EFFICIENT DILATION, EROSION, OPENING AND CLOSING ALGORITHMS

JOSEPH (YOSSI) GIL and RON KIMMEL
Department of Computer Science
Technion–Israel Institute of Technology
Technion City, Haifa 32000, Israel

Abstract. We propose an efficient algorithm for computing the dilation and erosion filters. For a p-element sliding window, our algorithm computes the 1D filter using $1.5 + o(1)$ comparisons per sample point. Our algorithm constitutes improvements over the best previously known such algorithm by Gil and Werman [5]. The previous improvement on [5] offered by Gevorkian, Astola and Atourian [2] was in better expected performance for random signals. Our result improves on [5] result without assuming any distribution of the input. Further, a randomized version of our algorithm gives an expected number of $1.25 + o(1)$ comparisons per sample point, for *any* input distribution. We deal with the problem of computing the dilation and the erosion filters simultaneously, and again improve the Gil-Werman algorithm in this case for independently distributed inputs. We then turn to the *opening* filter, defined as the application of the min filter to the max filter, and give an efficient algorithm for its computation. Specifically, this algorithm is only slightly slower than the computation of just the max filter. The improved algorithms are readily generalized to two dimensions for rectangular structuring element, as well as to any higher finite dimension for a hyper-box structuring element, with the number of comparisons per window remaining constant.

Key words: Max-Filter, Min-Filter, Running Window

1. Introduction

In signal and image analysis one often encounters the problem of min (or max) computation in a window with p elements in the one-dimensional (1D) case, or $p \times p$ elements in the 2D case. In mathematical morphology [7], the result of such an operator is referred to as the erosion (or dilation) of the signal with a *structuring element* given by a flat ramp of width p.

An important application is the morphological edge detector, obtained by applying both the min filter and the max filter, and then subtracting the results. Denoising is yet another example of using the min and max filters. The *opening* (respectively *closing*) filter is obtained by feeding the results of the max (resp. min) filter to the min (resp. max) filter. In image processing the opening filter eliminates small dark regions, while closing eliminates small white regions. In both filters, the size of the window determines the size of the regions that can be removed. Some other applications of the min and max filters in pattern analysis, adaptive signal processing and morphological analysis are mentioned in [2]. These applications call our interest to the problem of efficiently computing the min and max filters for a wide range of p.

The one-dimensional version of the problem can be formulated as follows:

1D MAX-FILTER: *Given a sequence* $x_0, \ldots, x_{n-1}$, *and an integer* $p > 1$, *compute* $y_i = \max_{0 \leq j < p} x_{i+j}$, *for* $i = 0, \ldots, n-p$.

The 1D MIN-FILTER problem is similarly defined.

As usual in filtering, we assume that $p \ll n$. As an efficiency measure of algorithms for this problem we use C_1, defined as the number of comparison operations per sample (or output) point as n goes to infinity.

Since any max filter computation must examine every input element at least once, we have that ($C_1 \geq 1$. A trivial algorithm for the 1D MAX-FILTER problem gives $C_1 = p-1$. On the other hand, since it is impossible to compute the filter without examining each input point at least once, there is a trivial information theoretical lower bound for the problem of $C_1 \geq 1$.

Two non-trivial algorithms for the problem were published in [6]: The first achieves $C_1 = O(\lg p)$[1] and the second $C_1 = 3 + o(1)$ for uniformly distributed independent input signals. The worst case performance for both of these algorithms depends on the window size.

Gil and Werman, in their work on computing the median filter [5], gave the first algorithm for computing the max filter whose performance does not depend on p. Their algorithm is more general since it can compute any semi-ring operation, $\diamond$, filter of size p while using $3-4/p$ applications of $\diamond$ per sample point. Since max is a semi-ring operation, their result gives $C_1 = 3 - 4/p$.

Gevorkian, Astola and Atourian [2] observed that in the special case when the semi-ring operation is max, the Gil-Werman algorithm can be improved, assuming locally uniform distributed signals, to achieve $E(C_1) = 2.5 - 3.5/p$. The expectation here is respectively to input distribution. In the worst input case, the performance of the algorithm of [2] is the same as the Gil-Werman algorithm. Here we describe an algorithm achieving further reduction, $C_1 = 1.5 + \frac{\lg p}{p} - O(1/p)$. This improvement is *deterministic* and does not make any assumptions on the input distribution.

