A FAST ALGORITHM FOR MORPHOLOGICAL EROSION AND DILATION

C. Jeremy Pye *J. A. Bangham*

School of Information Systems,
University of East Anglia, Norwich, NR4 7TJ, UK.
Tel: +44 0 1603 456161; Fax: +44 0 1603 453345
`Email:cjp@sys.uea.ac.uk,ab@sys.uea.ac.uk`

ABSTRACT

This paper describes a new algorithm for performing erosion and dilation which is suitable for flat line-segment structuring functions, and which has a computational complexity that is independent of the structuring function size. Unlike other proposed algorithms, the computation time required by this method is directly proportional to the number of extrema within the signal being processed. This makes it particularly suitable for signals and images that have large and slowly varying segments.

1 INTRODUCTION

Since its evolution in the late 1970's, mathematical morphology [1] has been regarded as a powerful discipline suited to image processing and analysis. Underlying mathematical morphology are two primitive operations, known as erosion and dilation. All morphological operations can be constructed from a combination of erosions and dilations. A traditional and naive implementation of these two operations is based upon the technique employed for order-statistic filtering, which involves passing a window, or structuring function, over the signal and at each position recording the ith ranked value within the window. Erosion and dilation require finding either the maximum or minimum value within the window. This is a special case of order-statistic filtering because it does not require a sorting of the windowed values. Instead, selection of either the maximum or minimum requires only a single pass through the windowed values. This has a computational complexity of $\mathrm{O}(k)$ where k is the number of samples contained within the window or structuring function. Through the wise use of structuring element decomposition [2] it is often possible to reduce the number of computations required by splitting the structuring element into a number of smaller structuring functions. For example, if an image is dilated using a flat $(k \times l)$ size structuring function, the same result can be achieved using two separate structuring elements of size $(1 \times k)$ and $(l \times 1)$. This clearly reduces the number of computations per element from kl down to $k + l$. Other methods [3] involve maintaining local histograms of the values within the window. As the window moves from one position to the next many of the samples within the window remain. By monitoring the maximum and minimum values as the histogram adapts it is possible to reduce the computation required by a $(k \times l)$ structuring function from kl down to $2l$ computations per element. A drawback of this type of method is that the data must be represented by a fixed number of bits in order to a build a histogram. Astola [4] presents a similar method based upon a double heap structure which elevates this constraint.
Recently several algorithms designed to perform the fast calculation of the running maximum or minimum on a one dimensional signal have been presented. Pitas [5, 6] develops an algorithm based upon the ”divide and conquer” principle designed to exploit the redundancy in the running calculations, resulting in a reduced number of comparisons per element $\log_2 k$. Herk [7] proposes an alternative algorithm based upon a restricted distance transform. Using this method the number of comparisons per element is reduced to just 3 and is independent of the window size.
Whilst these methods demonstrate significant improvement over the general $\mathrm{O}(k)$ solution under certain circumstances these algorithms are rather inefficient.

2 FAST RUNNING MAXIMUM

Consider a discrete input signal consisting of elements $x_i \in \mathbb{R}, i \in \mathbb{Z}$. In this paper we will only consider the running maximum as the running minimum can be obtained by simply inverting the signal before and after processing. In order to simplify the explanation of the algorithm the structuring element will consist only of delayed input elements, that is

$$y_i = \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \ldots, x_i\right]. \tag{1}$$

although the more usual running maximum [5, 7] can be obtained by simply shifting the output signal by $-(n-1)/2$ elements.
In order to understand the inefficiencies of the present algorithms it is necessary to examine a number of input scenarios.

2.1 Monotonic

Given a monotonic signal as input, that is a signal whose elements are all of a constant value,

$$x_i = x_{i-1} \Longleftrightarrow \max\left[x_{i-1}, x_i\right] = x_i \tag{2}$$

and hence,

$$\begin{aligned} y_i &= \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \cdots, x_i\right] \\ &= x_i. \end{aligned} \tag{3}$$

2.2 Positive ramp

Given a signal in which the elements are ordered so that they form a positive ramp,

$$x_i \geq x_{i-1} \Longleftrightarrow \max\left[x_{i-1}, x_i\right] = x_i \tag{4}$$

and hence,

$$\begin{aligned} y_i &= \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \cdots, x_i\right] \\ &= x_i. \end{aligned} \tag{5}$$

2.3 Negative ramp

Given a signal in which the elements are ordered so that they form a negative ramp,

$$x_i \leq x_{i-1} \Longleftrightarrow \max\left[x_{i-1}, x_i\right] = x_{i-1} \tag{6}$$

and hence,

$$\begin{aligned} y_i &= \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \cdots, x_i\right] \\ &= x_{i-(k-1)}. \end{aligned} \tag{7}$$

