An Exact Algorithm for Optimal MAE Stack Filter Design

Domingos Dellamonica, Jr., Paulo J. S. Silva, Carlos Humes, Jr., Nina S. T. Hirata, and Junior Barrera

***Abstract*—We propose a new algorithm for optimal MAE stack filter design. It is based on three main ingredients. First, we show that the dual of the integer programming formulation of the filter design problem is a minimum cost network flow problem. Next, we present a decomposition principle that can be used to break this dual problem into smaller subproblems. Finally, we propose a specialization of the network Simplex algorithm based on column generation to solve these smaller subproblems. Using our method, we were able to efficiently solve instances of the filter problem with window size up to 25 pixels. To the best of our knowledge, this is the largest dimension for which this problem was ever solved exactly.**

***Index Terms*—Boolean lattice, column generation, filter design, network flows, positive Boolean function, stack filter.**

I. Introduction

THE interest in stack filters is quite clear from the large amount of literature devoted to them [1]–[16]. Stack filters are nonlinear filters that commute with thresholding, i.e., the application of the filter directly on a gray-level (multilevel) signal followed by thresholding at any level t leads to the same signal as the one obtained by first thresholding the gray-level signal at level t and then applying the filter to the corresponding binary cross section [17]. The best known subclass of stack filters are the median filters (see, e.g., [18] and [19]).

A stack filter with respect to a sliding window of size d, when restricted to binary images, can be expressed as a Boolean function on d variables. For instance, the median filter on a window of size 3 can be modeled as the Boolean function $\psi(u_1, u_2, u_3) = u_1u_2 + u_1u_3 + u_2u_3$ on three binary variables u_1, u_2, and u_3. It has been shown that the only filters that have the property of commuting with thresholding are those that, restricted to binary images, correspond to monotone (positive) Boolean functions [1], [17].

The name "stack filters" refers to the fact that gray-level results can be obtained by stacking (summing) the binary signals resulting from the application of the filter to the multiple level cross sections [1]. Moreover, the mean absolute error (MAE) of stack filters can be modeled as a linear combination of the mean absolute errors of the corresponding Boolean function with respect to the multiple threshold levels [2]. These two facts reduce the problem of designing stack filters to the problem of designing monotone Boolean functions.

Coyle *et al.* showed that the design of MAE optimal stack filters with respect to a sliding window of size d can be formulated as a zero-one integer linear program (IP), with 2^d variables and $\Theta(d\,2^{d-1})$ constraints [2]. Even for moderate values of d, this formulation may not be feasible for practical purposes.

As a consequence, alternative approaches for the design of stack filters have been proposed. Most methods follow two main lines: adaptive search and graph search.

Adaptive search techniques use the number of patterns observed in the input images and update the filter based on the respective outputs in ideal images [3], [8]. Such algorithms enforce monotonicity of the function only after a given amount of data are processed. Recent improvements along this line appear in [12].

Graph search techniques are also an important class of methods for filter design [10], [11], [13], [14]. For example, [16] presents a unifying view for such methods. They all exploit the Boolean lattice representation of the window space. In the context of morphological increasing set operator design, similar algorithms to find optimal Boolean functions have also been proposed [20]–[22].

Techniques in both lines rely on heuristics to overcome computational limits. Thus, they compute suboptimal filters. The most efficient algorithm reported so far is Yoo's adaptive algorithm [12], which we will call `train`. One of the main parameters in this algorithm is the number of iterations, how many scans on the training data, it is allowed to do. As this value increases, the resulting filter asymptotically converges to an optimal one. However, there are no results concerning either the rate of convergence or its precision.

In this paper, we propose a new algorithm to find a minimum MAE stack filter. Our method borrows ideas from the two approaches outlined above. The rest of the paper is organized as follows. Section II introduces some notations and reviews basic concepts of minimum MAE stack filters. Section III builds our algorithm in many steps, emphasizing the role of each technique employed. Finally, Section IV presents our computational results, followed by our conclusions in Section V.

To our knowledge, the algorithm proposed in this paper is the first method that gives an exact solution to large instances of the minimum MAE stack filter problem in an acceptable training time.

Manuscript received December 15, 2005; revised August 4, 2006. D. Dellamonica, Jr., was supported by FAPESP under Grant 2004/11501-1; P. J. S. Silva was supported in part by CNPq, Brazil, under Grant 304133/2004-3 and in part by PRONEX-Optimization; C. Humes, Jr., was supported in part by CNPq and PRONEX Optimization; N. S. T. Hirata was supported in part by CNPq, Brazil, under Grant 303237/03-1; J. Barrera was supported in part by CNPq. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Giovanni Ramponi.

The authors are with the Computer Science Department of the Institute of Mathematics and Statistics, University of São Paulo, 05508-090 São Paulo SP, Brazil (e-mail: ddj@ime.usp.br; pjssilva@ime.usp.br; humes@ime.usp.br; nina@ime.usp.brip; jb@ime.usp.br).

Digital Object Identifier 10.1109/TIP.2006.888358

II. Background

A. Preliminaries and Notation

In this section, we briefly describe the notation used in this text and recall some definitions.

We use lower case letters for vectors and upper case letters for matrices. The $n \times n$ identity matrix will be denoted by I_n, or simply I if n is clear from the context. The all-zero vector on n coordinates is denoted 0_n, the all-one vector on n coordinates is denoted 1_n. Again, when n is clear from the context, we may drop the subscript.

Let x and y be vectors indexed by the same set S of coordinates, say $x = (x_a)_{a \in S}$ and $y = (y_a)_{a \in S}$. We can define a partial order $\leq$ between these vectors as follows:

$$x \leq y \quad \text{if } x_a \leq y_a \text{ for all } a \in S. \tag{1}$$

Similarly, we define the partial order $\geq$.