Further, we also describe a randomized algorithm which comes even closer to the lower bound in achieving $E(C_1) = 1.25 + \frac{\lg p}{p} - O(1/p)$, where the expectation is w.r.t. random selections made by the algorithm. I.e., this expected performance is obtained for any input.

The optimal L-filter, the morphological edge detector, and other applications call for the simultaneous computation of the min and max in each window, as summarized in the following problem definition.

1D MAX-MIN-FILTER: *Given a sequence* $x_0, \ldots, x_{n-1}$, *and an integer* $p > 1$, *compute* $y_i = \max_{0 \leq j < p} x_{i+j}$ *and* $z_i = \min_{0 \leq j < p} x_{i+j}$ *for* $i = 0, \ldots, n-p$.

We give an algorithm that solves the 1D MAX-MIN-FILTER problem faster than solving the 1D MAX-FILTER and the 1D MIN-FILTER. Let C_1^m be the number of comparisons per input sample for solving 1D MAX-MIN-FILTER. Then the algorithm achieves $E(C_1^m) \approx 2 + 2.3466\frac{\lg p}{p}$, for the special case of independent input distribution, i.e., the expectation is with regard to input distribution. In the worst case this algorithm does not improve on the independent computation of the Min- and Max filters. However, for natural images,

[1] We use $\lg(\cdot)$ to denote $\log_2(\cdot)$

the algorithm makes such an improvement.

The problem posed by the opening filter is similar to 1D MAX-MIN-FILTER, since in both it is required to compute both a Min-Filter and a Max-Filter. However, the fact that in the opening filter this filters are computed sequentially, where the results of one filter are fed to the other, makes it mush easier. Let C_1^o be the number of comparisons per input sample for computing the opening filter. Then, we show that $C_1^o \leq C_1 + O(\frac{\lg^2 p}{p})$. Clearly, the same result holds for the closing filter.

As described in [5], a 1D max filter can be extended to square (or rectangular) window 2D max filter. This is done by first applying the 1D filter along the rows, and then feeding the result to a 1D filter running along the columns. Let C_2 be the number of comparison operations required per input point for computing the 2D max filter. We have that $C_2 = 2C_1$, and more generally, $C_d = dC_1$, where C_d is defined accordingly for the d-dimensional filter. We similarly have that $C_d^m = dC_1^m$ and $C_d^o = dC_1^o$.

Outline. The remainder of this paper is organized as follows. Section 2 reviews the Gil-Werman algorithm. The deterministic and randomized algorithms improving it are described in Section 3. In Section 4 we give our algorithm for the 1D MAX-MIN FILTER PROBLEM. The efficient algorithm for computing the opening (and closing) filter is described in Section 5, and conclude with Section 6.

2. The Gil-Werman algorithm

The Gil-Werman algorithm is based on a partitioning of the input signal to overlapping segments of size $2p-1$, centered at $x_{p-1}, x_{2p-1}, x_{3p-1}, \ldots$. Let j be the index of an element at the center of a certain segment. The maxima of the p windows which include x_j are computed in one *batch* of the Gil-Werman algorithm as follows: First, define R_k and S_k for $k = 0, \ldots, p-1$:

$$R_k = \max(x_j, x_{j-1}, \ldots, x_{j-k}), \quad \text{and} \quad S_k = \max(x_j, x_{j+1}, \ldots, x_{j+k}). \tag{1}$$

Now, the R_k's and the S_k's can be merged together to compute the max filter:

$$\max(x_{j-k}, \ldots, x_0, \ldots, x_{j+p-k-1}) = \max(R_k, S_{p-k-1}), \tag{2}$$

for $k = 1, \ldots, p-2$. In addition, we have $\max(x_{j-p-1}, \ldots, x_j) = R_{p-1}$ and $\max(x_0, \ldots, x_{j+p-1}) = S_{p-1}$.

There are two steps to the Gil-Werman algorithm:

Preprocessing Computing all R_k and S_k from their definition (1) which is done in $2(p-1)$ comparisons.

Merge Merging the R_k and S_k together using (2), for which another $p-2$ comparisons are required.

Since this procedure computes the maximum of p windows in total, we have that the amortized number of comparisons per window is $3 - 4/p$. For large p, we have that the preprocessing step requires two comparison operations per element, while the merge step requires one more comparison.

