Fast Computation of a Class of Running Filters

Dinu Coltuc*, Ioannis Pitas
Department of Informatics
University of Thessaloniki, Greece

ABSTRACT

The paper focuses on the computation of a class of running filters defined as the n-ary extension of an associative, commutative and idempotent binary operation T on an ordered sequence of operands. The well-known max/min filters are the prominent representatives of the class. For any arbitrary window filter of size n, the existence of a fast algorithm of complexity $O(\log_2 n)$ T-operations is proven. A remarkable feature of the proof is its ability to generate a particular solution for every n. In addition to the theoretical results, practical implementation aspects regarding the flexibility of pipeline processors for fast computation of the 1D and 2D running filters are investigated.

Correspondence Address: Prof. I. Pitas, Department of Informatics, University of Thessaloniki, Thessaloniki 540 06, GREECE
Tel: +30-31-996304, Fax: +30-31-996304, email: pitas@zeus.csd.auth.gr

*Dinu Coltuc is with the Research Institute for Electrical Engineering, P.O. BOX 16-296, Bucharest, ROMANIA

1 Introduction

Max/min filters are widely used in nonlinear signal and image processing applications [1]-[4]. They are very attractive, since their computation requires only comparisons and no arithmetic operations. The computational complexity of max/min filters (i.e., the number of comparisons per sample) depends on the size of the filter window. A direct evaluation of the computational complexity may lead to $O(n)$ comparisons per sample. The particular case when the size of the filter window is a power of two is known to be of $O(\log_2 n)$ complexity. Thus, a fast "divide and conquer" algorithm and an efficient pipeline implementation were presented [5]. We mention that the development of the divide and conquer algorithm was based on the particular structure of the filter window. By extending the validity of the fast algorithm for arbitrary size windows, [6], we have figured out that the crucial point for the existence of running max/min algorithms is not the arithmetic properties of the number n but the intrinsic algebraic properties of the max/min operators, namely associativity, commutativity and especially idempotence. This observation leads us to a general approach by defining a running filter class based on an operation T possessing the above mentioned algebraic properties. T could be *maximum* or *minimum* as well as other operations that could be of interest in applications. Several examples of T operations are the *least common multiple* and the *greatest common divisor* on the lattice of integers, *union* and *intersection* on the lattice of partitions, etc.

The outline of the paper is the following. In section 2, the n-order running T filter is defined. The existence of a fast algorithm for arbitrary order n filter is proven in section 3. The solution proposed in section 3 is refined in section 4 towards the implementation of fully programmable pipeline processors for filtering on variable size windows. In section 5, the fast computation of two-dimensional running T filters is discussed. Finally, in section 6, we present our conclusions and we summarize the results of our research.

2 Running T filter

Let E be a set and let T be a binary operation, $T : E \times E \to E$. T is supposed to have the following properties:

1. associativity: $(xTy)Tz = xT(yTz)$, $\forall x, y, z \in E$;
2. commutativity: $xTy = yTx$, $\forall x, y \in E$;
3. idempotence: $xTx = x$, $\forall x \in E$.

Let X be a mapping, $X : N \to E$, $X(i) = x_i$.

Definition 1 *The n-th order running T filter on X, denoted by $RT_n(X)$, is the composite mapping $RT_n(X) \to E$, where:*

$$RT_n(X)(i) = (\ldots(x_i T x_{i-1}) T x_{i-2}) \ldots) T x_{i-n+1} \tag{1}$$

The ordered sequence (x_i) may represent an input signal (denoted by X). Thus, $RT_n(X)$ is the result of the running filter, i.e., the output signal. We shall further denote $y_i = RT_n(X)(i)$ and we shall consider the corresponding mapping $Y : N \to E$, $Y(i) = y_i$.

