. Then the adversary wants to make sure that the worst case comparison cost of finding a certificate in T plus t is at least $3u$.

The basic structure of the argument in the rest of the paper is the following. If no comparisons are made among elements in the chain and if elements of the chain are not killed as part of a dead-column, then the row minima are unknown. Suppose some comparisons are made between elements in the chain, then usually the adversary can move (2×2)–Boxes to avoid the effect of these comparisons. The moved (2×2)–Boxes creates certain box-types down the chain. Finally we show that the adversary can make all of these box-types costly enough to certify to complete the result claimed in this paper.

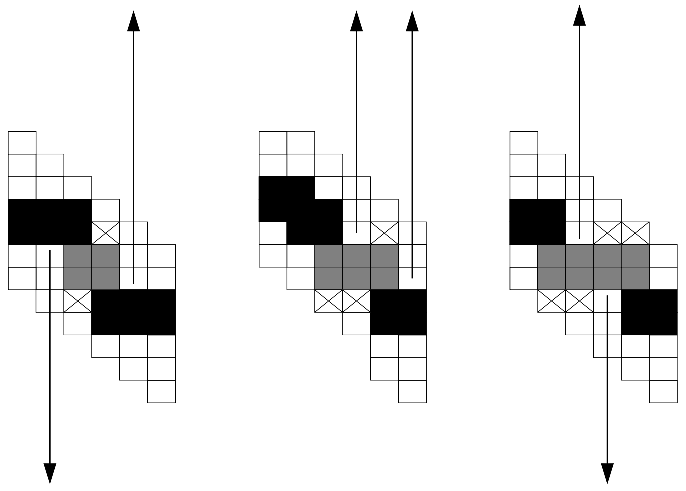

Fig. 5. The three different *isolated* box-types ((2×2)–Box, the (2×3)–Box and the (2×4)–Box) shaded in grey, with the arrows denoting the necessary half-dead columns to create these box-types and the X's denote the border cells associated with each of these $(2 \times b)$–Boxes. The black boxes are neighboring boxes in the chain.

During the run of any algorithm, there are six basic box-types that can emerge, the first three of these are depicted in Figure 5. These are *isolated* $(2 \times b)$–Boxes for $b \in \{2, 3, 4\}$. An isolated $(2 \times b)$–Box is a $(2 \times b)$–Box with no intersecting (2×2)–Box-cells from any other $(2 \times c)$–Boxes above or below. Recall that to verify a certificate in an isolated $(2 \times b)$–Box requires $2(b-1)$ comparisons in the worst case. However, to verify a certificate in a $(3 \times b)$–Box requires less than $3(b-1)$ comparisons in the worst case.

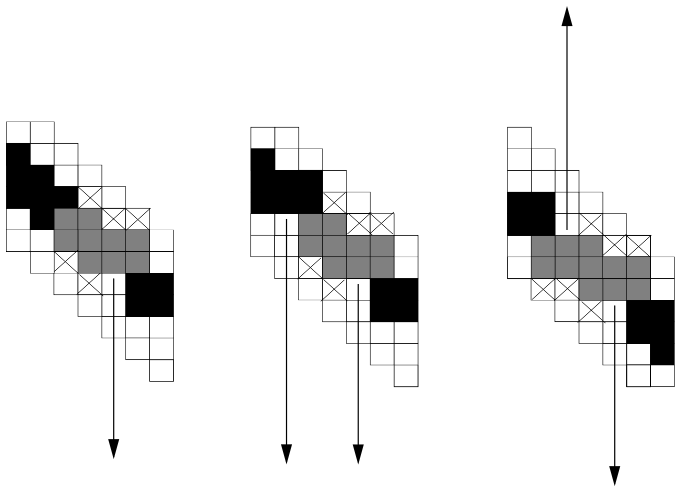

Fig. 6. The three different *non*-isolated box-types shaded in grey, with the arrows denoting the necessary half-dead columns to create these box-types. (We omit the symmetric cases.)

