1. The one dimensional dynamic programming algorithm

In this section we describe a linear time algorithm for solving the on-line recurrence (**??**) under the assumption that $W(.,.)$ is concave. Instead of solving the recurrence we solve the following equivalent on-line matrix searching problem.

THE MATRIX SEARCHING PROBLEM. Let M be an $n \times n$ upper triangular concave totally monotone matrix. The rows of M are indexed in the range $1, \ldots, n$, and the columns are indexed in the range $0, \ldots, n-1$. Find the minimum element in each row of the matrix M, under the constraint that the elements of column j of M (that are not defined to be ∞) are available only after the minimum element in row j has been found.

The *Skewed* Matrix Searching Problem is defined as follows.

THE SKEWED MATRIX SEARCHING PROBLEM. Let M be an $n \times (2n-1)$ triangular concave totally monotone matrix with rows indexed in the range $1, \ldots, n$, and columns indexed in the range $0, \ldots, 2n-2$, and where $M[i,j] = \infty$ for all $j > 2(i-1)$. Find the minimum element in each row of the matrix matrix M, under the constraint that the elements of columns $2j-1$ and $2j$ of M (that are not defined to be ∞) are available only after the minimum element in row j has been found.

We show how to reduce an instance of size $n \times n$ of the Matrix Searching problem to an instance of size $\lfloor n/2 \rfloor \times (2\lfloor n/2 \rfloor - 1)$ of the Skewed Matrix Searching Problem. Then, we show how to reduce an instance of size $n \times (2n-1)$ of the Skewed Matrix Searching problem to an instance of size $n \times n$ of the Matrix Searching Problem. Both reductions take $O(n)$ time.

Let $T(n)$ be the time it takes to solve an instance of size $n \times n$ of the Matrix Searching problem. The two reductions imply that $T(n) = T(n/2) + O(n)$, and hence $T(n)$ is linear in n. In the sequel we make the analysis more carefully and compute the constants in $T(n)$.

From Matrix Searching to Skewed Matrix Searching

Let M be an $n \times n$ input matrix for the Matrix Searching problem. Define M' to be the sub-matrix consisting of all entries $M[2i,j]$ for which $j < 2i-1$; that is, M' consists of all the non-infinity elements, except the last, in each even row of M. It is easy to see that M' is an $\lfloor n/2 \rfloor \times (2\lfloor n/2 \rfloor - 1)$ input matrix for the Skewed Matrix Searching problem. (See Figure 1.)

Suppose that we are given an algorithm $\mathcal{A}'$ that computes on-line the row minima of M'. We describe an algorithm that computes on-line the row-minima of M using the algorithm $\mathcal{A}'$.

Let $j_M(i)$ be the column index of the minimum element in row i of M. Similarly, Let $j_{M'}(i)$ be the column index of the minimum element in row i of M'.

```
algorithm REDUCE1
begin
    comment: Column 0 of $M$ is available, and hence, row 1 of $M$ and
      row 1 of $M'$ are available. (Notice that both rows consist of only
      one non-infinity element, $M[1,0]$ and $M'[1,0]$.)
    report $j_M(1) = 0$;
    comment: Columns 0 and 1, and hence, row 2 of $M$ are available.
    if $M[2,0] < M[2,1]$
        then report $j_M(2) = 0$
        else report $j_M(2) = 1$;
    activate algorithm $\mathcal{A}'$ to report $j_{M'}(1) = 0$;
    for $i$ from 2 to $\lfloor n/2 \rfloor$ do
        comment: Columns $0, \ldots, 2i-2$ of $M$, and hence, row $i$ of $M'$
          and row $2i-1$ of $M$ are available.
        activate algorithm $\mathcal{A}'$ to report $j_{M'}(i)$;
        comment: Row $i$ of $M'$ consists of the first $2i-1$ elements of
          row $2i$ of $M$. From this and the total monotinicity we get that
          $j_M(2i-2) \le j_M(2i-1) \le j_{M'}(i)$.
        report $j_M(2i-1)$ by computing the minimum among $M[2i-1, j]$,
          for $j = j_M(2i-2), \ldots, j_{M'}(i)$;
        comment: Column $2i-1$ of $M$ is available, and hence, the entry
          $M[2i, 2i-1]$ is available.
        if $M[2i, j_{M'}(i)] < M[2i, 2i-1]$
            then report $j_M(2i) = j_{M'}(i)$
            else report $j_M(2i) = 2i-1$;
    end for
    if $n$ is odd then
        report $j_M(n)$ by computing the minimum among $M[n, j]$, for
          $j = j_M(n-1), \ldots, n-1$;
  end
```

It is easy to see the correctness of the above reduction.

We consider two timing measures for the reduction: the number of comparisons, and the number of fetches to the matrix M. Notice that since we treat algorithm $\mathcal{A}'$ as a “black box”, we only count those comparisons and fetches needed by the reduction, i.e., those needed by the algorithm that are not done by $\mathcal{A}'$.