Since T is associative we have:

$$RT_n(X)(i) = x_i T x_{i-1} T x_{i-2} \ldots T x_{i-n+1} \tag{2}$$

We write (2) in a closed form by introducing a short-hand notation similar to the product of n operands:

$$RT_n(X)(i) = \mathbf{T}_{j=0}^{n-1} x_{i-j} \tag{3}$$

Since the computation of RT_n filter implies T-operations on position ordered operands, we introduce the delay operator D^m. For any sequence X, D^m is a composite mapping $D^m : X \to E$, $D^m(X)(i) = X(i-m)$. We assume that $X(i) = 0$ for $i < 0$. By definition we have $D^m(D^p(X)) = D^{m+p}(X)$ and $D^0(X) = X$. Using the delay operator, the running T filter is:

$$RT_n(X) = D^0(X)TD^1(X)\dots TD^{n-1}(X) = \mathbf{T}_{j=0}^{n-1}D^j(X) \tag{4}$$

We have defined, in a general setting, the class of running filters under study. (E,T) is a commutative semigroup and, besides, T is idempotent. No constraints have been imposed on E. The mapping X determines an ordered sequence on E that is further transformed by the natural n-ary extension of T. RT_n may also be approached as a mapping $RT_n : E \times E \dots \times E \to E$. However, its definition as a composite mapping is more related to the meaning of the problem at hand.

The direct evaluation of (1) requires a number $n-1$ of T-operations for the computation of each element y_i. We say that the computational complexity of the n-order running T filter is $O(n)$. It is easy to see that, in the evaluation of any two consecutive elements, e.g., y_i and y_{i+1}, all the operands but two are common:

$$y_i = (x_iTx_{i-1}\dots Tx_{i-n+2})Tx_{i-n+1} \tag{5}$$

$$y_{i+1} = x_{i+1}T(x_iTx_{i-i}\dots Tx_{i-n+2}) \tag{6}$$

This observation is of great importance in reducing the computational complexity of the running T filter. The idea is to factorize the computation in order to take advantage of the common terms, namely to use partial results on common terms in many computations. In the sequel, we prove that:

- the computation of the running T filters can be factorized to derive, for each order n, an algorithm of $O(\lceil \log_2 n \rceil)$ complexity, where $\lceil x \rceil$ is the least integer greater than x;
- the derived algorithm corresponds to a pipeline implementation;
- the algorithm is optimal in the sense that it defines the pipeline structure with the least complexity.

If (E,T) is a group there is the element 1_E, where $xT1_E = 1_ETx = x\ \forall x \in E$ and each element x has a symmetric $x^{-1} \in E$ so that $xTx^{-1} = x^{-1}Tx = 1_E$. In this case, the computational complexity of running T filter is two operations per element. Obviously, $y_{i+1} = x_{i+1}Ty_iTx^{-1}_{i-n+1}$. This is the case of moving average filter ($(Z,+)$ or $(R,+)$ are groups). Since this case is trivial, in what follows we analyze only the general setting of the problem, i.e., when (E,T) is a semigroup.

3 Factorization for arbitrary n

We introduce a flow graph to describe the computation of RT_n filters. Its elements are nodes connected by directional branches. The branches represent the flow of operands among nodes. The sequence transmitted by each branch is delayed by its corresponding delay operator. For simplicity reasons, we omit to label branches by the identity delay operator D^0. The nodes correspond to T operations among the operands of the incoming branches. Since T is a binary operation, we allow only two input branches for each node. No computation takes place in the input node. Nodes may be labeled with the result of the computation. Any number of branches can emerge from a node. In Fig. 1 an example of flow graph description is shown. If we denote by Y the output sequence, the computation described by this example is $Y = D^0(X)TD^m(X)$ or, in an equivalent form, $y_i = x_iTx_{i-m}$, $i = 1, 2, \dots$. For $m = 1$, Y is the output of the second-order running T filter on X.