3. The Improved Algorithms For the Max-Filter

Let us show how the two steps of the Gil-Werman algorithm can be carried out more efficiently.

3.1. An efficient merge procedure

We first show how to improve the merge step, by reducing the number of comparisons from 1 to $\lg p/p + o(1)$. In this step, we compute

$$\max(R_k, S_{p-k-1}), \tag{3}$$

for $k = 1, \ldots, p-2$. Observing that $R_{p-2} \geq R_{p-1} \geq \ldots \geq R_1$, and $S_{p-2} \geq S_{p-1} \geq \ldots \geq S_1$, we can eliminate most of these comparisons. Suppose that for some specific i it was found that $R_i \geq S_{p-i-1}$, then for all $k > i$, we have that $R_k \geq R_i \geq S_{p-i-1} \geq S_{p-k-1}$, and therefore there is no need to do the comparisons of (3) for all $k > i$. Similarly, if it is determined that $R_i \leq S_{p-i-1}$, then we do not need to do the comparisons of (3) for all $k < i$.

The optimized procedure for the merge step is therefore a binary search. We start by setting $i = \lceil (p-2)/2 \rceil$, and then continue with the remaining half of the problem size. The number of comparisons is thus reduced from $p-2$ to $O(\lg p)$. In fact, it can be easily checked that the number of comparisons in the binary search of the merge step is exactly $\lceil \lg p - 1 \rceil$. The amortized contribution of the improved merge step to the complexity is $\frac{\lceil \lg p - 1 \rceil}{p}$.

3.2. An efficient preprocessing computation

Let us now deal with the preprocessing step of the Gil-Werman algorithm. Gevorkian, Astola and Atourian [2] observed that preprocessing computation can be made more efficient for randomized input, using the fact that in the Gil-Werman algorithm, the suffixes S_k of one segment overlap with the prefixes R_k of the following segment. Specifically, the problem that needs to be solved is

Prefix-Suffix Max: *Given a sequence $x_0, \ldots, x_p$, compute all of its* prefix maxima*: $s_k = \max(x_0, \ldots, x_k)$, for $k = 0, \ldots, p-1$, and all its* suffix maxima*: $r_k = \max(x_k, \ldots, x_p)$, for $k = 1, \ldots, p$.*

Note that this problem does not call for computing the overall maximum of the input $s_p = r_0 = \max(x_0, \ldots, x_p)$.

The original Gil-Werman algorithm makes $2(p-2)$ comparisons in solving the Prefix-Suffix Max problem. We propose the following efficient solution for this problem. Let $q = \lfloor (p+1)/2 \rfloor = p/2 + (p \bmod 2)/2$. In the *first part* of the modified implementation, compute all s_k, for $k = 0, \ldots, q-1$ and r_k for $k = q, \ldots, p$. This is carried out using $p-1$ comparisons.

The *second part* of the modified implementation of the preprocessing stage begins in comparing s_{q-1} and r_q. If $r_q \geq s_{q-1}$, then we know that the overall maximum falls is one of $x_q, \ldots, x_p$. Therefore, it is unnecessary to further compute the value of $r_{q-1}, r_{q-2}, \ldots, r_1$. Instead, the algorithm outputs

$r_{q-1} = r_{q-2} = \ldots = r_1 = r_q$, and continues to compute $s_q, \ldots, s_{p-1}$. A similar situation occurs if $r_q \leq s_{q-1}$, in which case it is unnecessary to compute $s_q, \ldots, s_{p-1}$. In both cases, the number of comparisons that remain to be done is $\lfloor (p+1)/2 \rfloor$. The total number of comparisons in the more efficient algorithm for PREFIX-SUFFIX MAX is $(p-1)+1+\lfloor \frac{p+1}{2} \rfloor = 1.5p + \frac{p \bmod 2}{2}$. Noting that each batch requires (on amortization) solving one instance of PREFIX-SUFFIX MAX, we can combine our results so far to obtain:

Theorem 1 *There exists a deterministic algorithm for the* 1D MAX-FILTER *problem, achieving* $C_1 = 1.5 + \frac{\lceil \lg p - 1 \rceil}{p} + \frac{p \bmod 2}{2p} \leq 1.5 + \frac{\lg p}{p}$.