2.4 Local maximum

Consider now a signal that has a positive ramp followed by a negative ramp, forming a single maximum at position j. That is,

$$x_i \begin{cases} \geq x_{i-1} & i \leq j \\ < x_{i-1} & i > j. \end{cases} \tag{8}$$

From subsection (2.2) the component of the signal that forms the positive ramp is unaffected by the filtering process, $i \leq j$. Also, from subsection (2.3) the component of the signal that forms a negative ramp is also unaffected, $i > j + (k-1)$. However, part of the signal does not fit into either of these cases. From equation (1), between the limits $j < i \leq j + (k-1)$ the output of the filter is given by

$$y_i = \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \cdots, x_i\right] \tag{9}$$

When the window is positioned at the extreme left of this limit so that $i = j + 1$, then

$$y_i = \max\left[x_{j-(k-2)}, x_{j-(k-3)}, \cdots, x_{j+1}\right] \tag{10}$$

clearly the window contains x_j which by definition is known to be the maximum value within the signal. Similarly, when the window is positioned at the extreme right of these limits, that is $i = j+(k-1)$, it is clear that the window also contains the element $x_{i-(k-1)} = x_j$. In conclusion,

$$y_i = \begin{cases} x_i & i \leq j \\ x_j & j < i \leq j + (k-1) \\ x_{i-(k-1)} & i > j + (k-1). \end{cases} \tag{11}$$

2.5 Local minimum

A local minimum occurs when two local maxima exist within the same signal. It is formed at x_j when a positive ramp follows a negative ramp, that is

$$x_i \begin{cases} < x_{i-1} & i \leq j \\ \geq x_{i-1} & i > j \end{cases} \tag{12}$$

The output due to the negative ramp component of the signal, is formulated in subsection (2.3),

$$y_i = x_{i-(k-1)} \qquad i \leq j \tag{13}$$

Similarly, from subsection (2.2), the component of the signal that forms a positive ramp is unaffected by the filtering process. That is,

$$y_i = x_i \qquad i \geq j + (k-1) \tag{14}$$

The output between the limits $j < i < j + (n-1)$ cannot be determined so easily as none of the previous cases apply. From equation (1) the output between these limits is given by,

$$y_i = \max\left[x_{i-(k-1)}, x_{i-(k-2)}, \cdots, x_i\right]. \tag{15}$$

This can be decomposed into two parts. Given that the signal for all $i > j$ forms a positive ramp the maximum of the samples between i and j is x_i. Also, because the signal for all $i \leq j$ forms a negative ramp, the maximum of the samples between i and j is $x_{i-(k-1)}$. Therefore,

$$y_i = \max\left[x_{i-(k-1)}, x_i\right] \qquad j < i < j + (k-1) \tag{16}$$

In conclusion given a local minimum the output is given as,

$$y_i = \begin{cases} x_{i-(k-1)} & i \leq j \\ \max\left[x_{i-(k-1)}, x_i\right] & j < i < j + (k-1) \\ x_i & i \geq j + (k-1). \end{cases} \tag{17}$$

2.6 Combinations local maxima and minima

Whilst the cases so far discussed are valid they are in general unlikely to occur individually. It is much more likely that a number of local maxima and minima will exist within the same signal. If local maxima occur in close proximity to each other then the simplifications obtained by separating the signal into various component parts, as demonstrated in subsection (2.5), do not apply.

Figure (1) shows a signal containing three local maxima at positions a, b, c. Where $a < b < c$ and $b-a \leq (k-1), c-b \leq (k-1)$ and the ordering of the maximum are $x_b < x_a < x_c$. Each local maxima is separated by a corresponding local minima at the positions d, e, f, where $a < d < b < e < c < f$. It is clear from this diagram that the output due to the maximum x_b is affected by both x_a and x_c.

In order to calculate the output given this interference caused by proximity, it is necessary to re-formulate the output with

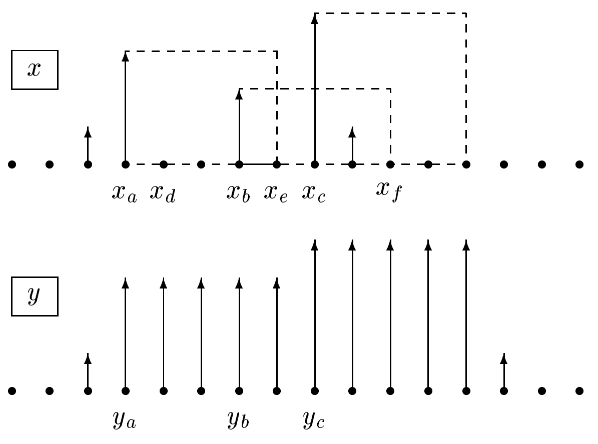

Figure 1: (Top) Shows a signal x which contains three local maxima x_a, x_b and x_c. The dotted line attempts to illustrate the problem caused by the close proximity of the individual maxima. (Bottom) Shows the result after applying a running maximum of width $k = 5$.

respect to each individual local maximum. Only the samples that constitute the individual local maxima are considered in each case.

