Computing 2-Dimensional Min, Median and Max Filters

Joseph Gil and Michael Werman
Dept. of Computer Science
The Hebrew University of Jerusalem
91904 Jerusalem, Israel

October 7, 1996

Abstract

Fast algorithms to compute min, median, max or any other order statistic filter transforms are described and lower bounds for their computation are shown. The algorithms take constant time per pixel to compute min or max filters and polylog time per pixel, in the size of the filter, to compute the median filter.

1 Introduction

Order filters are widely used as an effective tool for reducing certain types of noise and periodic interference patterns in signals and images without severely degrading the signal [3]. One of the more useful properties of the median filter is that it many situations it doesn't blur edges and monotone changes in the signal but it cleans up sporadic noise. The min and max filters are the primitive operations for all of the so called morphological filters [4, 6].

In this paper we describe fast algorithms to compute min, median, max or any other order statistic filter transforms and show lower bounds for their computation.

Let $A[1..n, 1..n]$ be a two dimensional array and let the shape of the filter be a square with a side of size p, $p = 2q - 1$, then B is a min, median or max filter transform of A if $B[i,j]$ equals respectively the min, median or max of the multiset of size p^2

$$\left\{ A[l,m] \,\middle|\, \begin{array}{c} i-q < l < i+q \\ j-q < m < j+q \end{array} \right\}$$

The boundary elements can be treated as a special case, or totally ignored.

The observation that the size of the symmetric difference of the supports of $B[i,j]$ and $B[i,j+1]$ consists of only $2p$ elements, leads to an $O(n^2 p \log p)$ time algorithm by storing the support of $B[i,j]$ in a balanced binary tree. Computing any order statistic takes $O(\log p)$ time using this data structure. Updating the tree for $B[i,j+1]$ given $B[i,j]$ takes $O(p \log p)$ time per element.

We show an algorithm that takes constant time per element to compute the min or max filter and $O(\log^2 p)$ time per element to compute a median filter transform. We show an $O(\log p)$ lower bound for computing the median filter. The one dimensional median filter was treated in [2] for which they gave the tight upper and lower bound of $O(\log p)$ for the case of the shape being a segment of length p. Previous known algorithms for computing the 2-D median filter [3] took either $O(p^2)$ or $O(ph)$, h being the number of grey levels in the image, time per element. As the proposed algorithm has a small constant it is already superior for moderately sized windows.

2 2-Dimensional min and max filters

We present an algorithm for computing the filtered image for any semi-group operation (for example min, max or $+$), which takes only constant time per element computed, this is obviously optimal.

Let $\diamond$ be any semigroup operation. In order to calculate the $p \times p$ filter it suffices to compute row by row a 1-D p filter and on the resulting filtered image a column by column 1-D p filter, this follows from the associative law:

$$a_{1,1} \diamond a_{1,2} \cdots \diamond a_{1,p} \diamond a_{2,1} \cdots \diamond a_{p,p} = (a_{1,1} \diamond a_{1,2} \cdots \diamond a_{1,p}) \diamond (a_{2,1} \diamond a_{2,2} \cdots \diamond a_{2,p}) \diamond \cdots (a_{p,1} \diamond a_{p,2} \cdots \diamond a_{p,p})$$

Thus it suffices to show how to compute a one dimensional filter. In order to compute a one dimensional p filter, B, for a $(2p-1) \times 1$ image $a_1, a_2 \cdots a_{2p-1}$ we compute the following values in $O(p)$ time:

$$\begin{array}{rcll} R_i & = & a_i \diamond a_{i+1} \diamond \cdots \diamond a_{p-1} = a_i \diamond R_{i+1} & 0 < i \le p-1 \\ S_j & = & a_p \diamond a_{p+1} \diamond \cdots \diamond a_{p+j} = S_{j-1} \diamond a_{p+j} & 0 \le j \le p-1 \end{array}$$

$B[l]$ is $R_u \diamond S_v$ where $u = \frac{2l-p+1}{2}$ and $v = \frac{2l-p-1}{2}$ (the cases where the index is out of bounds is treated vacuously). Thus the computation of the p values takes $O(p)$ time or $O(1)$ amortized time per element.

The fact that $+$ is a semi-group operation, gives rise to a constant time per element mean filter algorithm.