Let us further analyze the cascade of k such simple structures (Fig. 2). For each node $j = 1, 2, \ldots, k$, we have the equation:

$$Y^j = Y^{j-1} T D^{m_j}(Y^{j-1}) \tag{7}$$

The output of the structure is obtained by observing that each node multiplies by 2 the number of operands involved in the computation. Thus, $Y^j = Y^{j-2} T D^{m_{j-1}}(Y^{j-2}) T D^{m_j}(Y^{j-2} T D^{m_{j-1}}(Y^{j-2}))$ $= Y^{j-2} T D^{m_{j-1}}(Y^{j-2}) T D^{m_j}(Y^{j-2})\ T D^{m_{j-1}+m_j}(Y^{j-2})$, and so on. For the final node k, there are 2^k operands, namely the input sequence X and the sequences obtained by all the possible combinations of the delay operators $D^{m_1}, D^{m_2}, \ldots, D^{m_k}$. The order of the operands may be changed with no effect on the result because T is commutative. Let I be the set of delay values $\{m_1, m_2, \ldots, m_k\}$ and let $\mathcal{P}(I)$ be the power-set of I. For each $I_j \in \mathcal{P}(I)$ we denote by $S(j)$ the sum of the elements of I_j. With the above notation we have the output of the structure in a closed form:

$$Y = \mathbf{T}_{j=0}^{2^k-1} D^{S(j)}(X) \tag{8}$$

We want to compute $RT_n(X)$ by using the structure presented in Fig. 2. To do this, we have to determine in what conditions, if any, equations (4) and (8) are equivalent. Notice that for n power of two, $n = 2^k$, the right sides of (4) and (8) have the same number of operands. The delay factors $I = \{m_1, m_2, \ldots, m_k\}$ have to be determined so that the operands in both equations are identical. The set $I = \{1, 2, \ldots, 2^{k-1}\}$ is a solution, since $S(j)$, $j = 0, \ldots, 2^k - 1$, takes all the values in $[0, n-1]$. (The k-bits binary representation of any integer j, $0 \leq j < 2^{k-1}$, gives an immediate proof of this assertion.) The computation of $RT_n(X)$ for $n = 2^k$ is done only in k operations, $k = \log_2 n$. In fact, we have recovered the particular solution of [5]. Let us further consider the general case when n is an arbitrary integer. Since the number of operands in the right-hand side of equations (4) and (8) is different, the idempotence property of T is of crucial importance for the existence of the solution. Equation (8) gives the RT_n of a sequence, if the following three conditions are met:

1. $2^k \geq n$, i.e., there are enough operands in (8);
2. $S(j) < n$, i.e., the delay operators in (8) do not exceed the range of delay operators in (4);
3. $S(j)$ takes all the values in the closed interval $[0, n-1]$, i.e., all the delay operators in (4) appear also in (8).

We shall prove that, for each n, there is a minimal set I that satisfies the above conditions. From condition 1, we have $k \geq \log_2 n$. k is integer and its minimum admissible value is $k = \lceil \log_2 n \rceil$. We shall prove the existence of the minimal solution by finding the set I. Let I' be the set $\{2^i,\ 0 \leq i \leq k-2\}$ and let $I = I' \cup \{n - 2^{k-1}\}$. The set I has k elements. For all $I_j \in \mathcal{P}(I)$, we have $0 \leq S(j) \leq n-1$. If $I_j = I$, the sum $S(j)$ takes its maximum value, namely $\sum_{i=0}^{k-2} 2^i + n - 2^{k-1} = 2^{k-1} - 1 + n - 2^{k-1} = n - 1$. This satisfies condition 2. Let us consider any integer p, $0 \leq p < 2^{k-1}$. Obviously, there is a subset I_j of I' such that $p = S(j)$. Let us consider $2^{k-1} \leq p < n$ and let $q = n - p - 1$. We have $0 \leq q < 2^{k-1}$ and, based on the same consideration as before, there is an $I_j \subset I'$ such that $q = S(j)$. In this case the sum of the elements of the set-difference $I \backslash I_j$ is exactly p. Consequently, $S(j)$ takes all the integer values in the interval $[0, n-1]$ and condition 3 is met. The set $I = \{1, 2, \ldots, 2^{k-2}, n - 2^{k-1}\}$ is a solution for the general problem of arbitrary n.

Our proof shows that, for each n, the n-order running T filter can be computed by using the structure presented in Fig. 2, i.e., a pipeline of k simple structures as presented in Fig. 1, $k = \lceil \log_2 n \rceil$. The computational complexity of RT_n decreases from $O(n)$ to $O(\lceil \log_2 n \rceil)$.