Recall from the definition of a local maximum, equation (8), a local maxima at x_i results from samples that form a positive ramp $x_i \geq x_{i-1}$ followed by samples that form a negative ramp $x_i < x_{i-1}$. Therefore the collection of samples that form the local maximum x_a are denoted as x^A

$$x^A = \{x_0 \ldots x_a \ldots x_d\}. \tag{18}$$

From equation (11) the output given the input collection x^A, is

$$y_i^A = \begin{cases} x_i & 0 \leq i \leq a \\ x_a & a < i \leq a + (k-1) \\ x_a - (k-1) & a + (k-1) < i \\ & \leq d + (k-1). \end{cases} \tag{19}$$

The output given the collection of samples that constitute the local maximum x_b can also be calculated,

$$x^B = \{x_d \ldots x_b \ldots x_e\}. \tag{20}$$

and therefore,

$$y_i^B = \begin{cases} x_i & d \leq i \leq b \\ x_b & b < i \leq b + (k-1) \\ x_b - (k-1) & b + (k-1) < i \\ & \leq e + (k-1). \end{cases} \tag{21}$$

Indeed, the output due to every local maxima can be examined in this way,

$$x^C = \{x_e \ldots x_c \ldots x_f\}. \tag{22}$$

and

$$y_i^C = \begin{cases} x_i & e \leq i \leq c \\ x_c & c < i \leq c + (k-1) \\ x_c - (k-1) & c + (k-1) < i \\ & \leq f + (k-1). \end{cases} \tag{23}$$

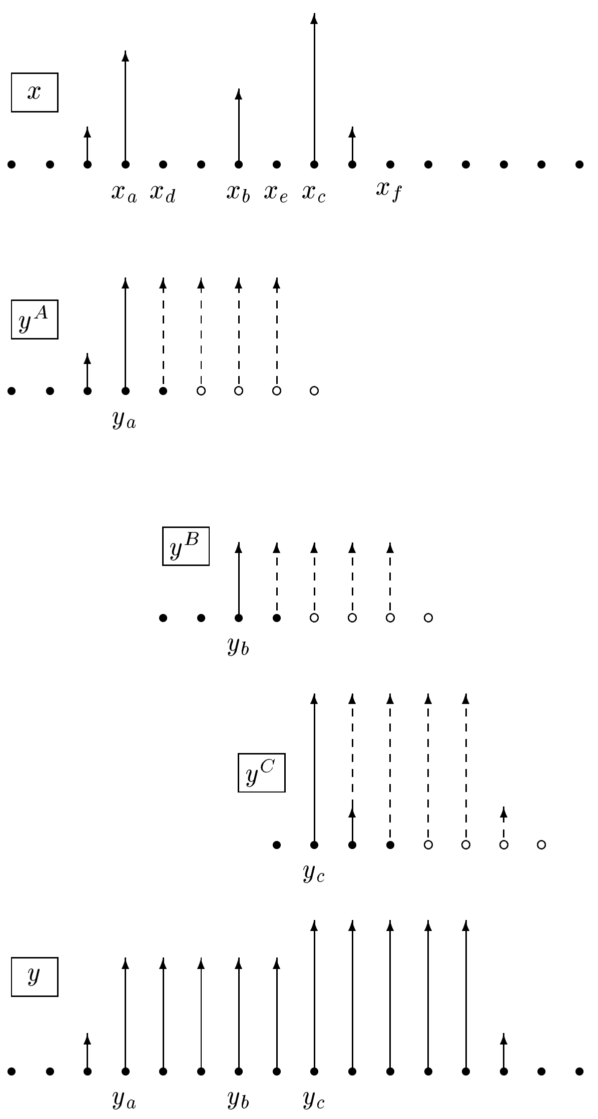

Figure 2: Illustration of processing the individual local maximum and recombining these intermediate results to form the output. The dotted lines (often occluded) represent the output, and the solid lines represent the input.

This procedure is illustrated graphically in figure (2). In order to calculate the output for the complete signal it is necessary to combine these intermediate results using a maximum operator. That is,

$$y_i = \max\left[x_i, y_i^A, y_i^B, y_i^C\right], \tag{24}$$

where all of the elements outside of the collections are assumed to be $-\infty$.