The three *non*-isolated box-types (without their symmetric counterparts) are in Figure 6. Non-isolated box-types consist of intersecting $(2 \times b)$–Boxes and $(2 \times c)$–Boxes. Whenever we write about *non*-isolated box-types we also mean to include their symmetric counterparts. Since non-isolated box-types 'fit inside' $(3 \times b)$–Boxes for some $b \in \{4, 5\}$ the adversary must take special care of these since finding the row minima of the middle row makes finding the row minima of the top and bottom row cheaper, even in the worst case.

Lemma 5. The isolated, non-isolated, and canonical box-types are the only box-types that can form by maintaining a chain via moving (2×2)–Boxes up or down one row.

The proof of Lemma 5 follows from the fact that adversary only moves a (2×2)–Boxes up or down one row at most.

3.2 Details of the Adversary

The adversary gives answers to an algorithm when it makes a comparison of two elements $x = M[i, j]$ and $y = M[i, k]$ from cells in row i, where $j < k$ and x and y are in either in the matrix regions U, L, or B. The adversary starts with the configuration of Figure 2 when all of the (2×2)–Boxes are in canonical form.

In Figure 7 we listed all but the $B : B$ comparisons since they are more complicated. Without loss of generality, take $B : B$ comparisons to be comparisons of two elements in the chain. Given our adversary, any comparison numbered 1 through 5 in Figure 7 can only increase the worst cost of any algorithm.

Now we focus on $B : B$ comparisons.

Number		Type of comparison	Outcome
1	$x : y$	$L : L$	$x > y$
2	$x : y$	$L : U$	$x > y$
3	$x : y$	$U : U$	$x < y$
4	$x : y$	$L : B$	$x > y$
5	$x : y$	$B : U$	$x < y$

Fig. 7. The adversaries answers to an algorithm's comparisons of cells from various regions. Assume all $B : B$ comparisons involve two cells that are both in $(2 \times b)$–Boxes.

Isolated Box-Types Here we consider $B : B$ comparisons in the isolated box-types, depicted in Figure 5. This part of the adversary also applies to $B : B$ comparisons in (2×2)–Boxes in canonical form. The three box-types depicted in Figure 5 are the only types of isolated $(2 \times b)$–Boxes that can emerge in the chain while an algorithm is run.

Assume some algorithm makes a $B : B$ comparison, say $x : y$. Let Box_1 be the (2×2)–Box x is in and let Box_2 be the (2×2)–Box y is in. These boxes can be uniquely defined: Box_1 and Box_2 must always be different boxes where one contains x and the other contains y and killing x or y will kill a border cell of Box_1 or Box_2, respectively. If we answer $x < y$, then a is the number of (2×2)–Box-cells killed. If we answer $x > y$, then b is the number of (2×2)–Box-cells killed. The numbers a and b are computed without letting a (2×2)–Box move. Then the adversary answers by the following three *Rules*:

1) If $a < b$, then answer $x < y$ and if Box_2 has not moved before, and if moving Box_2 down will not break the chain, then move Box_2 down one row.
2) If $a > b$, then answer $x > y$ and if Box_1 has not moved before, and if moving Box_1 up will not break the chain, then move Box_1 up one row.
3) If $a = b$, then answer so no border cell is killed–if possible. If both answers kill a border cell, try to move either of Box_1 or Box_2 (up or down one row) without breaking the chain. If no movement is possible answer arbitrarily.

Rules (1) and (2) are easily justified, for example consider these rules as applied to an algorithm that starts to make comparisons among elements in the canonical chain. Rule (3) is for box cells that are separated by at least one cell. In Rule (3) if both answers kill a border cell, then either answer will save one border cell. In Rule (3) if no box movement is possible, then Box_1 and Box_2 already have their 'third' comparisons accounted for.