The computational complexity of the derived structure is k operations. As we have proven, for a given n, k determined as above is the minimum number to satisfy the problem; from here we infer that the solution is optimal. However, the optimality of k should be understood in connection with the problem we have stated, namely to compute the T running filter by using the pipeline structure of Fig. 2. Therefore, we have found an optimal structure for the given pipeline processors.

The number of intermediate operands that must be stored for each result is given by all the delays involved in the computation, i.e., $\sum_{i=1}^{k} m_i = n-1$. The proof deserves a special mention since it is constructive. It gives a particular solution for the delay operators $D^{m_1}, D^{m_2}, \ldots, D^{m_k}$, namely $D^1, D^2, \ldots, D^{2^{k-2}}, D^{n-2^{k-1}}$. For example, for $n = 15$, the delay constants are $\{1, 2, 4, 7\}$, i.e., the delay operators are D^1, D^2, D^4 and D^7. The filter is computed by using the 4 cascaded structures with the delays specified above.

From the commutativity of T it follows that any permutation of the set I of delay constants is also a solution of the problem. The entire set of solutions for any given n can be determined if we approach the problem as an integer programming one. This means to determine all the sets $\{m_1, m_2, \ldots, m_k\}$, $k = \lceil \log_2 n \rceil$, such that conditions 2 and 3 from section 2 are fulfilled. Each m_i is bounded, $0 < m_i < n$. The problem may be solved by exhaustive search. The equations $S(j) = p$, $p = 0, 1, \ldots, n-1$ lead to efficient branch and bound of solutions space. The analysis of the complexity of the integer programming problem is beyond the purpose of this paper and, besides, our proof gives directly a particular solution. However, for a broader view of the problem, we give all the distinct solutions when n ranges $[5, 16]$ (Table 1). The last column of Table 1 shows the total number of solutions derived by permutation.

4 Structures for variable window length

The above mentioned structure is tailored to any particular T operation by defining the processing task of the nodes. T operation may be implemented either by software or by hardware. The delay operators are memory cells or, equivalently, registers. Their size depends on the nature of data required by the problem at hand. If the hardware implementation is needed, the structure presented in Fig. 2 is a pipeline processor. Its clock is determined by the processing time required by the computation of T operations.

For almost all practical applications of such running filters, the window size is a flexible design parameter. This problem has a serious impact on hardware implementations and demands for programmable processors. We shall propose two solutions to this problem.

The first one is generated by the particular solution to the fast algorithm given in section 3. We have shown that for $2^{k-1} < n \leq 2^k$ the running filter is implemented by k cascaded simple structures. Each structure is composed by a T processor fed on with operands through two delay lines: an identity delay line and a fixed value delay line, m_i. The delay constants are 1, 2, ..., 2^{k-2} and $n - 2^{k-1}$. If we keep the first $k-1$ structures fixed and we modify the value of the delay operator of the last stage in the range $1 \leq m_k \leq 2^{k-1}$, we cover all the n-order running T filters for window sizes in the interval $[2^{k-1}+1, 2^k]$. Furthermore, if $m_k = 0$, both inputs to the k-th T processor have the same value and, consequently, the processor has no effect in the computation. The structure operates with only $k-1$ stages. It follows that a fully programmable n-order running T filter for $2 \leq n \leq 2^k$ is implemented by using a programmable delay line in each stage. For stage i, $i \geq 2$, the delay line operates in the range $[0, 2^{i-1}]$. If the value of n is in the range $2^{j-1} < n \leq 2^j$, the delay values are: $m_i = 0$ for $j+1 \leq i \leq k$, $m_i = 2^{j-1} - (2^j - n) = n - 2^{j-1}$ for $i = j$ and $m_i = 2^{i-1}$ for $1 \leq i < j$, respectively. For example, for $n = 2, \ldots, 15$, when n takes the value 7 the delay constants are $\{1, 2, 3, 0\}$. If n is 3, the delay constants become $\{1, 1, 0, 0\}$. For a pipeline implementation, the total delay time, introduced by propagating the signal through the stages with delay constants 0, has to be taken into account. In the worst case, there are $k-1$ stages with zero delay constants. If this time is not admissible, a solution still exists by multiplexing the output of each stage and selecting only the output of the last working stage, i.e., stage j.