3 RESULTS AND DISCUSSION

It is clear from subsections (2.1-2.3) that the number of computations required to calculate the running maximum for a number of signals is low because many of the input samples remain unaffected by the filtering process. Subsection (2.6) describes a method of calculating the running maximum

which is suitable for all types of inputs whilst exploiting redundancy within the signal samples. Consider a monotonic signal as input, because the method depends upon finding and processing only the local maxima, the only calculation required is the initial search along the signal for the maxima, of which there are none. Alternatively, given a signal that has a local maxima at every other sample and therefore a maximum number of local maxima, it is clear that the amount of intermediate storage and computation required by this basic method is high, as it is dependent upon both the number of maxima and also the distance between them.
The underlying principle of this method is extending or stretching parts of the signal. This suggests that a suitable architecture for the data streams is a type of run-length-encoding (RLE). In fact whilst there is necessarily some computation overhead in encoding and decoding the input signal into RLE form, this type of architecture does allow the intermediate operations described in subsection (2.6) to be performed in place, elevating any storage problems. The algorithm is described below.

```
EXT = RLE.start;
while (EXT)
  if (EXT == local_maximum) then
    EXT.width = EXT.width + k;
  endif
  EXT = EXT.next;
end
EXT = RLE.start;
while (EXT)
  if (EXT == local_minimum) then
    remain = k;
    while (remain > 0)
      if (EXT.width > remain) then
        EXT.width = EXT.width - remain;
        remain = 0;
      else
        remain -= EXT.width;
        remove EXT;
        EXT = MIN(EXT.prev, EXT.next);
      endif
    end
  endif
  EXT = EXT.next;
end
```

The algorithm is two stage, firstly expanding all the local maxima and then removing the smallest samples from the minima in order to allow for the expansion.
For comparison the computational times for the classical running maximum selection algorithm, Herks [7] algorithm and the proposed extrema processing algorithm are given in figure (3). All of these algorithms were implemented on a SUN SPARCstation IPC and were written using the C++ language.

4 CONCLUSIONS

This algorithm seems appropriate for signals that have large slowly varying sections, such as images. Even in its worst case this algorithm appears to be comparable in performance to other methods.

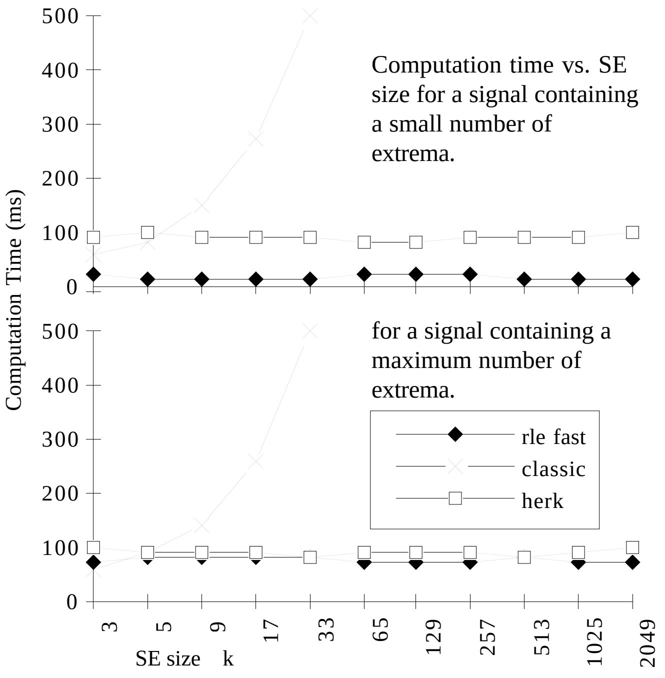

Figure 3: (TOP) Shows the computation times given a signal containing a minimum number of extrema. (BOTTOM) shows the computation times given a signal containing a maximum number of extrema. In each case the signal contains 8096 samples.

References

[1] J. Serra, *Image analysis and mathematical morphology Vol 1.* London: Academic Press, 1982.

[2] X. Zhuang and R. M. Haralick, “Morphological structuring element decomposition,” *Computer Vision, Graphics and Image Processing*, vol. 35, pp. 370–382, 1986.

[3] T. S. Huang, G. J. Yang, and G. Y. Tang, “A fast two-dimensional median filtering algorithm,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, pp. 13–18, 1979.

[4] J. Astola and T. G. Campbell, “On computation of the running median,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 572–574, 1989.

[5] I. Pitas, “Fast algorithms for running ordering and max/min calculation,” *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 795–804, 1989.

[6] D. Coltuc and I. Pitas, “Fast running max/min filters,” in *IEEE Workshop on Non-linear Signal and Image Processing*, pp. 871–874, June 1995.

[7] M. van Herk, “A fast algorithm for local minimum and maximum filters on rectangular and octogonal kernels,” *Pattern Recognition Letters*, vol. 13, pp. 517–521, July 1992.