Can we improve on this result? An information theoretical lower bound for the number of comparisons required to solve PREFIX-SUFFIX MAX, is $p + \lg p - O(1)$. This bound is derived as follows. A compact output of an algorithm for the problem uses $p + \lg p - O(1)$ bits comprised as follows:

1. $\lg p$ bits to designate the location of the overall maximum (for simplicity, we assume that p is a power of 2),
2. one bit for each location prior to the maximum, designating whether the corresponding element changes the prefix maxima, and
3. one bit for each location following to the maximum, designating whether the corresponding element changes the suffix maxima.

Moreover, there are distinct inputs which produce all the bit combinations of this compact representation. Thus, in order to make the distinction between these inputs, the algorithm is forced to make at least $p + \lg p - O(1)$ comparisons.

Although we are unable to meet this lower bound, we can come close to it in an important special cases. Suppose that in an input to the PREFIX-SUFFIX MAX problem, the overall maximum is located at a random location ℓ in the input sequence. (This does not necessarily mean that the input is uniformly and independently distributed). Then, once the comparison between s_{q-1} and r_q is made, all that remains is to proceed to compute outputs $s_q, s_{q+1}, \ldots, s_{\ell-1}$ in the case that $s_{q-1} < r_q$, or $r_{q-1}, r_{q-2}, \ldots, r_{\ell+1}$ otherwise. The expected number of comparisons in this completion stage is

$$\frac{1}{p+1}\left(\sum_{i=0}^{q-1} i + \sum_{i=0}^{p+1-q} i\right) = \frac{p^2-(p \bmod 2)}{4(p+1)} \leq \frac{p}{4} - \frac{1}{4} + \frac{1}{4(p+1)} \leq \frac{p}{4}. \qquad (4)$$

In general, it cannot be assumed that an arbitrary input to the PREFIX-SUFFIX MAX problem will have its maximum at a random location. However, in using this procedure as part of an algorithm for solving the 1D MAX-FILTER problem, we can achieve this effect by choosing at random the starting point for segmentation. Thus, the segments will be centered at positions indexed $\tau, \tau + p, \tau + 2p, \ldots$, where τ is an integer selected at random in the range $[0, \ldots, p-1]$. Such a random selection does not degrade the efficiency due to the assumption that $p \ll n$. We have thus obtained:

Theorem 2 *There exists a randomized algorithm for the* 1D MAX-FILTER *problem, achieving* $E(C_1) \leq 1.25 + \frac{\lceil \lg p - 1 \rceil}{p} + \leq 1.25 + \frac{\lg p}{p}$.

The interested reader is referred to e.g., [3] and the references thereof for examples of applying the template mechanism for non-trivial compile-time computation and code generation that are useful for implementation of logical cases involved in the proposed algorithms.

4. Efficient Algorithm for Computing the Max and Min Together

Let us deal with the 1D MAX-MIN FILTER problem, and show how the min and max filters together can be computed more efficiently than an independent computation of both. We start again from the Gil-Werman algorithm. The gain comes from partitioning the input signal into pairs of consecutive elements, and comparing the values in each pair. The greater value in each pair carries on the maximum computation while the lesser one carries one to the minimum computation.

4.1. THE PREFIX MAX-MIN PROBLEM

Let us first consider the following problem,

PREFIX MAX-MIN: *Given a sequence* $x_0, \ldots, x_{q-1}$*, compute* $M_k = \max(x_0, \ldots, x_k)$*, and* $m_k = \min(x_0, \ldots, x_k)$*, for* $k = 0, \ldots, q-1$*.*

The straightforward solution for PREFIX MAX-MIN uses a total of $2(q-2)+1$ comparisons. Analyzing this problem from an information theoretic point of view we find that for all $i > 2$, there are three cases for element x_i. It either increases the running prefix maximum, or it decreases the running prefix minimum, or makes no changes to those. There are only two possible cases for x_1, while there is exactly one case for x_0. Thus, we obtain $1+(q-2)\lg 3 \approx 1.58496q$, as an information theoretic lower bound for the number of comparisons for this problem.

We do not know of a general way of bringing the amortized number of comparisons from $2 - o(1)$ closer to the $\lg 3$ lower bound, or proving a stronger lower bound. However, if it is known that the distribution of input elements is independent, we can even do better than the lower bound! This improvement is carried out as follows. Suppose that M_i and m_i were already computed. Then, to compute M_{i+1}, M_{i+2}, m_{i+1} and m_{i+2}, we apply the following *incorporate-next-input-pair* algorithm.