There is a second approach to this problem. The basic idea is to implement the architecture for the maximal window size of interest, n, and to perform the computation for windows smaller than n by introducing dummy operands up to n. Let m be the size of a window, where $2^{k-1} < m \leq n$. We adjust the input data taking care to have in each group of n data exactly m original operands.

The dummy operands may be either the unity element on E (if 1_E exists), or any values of some operands within the group of m (by idempotence of T). We shall give next a procedure for the expansion of the input sequence. Let ϕ be an increasing mapping, $\phi : M \to K$, where M and K are the set of integers $\{0, 1, \ldots, m-1\}$ and $\{0, 1, \ldots, n-1\}$, respectively. (The condition imposed on ϕ to be increasing, preserves the order of the samples in the expanded sequence.) Let J be the set $\phi(M)$. The set J has m distinct values and the set $K \backslash J$ has p values, $m + p = n$. Let $\psi : Z \to Z$ be a mapping defined by:

$$\psi(i) = \lfloor i/m \rfloor n + \phi(i \bmod m) \tag{9}$$

where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x and mod is the modulus operation. ψ is an increasing mapping too. The expanded sequence of the input (x_i) is generated by the following procedure: each sample x_i is placed in position $\psi(i)$; between samples $x_{\psi(i)}$ and $x_{\psi(i+1)}$ $\psi(i+1) - \psi(i) - 1$ dummy samples are inserted. $\psi(0)$ dummy samples are inserted in front of $x_{\psi(0)}$. By construction, the sample placed at position j belongs to the initial sequence if $j \bmod n \in J$. Similarly, if $j \bmod n \in K \backslash J$, a dummy sample is inserted on position j. Let us consider a group of n consecutive samples of the expanded sequence and let j be the index of its first sample. Their indices are $j + q$, where $0 \leq q < n$. $(j + q) \bmod n$ takes distinct values for distinct values of q. (If $q_1 \neq q_2$ and $(j + q_1) \bmod n = (j + q_2) \bmod n$, we have $(q_1 - q_2) \bmod n = 0$. Thus $q_1 - q_2 = rn$. Since q_1, $q_2 < n$ then $q_1 = q_2$). We get n distinct values for $(j+q) \bmod n$, i.e., $0, \ldots, n-1$. The set $\{(j+q) \bmod n\}$ is identical with set K. Therefore, exactly m samples belong to the initial sequence and p samples are dummy ones. Since j has been taken arbitrarily, the property is valid for each group of n consecutive samples. Let us consider a window that starts with a dummy sample z_i. When the window moves one position, z_i is discarded and a new sample is added. According to this expansion procedure, the new sample is a dummy sample as well, z_{i+n}. Since dummy samples do not influence the computation, the result is the same in both windows and one sample must be discarded from the output sequence. Let $\phi : I \to K$ be:

$$\phi(i) = \begin{cases} 2i & \text{if } i < p \\ i + 2p & \text{otherwise} \end{cases} \tag{10}$$

Each inserted sample z_i takes the value of its previous sample x_i. The expanded sequence is:

$$x_0, z_0, x_1, z_1, \ldots, x_{p-1}, z_{p-1}, x_p, x_{p+1}, \ldots, x_{m-1}, x_m, z_n, \ldots \tag{11}$$

The mapping (10) is appropriate for hardware implementation with no apparent loss of performance due to dummy insertion/elimination. The idea is to keep input/output data samples synchronized and, meantime, to insert hidden clock cycles for the processing of dummy samples. Since each dummy sample is a copy of the previous sample and if the T processors are fast enough, we may compute two cycles instead of one when, according to (11), a dummy sample has to be inserted. This means to load two times the data sample x_i (corresponding to x_i and z_i), to forward it in the pipeline and to output only one result. Therefore, when no dummy samples have to be inserted, the pipeline and the input/output data have the same timing, otherwise the processing chain executes two clock cycles instead of one. The domain of operation can be further extended to $[2, n]$ by multiplexing the output of the T processors and selecting the command timing and the output according to the range of n. The second solution has the advantage of eliminating the programmable delay lines required by the first solution by using an adequate clock signal. More aspects regarding non-pipeline schemes derived by optimizing the sequence expansion scheme are given in [7].