In Figure 8, we let '2-Top,' denote the cell labeled '2' in the top of the non-isolated boxes. Further, the 4-right cell is the rightmost cell labeled '4,' *etc.*

In the last three parts of Figure 8 a comparison can be made in the neighboring (non-intersecting) dark (2×2)–Boxes invoking Rules (1),(2) and (3) making the dark (2×2)–Boxes intersect the (former) isolated box-type. These non-isolated box-types are handled with Rules (4) through (7).

Whenever a (2×2)–Box is moved, then no (2×2)–Box-cells are killed by comparison that caused the movement. So, this first comparison that cause a (2×2)–Box to move, is the "third comparison" associated with verifying the certificate in the moved (2×2)–Box. The moved (2×2)–Box has one alive border-cell remaining.

With the exception of the upper leftmost and lower rightmost (2×2) or (2×3)–Boxes (those containing cells $M[1, 1]$ and $M[n, n]$ respectively), every $(2 \times b)$–Box, $b \in \{2, 3, 4\}$ has b border cells assigned to it.

The isolated (2×3)–Boxes and (2×4)–Boxes already have at least one or at least two dead border cells, respectively (see Figure 5). Therefore, the adversary must keep one more border cell alive per (2×3)–Box or (2×4)–Box while a certificate is verified for these boxes. This is easy to do by keeping alive a (2×2)–Box-cell directly under or directly above a border cell. Since a (2×4)–Box is made of three (2×2)–Boxes this accounts for three comparisons per (2×2)–Box in the (2×4)–Box.

Theorem 6. Given an isolated $(2 \times b)$–Box, for $b \in \{2, 3, 4\}$, except the leftmost and the rightmost such $(2 \times b)$–Boxes. Using Rules (1) through (3), our adversary can keep alive at least one border cell of the $(2 \times b)$–Box while an algorithm makes $2(b - 1)$ comparisons among cells of the $(2 \times b)$–Box.

These border cells can always be chosen to be consistent with a totally monotone matrix. A border cell (and the associated alive half-column up or down) must eventually account for an additional comparison, otherwise the row minima are not known. Considering the border cells (of the isolated $(2 \times b)$–Box) that were killed to generate each of the isolated box-types along with the comparisons for finding a certificate an isolated $(2 \times b)$–Box and the extra comparison implied by Theorem 6, we have a total of $3(b - 1)$ comparisons. That is for $b - 1$ (2×2)–Boxes, three comparisons each.

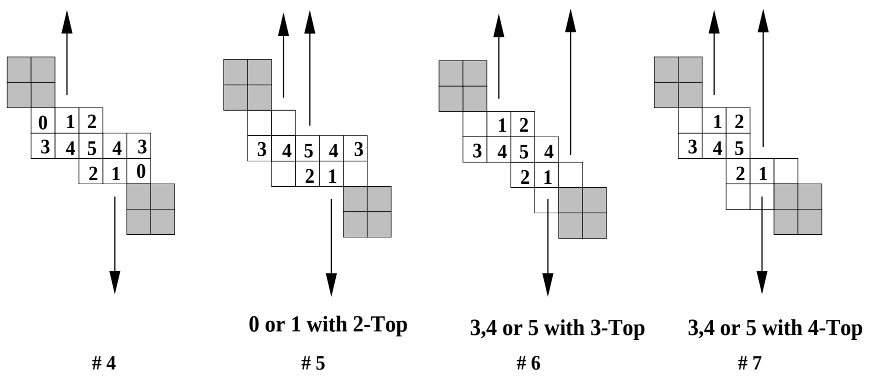

Fig. 8. Details of *non*-isolated $(2 \times b)$–Boxes and $(2 \times c)$–Boxes. The leftmost box is type of non-isolated Boxes we will consider.