Number of comparisons: The number of comparisons done to determine the minima of each even row of M is only one, i.e., $\lfloor n/2 \rfloor$ altogether. The number of comparisons needed in all the interpolation steps (Step 2 in all the iterations) is at most $n-1$. Thus, $3n/2$ is an upper bound on the number of comparisons done by the reduction.

Number of fetches: We refer to an entry $M[i, i-1]$ as a *diagonal* entry of M. To compute the minima of the even rows of M, the diagonal entry in each even-indexed row has to be fetched. The minimum of all other entries in that row does not need to be fetched, since it has been fetched by $\mathcal{A}'$ and can be retained. Thus, $\lfloor n/2 \rfloor$ fetches suffice for the even rows. The interpolation steps for all the odd rows together require at most $3n/2$ fetches. We conclude that $2n$ is an upper bound on the number of fetches needed by the reduction. Later we show that in the recurrence it suffices to consider only the fetches to odd rows, and there are no more than $3n/2$ of those fetches. This is since the diagonal entries have been fetched already at a higher level in the recursion and can be retained.

From Skewed Matrix Searching to Matrix Searching

Let M be an $n \times (2n-1)$ input matrix for the Skewed Matrix Searching problem. We show how to reduce the problem of finding the row-minima of M to the problem of finding the row minima of an $n \times n$ sub-matrix M' of M that is an instance of the Matrix Searching problem.

Figure 1 shows an example of such a sub-matrix M'.

The sub-matrix is defined by *headers.* Each row of M contains exactly one header. The matrix M' consists of the headers together with all entries that are in the columns of those headers and below them. The headers have the property that the column index of the header in row $i+1$ is strictly higher than the column index of the header entry in row i. It follows that the headers are exactly the diagonal entries of M'.

Suppose that we are given an algorithm $\mathcal{A}'$ that computes on-line the row minima of M'. We describe an algorithm that computes on-line the row-minima of M using the algorithm $\mathcal{A}'$.

Unlike the previous reduction, the actual set of entries of the sub-matrix M' is not known in advance. In fact, the header of row i, that defines column $i-1$ of M', is determined only at the last possible moment before this column is needed to be passed on to the interleaved algorithm $\mathcal{A}'$ (that is, before the minimum of row i of M' is computed).

We use an array S to store the tentative headers. The entry $S[i]$ is either undefined, or is the column index of the current tentative header in row i. If this entry survives the elimination process, then column $i-1$ of M' will be the portion of column $S[i]$ of M from the header and below. The array S can be viewed as a stack, as in the algorithms of [1, 2].

Informal description of the algorithm: The entry $S[1]$ is initialized to be 0. Then, for $j = 1, \ldots, 2n-2$, each column j is processed as follows. The processing consists of repeated matching of the top entry on the stack S, say $S[t]$ with the entry of column j in row t. Each time column j "wins" the match, i.e., $M[t,j] < M[t, S[t]]$, the stack is popped (and t is decremented by one). The stack can be popped (i.e., column $S[t]$ can be omitted), beacause the concavity of M implies that none of the elements of column $S[t]$ below (and including) the "losing" header are candidates for the minimum elements in their rows. The first time column j "loses" the match, i.e., the first time $M[t,j] > M[t, S[t]]$, or when column j runs out of tentative headers to challenge, the entry just below the last "losing" entry in column j is pushed to the stack and becomes the new top tentative header.

Figure 1 shows column j "winning" the match and eliminating the top two tentative headers in the stack, then "losing" to the next one.

Notice that a tentative header in row i may be replaced only by tentative headers whose column index is less or equal to $2i-2$. This is, since for $j > 2i-2$, $M[i,j] = \infty$, and thus the tentative header will always "win" the match with headers from these columns. This implies that after $2i-2$ columns of M have been processed the tentative header in row i becomes *permanent.* At this time column $S[i]$ can be passed as column $i-1$ of M' to the algorithm $\mathcal{A}'$, and it can be activated to report the minimum of row i of M'. It is important to note that tentative headers become permanent headers just exactly at the last possible moment when they are needed by algorithm $\mathcal{A}'$. Thus, the columns of M' become visible to algorithm $\mathcal{A}'$ just when it needs them.