5 2D Filtering

In the previous sections we have presented the 1D n-order running T filter; the 2D case immediately follows. Thus, the mapping X extends naturally to a 2-dimensional one, $X : N \times N \to E$, $X(i, j) =$

$x_{i,j}$, where $0 \leq i,j < N$. The delay operators are mappings $D^{p,q} : X \to E$, $D^{p,q}(X)(i,j) = X(i-p, j-q)$. They preserve the properties shown in section 2. With the above notations we have:

Definition 2 *The 2D running T filter on X for a $n \times m$ window size is the composite mapping $RT_{n,m}(X) \to E$ defined by:*

$$RT_{n,m}(X)(i,j) = \mathbf{T}_{p=0,q=0}^{p=n-1,q=m-1} x_{i-p,j-q} \tag{12}$$

Or, alternatively:

$$RT_{n,m}(X) = \mathbf{T}_{p=0,q=0}^{p=n-1,q=m-1} D^{p,q}(X) \tag{13}$$

The number of operands in equations (12,13) is $n \times m$. Therefore, the direct computation of $RT_{n,m}(X)$ requires $n \times m - 1$ operations for each output sample, i.e., the computational complexity of 2D running transform is $O(n \times m)$.

Let us assume, for simplicity, that $n = m$ and let us investigate the computation of $RT_{n,n}(X)(i,j)$ in the j direction (we consider directions i and j as rows and columns). For consecutive outputs, $n^2 - n$ operands are common and n operands are different. This means that, when the filter window moves one position a new column of n operands is added, let say column j. Simultaneously, the n operands of the column $j - n$ are thrown away. The passage from one window position to the next one may be seen as a sequential procedure that takes place in n steps; in each step an operand is discarded and a new one is added. Thus, the 2D mapping X is transformed to a 1D mapping, and the 2D running filter can be implemented by a 1D n^2-order running filter whose output preserves only one out of n consecutive results. By using the algorithm proposed in section 3, the computational complexity of the 2D running filter becomes $O(n\lceil 2\log_2 n \rceil)$ operations per sample, ($\lceil 2\log_2 n \rceil$ T-operations for each result, and n results have to be computed until a final result is obtained). This approach requires $n^2 - 1$ storage cells for the intermediate results.

Better performance in terms of computational complexity is obtained by taking into account the separability of the 2D filter. By T associativity and commutativity, we may write (12) in a slightly different form:

$$RT_{n,n}(X)(i,j) = \mathbf{T}_{p=0}^{p=n-1} \left(\mathbf{T}_{q=0}^{q=n-1} x_{i-p,j-q}\right) \tag{14}$$

Thus, the computation is done in two passes. First the 1D n-order filter is computed on rows and then a second 1D n-order filter is computed on the columns of the results. By using the fast algorithm, the computational complexity of the 2D filter is $O(2\lceil \log_2 n \rceil)$ at the cost of two passes over the 2D sequence (the entire intermediate sequence have to be stored). Since the intermediate sequence can be stored in place, this is not a major drawback of the separable implementation. The decrease of the computational complexity for the separable approach is significant.

6 Conclusions

We have proven the existence of a fast computation method, data independent, for a class of 1D and 2D running filters. The definition of the filters under study is based on the natural n-ary extension of a binary operation T. The cornerstone of our approach is given by the algebraic properties of T, namely associativity, commutativity and idempotence. Our approach has considered a general pipeline computational structure that has been constrained to compute the desired filter output by specifying its elements, i.e., the number of T processors and the values of delay constants. We have proven that, for each window size n, there is a set of delay constants, namely $\{1, 2, \ldots, 2^{k-2}, n - 2^{k-1}\}$, so that the computation is done in $k = \lceil \log_2 n \rceil$ T-operations per sample. The resulted computational complexity is optimal with respect to the considered pipeline structures. Programmable structures for variable size windows have been investigated and two methods have been discussed.