Algorithm incorporate-next-input-pair: Extend the result of a solution to PREFIX MAX-MIN to include input elements x_{i+1} and x_{i+2}, using the four following comparisons:

1. *Compare* x_{i+1} *and* x_{i+2}*. Assume, without loss of generality, that* $x_{i+1} \geq x_{i+2}$*.*
2. *Compare* M_i *with* $x_{i+1} = \max(x_{i+1}, x_{i+2})$*.*
3. *Compare* m_i *with* $x_{i+2} = \min(x_{i+1}, x_{i+2})$*.*
4. *At this stage, the algorithm has determined both* M_{i+2} *and* m_{i+2}*. Specifically,* $M_{i+2} = \max(x_{i+1}, M_i)$ *and* $m_{i+2} = \min(x_{i+2}, m_i)$*. There are four cases to consider in computing* m_{i+1} *and* M_{i+1}*.*
   a) No changes: $x_{i+1} \leq M_i$ *and* $x_{i+2} \geq m_i$*. No more comparisons need to be done in this case, and the algorithm outputs* $M_{i+2} = M_{i+1} = M_i$ *and* $m_{i+2} = m_{i+1} = m_i$*.*

b) Changes to both the maximum and the minimum: $x_{i+1} \geq M_i$ *and* $x_{i+2} \leq m_i$. *Again, no more comparisons need to be done in this case, and the algorithm outputs* $M_{i+2} = M_{i+1} = x_{i+1}$, $m_{i+1} = m_i$, *and* $m_{i+2} = x_{i+2}$.

c) Change to the maximum: $x_{i+1} \geq M_i$ *and* $x_{i+2} \geq m_i$. *The algorithm outputs* $M_{i+2} = M_{i+1} = x_{i+1}$ *and* $m_{i+2} = m_{i+1} = m_i$. *without additional comparisons.*

d) Possible change to the minimum: $x_{i+1} \leq M_i$ *and* $x_{i+2} \leq m_i$. *This is the only case in which an additional comparison is required: The algorithm first outputs* $M_{i+2} = M_{i+1} = M_i$, $m_{i+2} = x_{i+2}$ *and then determines* m_{i+1} *by comparing* x_{i+1} *with* M_i. *If* $x_{i+1} < m_i$ *then* $m_{i+1} = x_{i+1}$, *otherwise,* $m_{i+1} = m_i$.

Thus, in the worst case, the algorithm makes four comparisons for each pair x_{i+1} and x_{i+2}, where $i > 0$ is odd, which does not improve on the two comparisons for element by the trivial algorithm. The fourth comparison however is needed only in case

$$x_{i+2} < m_i = \min_{0 \leq j \leq i}(x_i), \tag{5}$$

or in the dual case, namely when the first comparison yields $x_{i+1} \leq x_{i+2}$, and

$$x_{i+2} < M_i = \max_{0 \leq j \leq i}(x_i). \tag{6}$$

With i.i.d. the probability of (5) or (6) holding is $1/(i+3)$, for all $i > 0$. Let $u = \lfloor q/2 \rfloor - 1 = (q - (q \bmod 2)) - 1$. Then in the last application of the above algorithm we deal with the pair x_{2u} and x_{2u+1}. In total, F_q, the *expected* (with regard to input distribution) number of times the fourth comparison is made is given by

$$F_q = \frac{1}{4} + \frac{1}{6} + \frac{1}{8} + \cdots + \frac{1}{2u+2} = (H_{u+1} - 1)/2, \tag{7}$$

where H_u is the uth harmonic number. It is well known that

$$\lim_{u \to \infty} H_u = \ln u + \gamma \quad \text{and} \quad \ln u + \gamma \leq H_u \leq \ln u + 1, \tag{8}$$

where $\gamma \approx 0.577216$ is Euler's constant (also called Mascheroni's constant). Combining (7) and (8) we have

$$\begin{aligned} F_q &= \tfrac{\ln(u+1)}{2} + \tfrac{\gamma}{2} - 0.5 + o(1) \\ &\approx \tfrac{\ln(u+1)}{2} - 0.211392 + o(1) \leq \tfrac{\ln(u+1)}{2} \leq \tfrac{\ln q - 1}{2}. \end{aligned} \tag{9}$$