Below, we give a formal description of the algorithm:

```
algorithm REDUCE2
begin
  $t \leftarrow 1$;
  $S[1] \leftarrow 0$;
  report $j_{M'}(1) = 0$;
  for $j$ from $1$ to $2n-2$ do
    while $M[t,j] < M[t,S[t]]$ do
      $t \leftarrow t-1$;
    end while
    if $t < n$ then
      $t \leftarrow t+1$;
      $S[t] \leftarrow j$;
    end if
    if $j$ is even then
      pass column $S[1+j/2]$ of $M$ as column $j/2$ of $M'$;
      comment: All entries of row $1+j/2$ of $M'$ are available.
      activate algorithm $\mathcal{A}'$ to report $j_{M'}(1+j/2)$;
      report $j_M(1+j/2) = S[1+j_{M'}(1+j/2)]$;
    end if
  end for
end
```

Correctness of the algorithm. We say that an entry $M[i,j]$ can be *eliminated* if, based on the results of the tests made so far, $M[i,j]$ is not a candidate for the minimum of row i. Note that because of the concavity of M, if $M[i,j_1] \leq M[i,j_2]$ and $j_1 < j_2$, then all the entries $M[k,j_2]$, for all $k \leq i$, can be eliminated. Similarly, if $M[i,j_1] \leq M[i,j_2]$ and $j_1 > j_2$, then all the entries $M[k,j_2]$, for all $k \geq i$, can be eliminated.

Lemma 1.1: *The following loop invariants hold at the end of each iteration j.*

Loop invariants for iteration i.

1. *For $1 \leq i \leq t$, the portion of column $S[i]$ above row i can be eliminated.*
2. *For $1 \leq k \leq j$, if k is not an entry in S then the entire column k can be eliminated.*

These invariants readily imply the correctness of the algorithm.

Proof of Invariants: The invariants hold vacuously before starting the first iteration. Suppose that the invariants hold for all iterations $1, \ldots, j-1$. We show that they hold also for iteration j. Each test of the while loop, if successful, allows column $S[t]$ to be completely eliminated. This is, since by the hypothesis on Invariant (1), the portion of this column above row t can be eliminated, and by the test of the while loop its portion below and including row t can also be eliminated. Thus, Invariant (2) holds for iteration j. If the test of the while loop fails, then that test allows the portion of column j with row indices $1, \ldots, t$ to be eliminated. Thus Invariant (1) also holds. □

Again, we consider two timing measures for the reduction: the number of comparisons, and number of fetches to the matrix M. We show that the above reduction requires at most $3n$ comparisons and $5n$ fetches of entries of M.

Number of comparisons: The comparisons are made only in the test of the while loop. We charge each such comparison to the "losing" column. In all the comparisons in which the test of the while loop fails, the index of the "losing" column is pushed into the stack. In all the comparisons in which the test of the while loop succeeds, the index of the "losing" column is popped off the stack. Thus, each of the n columns in the stack at the end of the algorithm has lost at most once, and each of the other $n-1$ columns has lost at most twice. We conclude that the total number of comparisons is less than $3n$.

Number of fetches: The fetches are needed only for the comparisons. To analyze the number of fetches assume that whenever a value j is pushed into the stack the entry $M[t,j]$ is fetched and stored in the the stack along with j.

In each comparison between $M[t,j]$ and $M[t,S[t]]$, one fetch is needed to bring $M[t,j]$. The value of $M[t,S[t]]$ has been fetched already and stored at the top of the stack, and hence does not need to be fetched again. We conclude that the number of fetches is the number of comparisons, which is less than $3n$, plus the number of pushes, which is $2n-1$, for a total of less than $5n$.

Theorem 1.2: *The algorithm given by the above reductions solves the $n \times n$ concave triangular matrix searching problem. Moreover,*

(i) The number of comparisons executed by the algorithm is no more than $6n$.

(ii) The number of fetches of entries of M executed by the algorithm is no more than $8.5n$.

Proof: The correctness of the algorithm follows from the above discussion.

Let $C(n)$ be the number of comparisons executed. Using our reductions we get the recurrence

$$C(n) \le 3n/2 + 3n/2 + C(\lfloor n/2 \rfloor).$$

Solving the recurrence, we obtain that $C(n) \leq 6n$.

Let $F(n)$ be the number of fetches executed. Using our reductions it seems that we get the recurrence

$$F(n) \leq 2n + 5n/2 + F(\lfloor n/2 \rfloor),$$

that gives $F(n) \leq 9n$. A closer look at the fetches executed by the algorithm reveals that some fetches are counted twice. Consider a reduction from Matrix Searching to Skewed Matrix Searching. Recall that in this reduction all the diagonal entries of the even rows have to be fetched. However, if this is not the first reduction all these entries have been fetched already in the previous reduction from Skewed Matrix Searching to Matrix Searching. Let $F'(n)$ be the number of fetches executed given that all the *diagonal* entries of M are already fetched. Using our reductions we get the recurrence

$$F'(n) \leq 3n/2 + 5n/2 + F'(\lfloor n/2 \rfloor),$$

that gives $F'(n) \leq 8n$. The only diagonal entries that have to be fetched are the $\lfloor n/2 \rfloor$ diagonal entries of the even rows needed for the first reduction. Hence, $F(n) = F'(n) + \lfloor n/2 \rfloor \leq 8.5n$. $\square$

References

[1]

[2]