The 1-D algorithm can be used iteratively on all axes to compute a $\overbrace{[\ \ }d]p \times p \times \cdots \times p$ semi-group filter in $O(d)$ time per element.

3 An $\Omega(\log p)$ lower bound per element for computing the median filter

Our lower bound is based on reduction of sorting to the computation of a two dimensional median filter. Let $x_1, x_2, \ldots x_{p^2}$ be a sequence of different values to be sorted. We claim that the power of the median filter combined with $O(p^2)$ preprocessing is enough to sort the sequence.

We arrange the sequence in a $p \times p$ matrix

$$X = \begin{pmatrix} x_1 & \cdots & x_p \\ \vdots & \ddots & \vdots \\ & \cdots & x_{p^2} \end{pmatrix}$$

The matrix X is then placed in the middle of a $(3p-2) \times (3p-2)$ matrix A.

Let $A(i,j)$ be the $(2p-1) \times (2p-1)$ square sub-matrix of A centered in $A[i,j]$, $p \leq i,j < 2p$. The center of each of the matrices $A(i,j)$ is a member of X, and all of them contain X as a sub-matrix. The median filter of size $2p-1$ generates the medians of the sub-matrices $A(i,j)$.

The rest of A is filled by x_- and x_+, where x_- is a value less than all the x_i and x_+ is a value greater than all the x_i. Clearly, both x_- and x_+ can be computed in $O(p^2)$ time.

Let $N_+(i,j)$ and $N_-(i,j)$ be the number of x_+ and x_- in $A(i,j)$. The layout of x_- and x_+ in A is devised so that the following properties of $A(i,j)$ hold:

1. The $(2p-1)^2$ elements of the top left sub-matrix $A(p,p)$ consists of:

   - $N_+(p,p) = (p-1)^2$

- p^2 members of X
- $N_-(p,p) = (p-1)^2 + p^2 - 1$

This requirement causes the median of $A(p,p)$ to be the minimum of X.

2. If both $A(i,j)$ and $A(i,j+1)$ exist then

   - $N_+(i,j+1) = N_+(i,j) + 1$
   - $N_-(i,j+1) = N_-(i,j) - 1$

   In other words, in each columnwise step over the $A(i,j)$ matrices, one of the x_- turns into x_+.

3. If both $A(i+1,j)$ and $A(i,j)$ exist then

   - $N_+(i+1,j) = N_+(i,j) + p$
   - $N_-(i+1,j) = N_-(i,j) - p$

   In other words, in each rowwise step over the $A(i,j)$ matrices, exactly p of the x_- turn into x_+.

Satisfaction of the above properties guarantees that the median filter will sort the x_i's. $B[p,p]$ is the minimum of X and a row by row, column by column scan over B generates the rest of X in ascending order.

We give the layout of A for the case $p = 5$. For brevity x_- and x_+ are marked as $-$ and $+$, and since the exact placement of $x_1, x_2, \ldots x_{p^2}$ in X is not important they are collectively marked as x's.

$$
\begin{array}{ccccccccccccc}
- & - & - & - & - & - & - & - & - & - & - & - & + \\
- & - & - & + & - & - & - & - & - & - & - & + & + \\
- & - & + & + & - & - & - & - & - & - & + & + & + \\
- & + & + & + & - & - & - & - & - & + & + & + & + \\
+ & + & + & + & x & x & x & x & x & + & + & + & + \\
+ & + & + & - & x & x & x & x & x & + & + & + & - \\
+ & + & - & - & x & x & x & x & x & + & + & - & - \\
+ & - & - & - & x & x & x & x & x & + & - & - & - \\
- & - & - & - & x & x & x & x & x & - & - & - & - \\
- & - & - & - & + & + & + & + & + & - & - & - & + \\
- & - & - & + & + & + & + & + & + & - & - & + & + \\
- & - & + & + & + & + & + & + & + & - & + & + & + \\
- & + & + & + & + & + & + & + & + & + & + & + & +
\end{array}
$$

The reader can easily verify that the desired properties hold in this layout. This symmetrical construction is readily generalized for all values p.

Theorem 1 *Amortized time for computing $B[i,j]$ (in the algebraic decision tree model) is $\Omega(\log p)$ if $n \geq 3p-2$.*

Proof Let $n = 3p-2$. The time to sort X is $\Omega(p^2 \log p)$. All the preprocessing can be done in $O(p^2)$ time and after it the median filter on p^2 points can be used to sort.