Other than these, in solving PREFIX MAX-MIN, there are u applications of *incorporate-next-input-pair*, in which $3u$ comparisons are made, one comparison in which x_0 is compared with x_1 to determine M_0, M_1, m_0 and m_1, and finally, and only if q is odd, two comparisons to determine M_{q-1} and m_{q-1}. The number of these comparisons is

$$1 + 3u + 2(q \bmod 2) = \frac{3q}{2} - 2 + \frac{q \bmod 2}{2}. \tag{10}$$

Adding (9) and (10) we have that the expected total number of comparisons in our solution to PREFIX MAX-MIN is at most

$$\frac{3q}{2} + \frac{\ln q}{2} - 2, \tag{11}$$

and the expected amortized number of comparisons per element is $1.5 + \frac{\ln q}{2q} - 2/q$. It should be noted that one cannot hope to improve much on this result. The reason is that solving PREFIX MAX-MIN also yields the maximum and the minimum of the whole input. However, computing both these values cannot be done in less than $\lceil 3p/2 \rceil$ comparisons [1, page 187] even for randomized inputs.

4.2. COMPUTING THE MIN-MAX FILTER

We now employ algorithm *incorporate-next-input-pair* in the pre-processing stage of the modified Gil-Werman algorithm adapted for finding both the minimum and the maximum filters. Specifically, we are concerned in this stage in finding an efficient algorithm to the PREFIX-SUFFIX MAX-MIN problem, defined as computing the maximum and the minimum of all prefixes and all suffixes of an array of size $p+1$. Such an efficient algorithm is obtained by partitioning the input array into to two halves. In the lower half which comprises $q = \lfloor (p+1)/2 \rfloor = p/2 + (p \bmod 2)/2$ elements we repetitively apply *incorporate-next-input-pair* to compute the prefix maxima and the prefix minima in this half. A similar computation is carried out in the upper half with $p - q + 1 = \lceil (p+1)/2 \rceil$ elements of the input array, except that algorithm *incorporate-next-input-pair* is mirrored to compute the *suffix* minima and the suffix maxima in this half. The total expected number of comparisons so far can be computed from (11):

$$\tfrac{3q}{2} + \tfrac{3(p+1-q)}{2} + \tfrac{\ln q}{2} + \tfrac{\ln(p+1-q)}{2} - 4 \le \tfrac{3p}{2} + \ln p - 2.5. \tag{12}$$

Once this computation is done, we carry on as before to produce the rest of the required output. In two more comparisons we find out where the maximum and the minimum of the whole array occur. If the maximum occurs in the lower (resp. upper) half then it remains to compute the suffix (resp. prefix) maxima from the mid-point down-to (resp. up-to) the location of the maximum. From (4) we have that this computation costs another 0.25 comparison per input element. A similar completion stage must be carried out for the minimum prefixes or suffixes, using another 0.25 amortized comparisons. All that remains is the merge step, which has to be carried out twice, once for the minimum and once for the maximum. The number of comparisons for the merge is at most $2 \lg p$. Combining this with (12) we obtain:

Theorem 3 *There exists an algorithm for the* 1D MIN-MAX FILTER *problem, that at the worst case makes twice the number of comparisons as that of Theorem 2. For independently distributed inputs, the amortized number of comparisons that the algorithm makes is*

$$C_1^m < 2 + 2\frac{\ln p + \lg p}{p} = 2 + (2 + \frac{\ln 2}{2})\frac{\lg p}{p} \approx 2 + 2.3466\frac{\lg p}{p}.$$

Stated differently, we have that asymptotically for large p, and for i.i.d. one comparison per element is required to compute each of the minimum and the maximum filters, provided they are computed together. We refer to [4] for further details like performance on natural images.

5. An Efficient Algorithm for the Opening and Closing Filters

In this section we describe how the opening (and closing) filter can be computed more efficiently than a mere sequential application of the Max-Filter and then the Min-Filter.