The most obvious application of our results is for the computation of max/min filters. This is of large interest in signal and image processing, notably in mathematical morphology. Other applications may be taken into consideration as well, since the assumptions on T are fulfilled also by other operations (e.g., the greatest common divisor, the least common multiple, union and intersection).

References

[1] R. M. Haralick, S. R. Sternberg, X. Zhuang, "Image Analysis Using Mathematical Morphology", *IEEE Trans. on PAMI*, vol. PAMI-9, no. 4, pp. 532-530, July 1987.

[2] M. Werman, S. Peleg, "Min-Max Filters in Texture Analysis", *IEEE Trans. on PAMI*, vol. PAMI-7, pp. 730-733, 1986.

[3] I. Pitas, A. N. Venetsanopoulos, *Nonlinear digital filters: principles and applications*, Kluwer Academic, 1990.

[4] P. W. Verbeek, H. A. Vrooman, L.J. Vliet, "Low-Level Image Processing by Max-Min Filters", *Signal Processing*, vol. 15, no. 3, pp. 249-258, Oct. 1988.

[5] I. Pitas, "Fast Algorithms for Running Ordering and Max/Min Calculation", *IEEE Trans. on Circuits and Systems*, vol. 36, no. 6, pp. 795-804, 1989.

[6] D. Coltuc, I. Pitas, "Fast Running Max/Min Filters", *IEEE Workshop on Nonlinear Signal and Image Processing*, June 1995, Greece, pp. 871-874, 1995.

[7] D. Coltuc, I. Pitas, "On Fast Running Max/Min Filtering", *IEEE Trans. on Circuits and Systems - II: Analog and Digital Signal Processing*, vol 44 no. 8, pp. 660-663, 1997.

Table 1. Solutions for n ranges $[5, 16]$

n	Distinct solutions	Total solutions
5	$\{1, 1, 2\}$	3
6	$\{1, 1, 3\}$, $\{1, 2, 2\}$	6
7	$\{1, 2, 3\}$	6
8	$\{1, 2, 4\}$	6
9	$\{1, 1, 2, 4\}$, $\{1, 1, 3, 3\}$, $\{1, 2, 2, 3\}$	30
10	$\{1, 1, 2, 5\}$, $\{1, 1, 3, 4\}$, $\{1, 2, 2, 4\}$, $\{1, 2, 3, 3\}$	48
11	$\{1, 1, 3, 5\}$, $\{1, 2, 2, 5\}$, $\{1, 2, 3, 4\}$	48
12	$\{1, 1, 3, 6\}$, $\{1, 2, 2, 6\}$, $\{1, 2, 3, 5\}$, $\{1, 2, 4, 4\}$	60
13	$\{1, 2, 3, 6\}$, $\{1, 2, 4, 5\}$	48
14	$\{1, 2, 3, 7\}$, $\{1, 2, 4, 6\}$	48
15	$\{1, 2, 4, 7\}$	24
16	$\{1, 2, 4, 8\}$	24

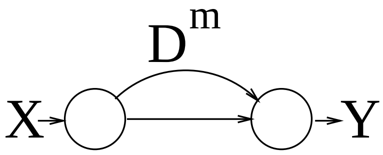

Fig. 1.

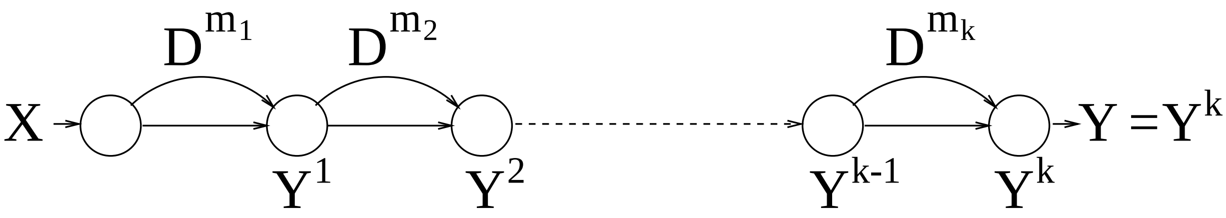

Fig. 2.