If n if greater than $3p-2$ then the outputs from the median filter can be selectively pruned so that they can be described as the composition of $(3p-2) \times (3p-2)$ sized totally independent median filter problems. This decomposition permits sorting many independent matrices. ∎

4 A 2-dimensional median filter algorithm

As in the lower bound proof we are interested only in the case where $n = 3p-2$. The generalization for $n > 3p-2$ is straight forward. Our algorithm will produces the p^2 median transform values in $O(p^2 \log^2 p)$ time, which is $O(\log^2 p)$ time per element.

As a building block the algorithm uses a balanced binary search tree (BBST) scheme. Such a scheme could be AVL [1], 2-3 trees [1] or even, since only total time is of concern, self-adjusting binary search trees [5]. Note that if the size of each sub-tree is stored in the node defining it, then the median of the set represented in the BBST can be found in logarithmic time, without affecting the logarithmic time behavior of the other tree operations.

The algorithm computes the matrix B column by column, where an entire column of B is computed in each iteration. When the algorithm is in its j^{th} phase all elements of the p windows $A(i,j)$, $p \leq i < 2p$, ($A(i,j)$ being the $p \times p$ window centered at $A[i,j]$), are stored. The main idea for saving time is the fact that although each window has p^2 elements, the windows are not disjoint.

In each phase the $(2p-1)p$ distinct elements of the p windows, are stored and manipulated. All $(2p-1)p$ elements are stored together in a BBST, using the pixel value as a search key. This BBST will be called the *main BBST*. Search and update operations on the *main BBST* take $O(\log p)$ operations

Since positional information is lost in the *main BBST* extra information has to be stored. In addition to its key, each node in the *main BBST* has an array of size $2p-1$ that stores in the i^{th} place the number of elements in A that are in its i^{th} row and the current p columns and are in the sub-tree rooted at the node. This array constitutes the leaves of a fixed structure *row BBST*. In the *row BBST* the search key is the row number. In addition to the search key, each node in the *row BBST* stores the sum of its leaves.

The advantage of using *row BBST* is that partial sums of the array can be easily computed. Change of one of the values stored in the array will cause no more than $O(\log p)$ updates operations to a single *row BBST*. Furthermore, the sum of any consecutive sub-array can be computed in $O(\log p)$ time.

Given this tree of trees structure, the median of $A(i,j)$ is computed. Selection in a BBST given subtree cardinality is $O(\log p)$ time and given the *row BBST* the cardinality of each subtree can be computed in $O(\log p)$ time, for a total of $O(\log^2 p)$ time to compute the median of a window. Perhaps a better understanding of this process can be gained if we view the *main BBST* as a compressed representation of p BBST of the $A(i,j)$'s for some fixed j. In each phase the medians of p such windows are computed from the compressed trees.

Basic update operations to this tree of trees structure require $O(\log^2 p)$, in proceeding to the next column of A exactly $2p-1$ elements have to be removed and $2p-1$ elements have to be inserted into the *main BBST*. This phase maintenance takes $O(p \log^2 p)$ which can be charged to the p medians produced. Initial erection of the tree takes $O(p^2 \log^2 p)$ time, which charges to each of the p^2 medians produced another $O(\log^2 p)$. Resulting in a total of $O(\log^2)$ time time per element processed. The memory requirement is $O(p^2)$.

5 Weighted Median

The case of a weighted median filter where each element is repeated a certain number of times depending on its location can also be treated in a like manner, details will appear in the final version.

6 Higher Dimensions

Using similar ideas a d-dimensional filter with window of size p^d will be able to be computed in time $O(\log(p^d))$, details will appear in the final version.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[2] J. Gil, W. Steiger, and A. Wigderson. *Running Medians*. To be published.

[3] T.S. Huang. *Two-dimensional Digital Signal Procesing II: Transfoms and Median Filters*. Springer-Verlag, 1981.

[4] J. Serra. *Image Analysis and Mathematical Morphology*. Academic, 1982.

[5] D.D. Sleator and R.E. Tarjan. Self adjusting binary search trees. *J. Assoc. Comput. Mach.*, 32:652–686, 1985.

[6] M. Werman and S. Peleg. Min max operators in texture analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-7:730–733, November 1985.