To understand the improvement, consider the problem of computing the prefix-minimum, in the case that the input of length p is given as a sequence of L monotonically increasing or decreasing segments. Suppose that the prefix-minimum has been computed up to a point i, i.e., that the value of $m_i = \min(x_0, \ldots, x_i)$ is known, and that $x_{i+1}, \ldots, x_{i+k}$ is a monotonically decreasing segment of the input of length k. Then, in order to compute $m_{i+1}, \ldots, m_{i+k}$, all that is required is to find the smallest ℓ such that $m_\ell < m_i$. This ℓ can be easily found using a binary search in $\lceil \lg k \rceil$ comparisons. We then have

$$m_{i+j} = \begin{cases} m_i & \text{if } j < \ell \\ x_{i+j} & \text{if } \ell \le j \le k. \end{cases}$$

If on the other hand $x_{i+1}, \ldots, x_{i+k}$ is a monotonically *increasing* sequence, all that is required in order to compute $m_{i+1}, \ldots, m_{i+k}$ is to compare x_{i+1} and m_i. In this case we have that $m_{i+1} = m_{i+2} = \cdots = m_{j+k} = \min(x_{i+1}, m_i)$. Using Lagrange multipliers we obtain that the number of comparisons is bounded above by

$$L \left\lceil \lg \frac{p}{L} \right\rceil. \tag{13}$$

Recall now the improved merge step described in Section 3.1. Each iteration of the binary search algorithm generates about half of the outputs of the max-filter that remained to be computed. Note that all values generated in one such iteration are consecutive in the output. Further, since these values are obtained from computing either R_i or S_i, they are either monotonically increasing or monotonically decreasing. Thus an application of the modified max filter algorithm also partitions each stretch of p outputs into at most $\lceil \lg p \rceil$ monotonic segments.

The improved opening filter algorithm is thus obtained by first applying the modified Gil-Werman max-filter algorithm, while preserving this partitioning of the output. Then, the results are fed into the modified Gil-Werman min-filter algorithm. The partitioning information is then used for an efficient implementation of the preprocessing stage in which prefix- and suffix-minima are computed. It follows from (13) that the preprocessing stage can be done in at most $O(\lg^2 p)$ comparisons. Since the merge step can be done in $O(\lg p)$ comparisons, we obtain:

Theorem 4 *There exists an algorithm which computes the opening filter, achieving* $C_1^o = C_1 + O(\frac{\lg^2 p}{p})$.

In other words, asymptotically computing the opening filter is not more expensive than computing just the max-filter.

6. Conclusions

We presented improvements of the Gil-Werman algorithm for running min and max filters. The average computational complexity was shown to be $1.25 + o(1)$ per element for a randomized algorithm, without any assumption on the distribution of the data, and $1.5 + o(1)$ for a deterministic algorithm. These improvements, which come close to the best known lower bound for the problem, were enabled by careful examination of the redundancies in the preprocessing and the merge steps of the Gil-Werman algorithm.

We continued to study a related problem, namely the computation of the min and the max filter together. We found that for independently distributed input elements, it is possible to compute the minimum and the maximum filters together in $2 + o(1)$ comparisons per data point. This is less than $2.5 + o(1)$ comparisons required by applying twice the best max filter algorithm.

The opening and closing filters which are similar to the problem of computing the min- and max-filters together, can be computed much more efficiently. We found algorithms for these filters using $1.5 + o(1)$ comparisons deterministically, or $1.25 + o(1)$ comparisons randomly, for worst case inputs.

All algorithms are readily extendible to higher dimensions.

Acknowledgements

Stimulating discussions of both authors with Reuven Bar-Yehuda of the Technion during the writeup of this paper are gratefully acknowledged. The second author is grateful to Renato Keshet from HP Labs. Israel, for intriguing discussions on efficient morphological operators.

References

1. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithmms.* MIT Press, Cambridge, Massachusetts, 1990.
2. D. Z. Gevorkian, J. T. Astola, and S. M. Atourian. Improving gil-werman algorithm for running min and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):526–529, May 1997.
3. J. Y. Gil and Z. Gutterman. Compile time symbolic derivation with C++ templates. In *Proceedings of the fourth Conference on Object-Oriented Technologies and Systems (COOTS'98)*, Santa Fe, New Mexico, May 1998. USENIX.
4. J. Y. Gil and R. Kimmel. Further improvements to the gil-werman's min and max filters. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.
5. J. Y. Gil and M. Werman. Computing 2-D min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):504–507, May 1993.
6. I. Pitas:1989:FAR. Fast algoirthsm for running ordering and max/min recalculations. *IEEE Transactions on Circuits and Systems*, CAS-36(6):795–804, June 1989.
7. J. Serra. *Image analysis and mathematical morphology.* Academic Press, New York, 1982.