Non-Isolated Box-Types Here we consider non-isolated boxes as in Figure 6. These boxes consist of intersecting $(2 \times b)$–Boxes and $(2 \times c)$–Boxes. Rules (1) through (3) work for the first non-isolated box-type in Figure 6. The second non-isolated box-type can be handled very similarly to the third non-isolated box-type, so we focus on the third non-isolated box-type. The basic idea is that given a $(3 \times b)$–Box, getting a certificate for the middle row makes finding certificates for the top and bottom rows cheaper. To foil this, after some comparisons are made the adversary 'switches back' from a $(3 \times b)$–Box to a $(2 \times b')$-Box and another $(2 \times b'')$-Box that do not intersect with each other.

If an algorithm makes a $B : B$ comparison between elements in the third case of Figure 6, then we augment the adversary with the following additional rules. (See Figure 8.)

Without loss of generality, suppose that we have not made any $B : B$ comparisons between any of the (2×3)–Box cells. Take the cell numbering given in Figure 8 and the following additional rules.

4) Comparing a 1-cell and a 0-cell. Always kill the 1-cell and without loss of generality assume that no (2×3)–Box cells are dead. Apply these rules again if another $B : B$ comparison is done in such non-isolated boxes.
5) Comparing a 0-cell or a 1-cell with the 2-cell. Kill the 2-cell while shifting down the (2×2)–Box containing the 2-cell. (See Figure 8 # 5).
6) Comparing cells 3-left, 4-left or 5-cell with cell 3-right. Here kill the 3-cell and the half column above it while moving down the one (2×2)–Box. Note that this (2×2)–Box avoids the now-dead 3-cell, but it now contains the dead border cell that was killed and caused the creation of the non-isolated boxes. That is, a dead cell is swapped for a dead cell, while moving a (2×2)–Box.
7) Comparing a 3-left, 4-left or 5-cell with cell 4-right. Kill the 4-cell while moving down *two* (2×2)–Boxes that are under the now-dead 4-cell. Note that these (2×2)–Boxes avoid the now-dead 4-cell, but they now contain the dead border cell that was killed and caused the creation of the non-isolated boxes. That is, a dead cell is swapped for a dead cell, while moving two (2×2)–Boxes.

Rule (4) makes comparing 0-cell and 1-cell do nothing for any algorithm since it just re-kills one of the border cells that caused the creation of the non-intersecting boxes. Rule (5) causes an additional border kill. This border kill, in conjunction with Theorem 6, gives three comparisons per (2×2)–Box in these non-isolated boxes. Rule (6) produces a new instance of the first non-isolated box-type (See the first part of Figure 6, note that here we have this one *upside-down* and with one dead cell in it.). Further, take # 6 in Figure 8 and there the lowest and rightmost (2×2)–Box that was in the non-isolated boxes now has one dead cell in it and one dead border cell. But this (2×2)–Box needs one more comparison to verify its certificate. Rule (7) introduces one isolated (2×3)–Box and an instance of the first non-isolated box-type with one dead (2×2)-Box-cell in it and another dead border cell.

The most important idea these rules introduce is that sometimes we swap one dead-cell for another by moving one or two of the (2×2)–Boxes. That is, in the worst case we can essentially go from a $(3 \times b)$–Box (or a non-isolated box type) back to a $(2 \times b')$–Box and a $(2 \times b'')$–Box that don't intersect each other. Where one of these $(2 \times b)$–Boxes is isolated and the other is a smaller non-isolated box-type. In particular, by this swapping of (2×2)–Boxes in Rules (5),(6) and (7) we never kill more (2×2)-Box-cells, but rather we trade one dead (2×2)-Box-cell for another in order to get another sub-case we know how to deal with. This "trading" of dead cells allows us to keep enough alive border cells to continue.

All of this leads to the following lemma.

Lemma 7. Given two non-isolated (2×3)–Boxes. Rules (4) through (7) make finding a certificate in these two non-isolated boxes take at least 12 comparisons.

Lemma 7 is based on keeping a border cell (hence half-column) alive that may contain the row minima. That is, the adversary can keep a border cell alive that is consistent with a totally monotone matrix.

Lemma 7 holds with 9 comparisons in the case of a non-isolated (2×2)–Box and a (2×3)–Box (in the middle of Figure 6). That is, three comparisons per (2×2)–Box in this case.

As Figure 8 shows, if the adversary is in a situation where there is a non-isolated box, then it adjusts its strategy to charge three comparisons to each (2×2)–Box. Furthermore, consider the cells immediately surrounding the two non-isolated (2×3)–Boxes. If any of these bordering cells are killed before the first $B : B$ comparison is made between the (2×2)-Box-cells, then the adversary focuses on keeping one border cell alive. The adversary can do this by Theorem 6.

In all four parts of Figure 8 a comparison can be made in the neighboring (non-intersecting) black (2×2)–Boxes forcing the black (2×2)–Boxes to overlap the grey non-isolated boxes. This is simply another case of non-isolated boxes which the adversary deals with as the non-isolated case.

Consider if two of the situations in #7 of Figure 8 'collide.' Take the case where the lower (2×3)-Box in #7 of Figure 8 forms another instance of the non-isolated box of #4 in this figure. If the 4-right cell was already dead, then we would not have formed such a non-isolated box. On the other hand, say we compare the 4-right with the 5-cell, then the adversary will just execute Rule (7) as usual.

Completing the Lower Bound We are now ready to put everything together and prove our main result.

Theorem 8. Any algorithm solving the row minima problem on a totally monotone $n \times n$ matrix must make at least $3n - 5$ comparisons between matrix elements.

Proof. Since there are $n - 1$ (2×2)–Boxes we know by Theorem 6 and Lemma 7 that we can assign to $n - 3$ of the (2×2)–Boxes a border kill that did not kill a (2×2)–Box-cell. However all minima are within the (2×2)–Box-cells. So to find them we must kill all but n of the (2×2)–Box-cells. By Rules (1) through (7) we know that we can kill at most one (2×2)–Box-cell with a single comparison. So by Lemma 4 we have to do $2n-2$ comparisons to kill the dead (2×2)–Box-cells. Together with the $n-3$ comparisons for border cells as just mentioned gives a total of $3n-5$ comparisons.

Certainly any algorithm must access all of the elements in a chain giving at least $3n-5$ different matrix accesses. By Theorem 6 and Lemma 4 a total of $4n-5$ matrix accesses are necessary to solve the row minima problem.

4 Conclusions

This work leaves the open problem of closing the gap between the lower and upper bounds of the row minima problem in a totally monotone matrix.

Acknowledgments

We are grateful to Shiva Chaudhuri, Volker Priebe, and Jesper Träff for their comments. Many thanks to Lawrence L. Larmore for pointing out an exact upper bound of the SMAWK algorithm. Thanks to Rudolf Fleischer, and Michiel Smid for stimulating discussions.

References

1. A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber: "Geometric Applications of a Matrix Search Algorithm," *Algorithmica*, Vol. 2, 195-208, 1987.
2. N. Alon and Y. Azar: "Comparison-Sorting and Selecting in Totally Monotone Matrices," *Proceedings of the Third Symposium on Discrete Algorithms (SODA '92)*, ACM Press, 403-408, 1992.
3. M. M. Klawe: "Super Linear Bounds for Matrix Searching Problems," *J. of Algorithms*, **13**, 55-78, 1992.
4. L. L. Larmore: "An Optimal Algorithm with Unknown Time Complexity for Convex Matrix Searching," *Information Processing Letters*, Vol. 36, 147-151, 1990.
5. L. L. Larmore: *Personal communication*, 1995.
6. L. L. Larmore and B. Schieber: "On-Line Dynamic Programming with applications to the Prediction of RNA Secondary Structure," *Journal of Algorithms*, **12**, 490-515, 1991.

This article was processed using the LaTeX macro package with LLNCS style