The Euclidean *inner product* of x and y will be denoted by $\langle x, y \rangle$ and is defined as $\langle x, y \rangle := \sum_{a \in S} x_a y_a$.

1) Matrix Representation by Blocks: Given an $m \times n$ matrix A and an $m \times l$ matrix B, we denote by $[A\, B]$ the $m \times (n + l)$ matrix formed by joining the n columns of A and the l columns of B. Similarly, if A is an $m \times n$ matrix and B is an $k \times n$, we denote by

$$\begin{bmatrix} A \\ B \end{bmatrix}$$

the $(m + k) \times n$ matrix formed by joining the m rows of A and the k rows of B.

In this paper, we use the notion of a *Boolean lattice*. Let d be a positive integer and consider the set of all vectors in $\{0,1\}^d$ with partial order $\leq$ given by (1). Denoting $\mathcal{R} = \{0,1\}^d$ we have that $(\mathcal{R}, \leq)$ is a Boolean lattice of dimension d. We denote by $\operatorname{supp}(u) = \{i \mid u_i = 1\}$ the support of u.

B. Minimum Mean Absolute Error Stack Filters

A translation-invariant binary image operator with respect to a window W of size d can be modeled as a Boolean function $\psi : \{0,1\}^d \rightarrow \{0,1\}$. A Boolean function ψ is monotone if $\operatorname{supp}(u_1) \supseteq \operatorname{supp}(u_2)$ and $\psi(u_2) = 1$ imply $\psi(u_1) = 1$ (or, equivalently, $\operatorname{supp}(u_1) \subseteq \operatorname{supp}(u_2)$ and $\psi(u_2) = 0$ imply $\psi(u_1) = 0$). The idea of support comparison corresponds to the natural extension [see (1)] of order $\leq$ defined on integers, by $\operatorname{supp}(u) \subseteq \operatorname{supp}(v) \Leftrightarrow u \leq v$.

The minimum mean absolute error (MMAE) stack filter with respect to a window W of size d is the one that, among all stack filters with respect to W, possess the smallest mean absolute error. In order to consider any notion of mean error, we have to accept some statistical assumptions. Following [2], we suppose that images to be filtered and their respective ideal desired images are realizations of a jointly stationary process.

In practice, the probabilities that determine the MAE of a stack filter are estimated from the binary cross sections of training images. Given sample pairs of observed-ideal images, let $f_{i,u}^t$ be the number of times a pattern $u \in \mathcal{R}$ is observed in the cross sections at level t with ideal value $y = i$, $i \in \{0,1\}$.

Let d denote the size of W, let k be the number of gray tones in the images, and $f_{i,u} = \sum_{t=1}^{k} f_{i,u}^t$, $i \in \{0,1\}$. Then, the minimum MAE stack filter with respect to W corresponds to a monotone Boolean function ψ that minimizes (see [2], [10], and [16] for details)

$$\begin{aligned}\sum_{u \in \mathcal{R}} \{f_{0,u}\psi(u) + f_{1,u}(1 - \psi(u))\} \\ = \sum_{u \in \mathcal{R}} f_{1,u} + \sum_{u \in \mathcal{R}} (f_{0,u} - f_{1,u})\psi(u).\end{aligned}$$

As the first sum on the right side is constant, it is equivalent to minimize

$$C(\psi) = \sum_{u \in \mathcal{R}} (f_{0,u} - f_{1,u})\psi(u). \tag{2}$$

This minimization is trivial if we do not impose the monotonicity of the function.

III. Minimum Cost Flow Approach

If we impose the monotonicity as a constraint for the design of ψ, we arrive at a large scale linear integer programming (IP) problem. This formulation of the filter design problem is closely related to flow models, as first suggested by Gabbouj and Coyle in [23]. For such models, it is possible to drop the integrality constraints and deal with ordinary linear programs (LPs).

In this section, we present the LP formulation in [2] and show that its dual can be seen as a network flow problem that may be efficiently solved by a network Simplex method.

However, as the window size grows, the number of variables and constraints in the LP formulation grows exponentially to a point that the naive use of the network Simplex method may not be feasible anymore. To overcome this difficulty, we present a technique that allow us to split the original problem in smaller subproblems. We also show that it is possible to iteratively improve a candidate filter without loading the complete formulation in main memory.

For the sake of clarity, the conceptual introduction of the algorithm is presented in six sections.

A. Associated LP

In order to formulate the optimization model of the filter design problem, we initially turn to the formulation given in [23], i.e., for each element $u \in \mathcal{R}$, we associate a variable $x_u \in \{0,1\}$ corresponding to the value of u under the mapping of the Boolean operator.

Considering the objective function given in (2) and, setting $c_u = f_{0,u} - f_{1,u}$, the design problem can be formulated as the following integer linear program

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ \text{subject to } & Ax \geq 0_m \\ & -I_{2^d} x \geq -1_{2^d} \\ & x \geq 0_{2^d} \\ & x \text{ is integral} \end{aligned} \tag{3}$$

where A is a matrix corresponding to (monotonicity) inequalities $x_u \leq x_v$ for each pair $u < v \in \mathcal{R}$ and m is the number of rows of A. More formally, the rows of A are indexed by pairs

$(u,v) \in \mathcal{R}^2$ with $u < v$ and the columns of A are indexed by $\mathcal{R}$. We have

$$A_{(u,v),w} = \begin{cases} -1, & \text{if } u = w \\ 1, & \text{if } v = w \\ 0, & \text{otherwise.} \end{cases}$$

The identity matrix and the corresponding -1_{2^d} vector in (3) ensure that $x_u \le 1$ for all $u \in \mathcal{R}$.

Since the constraint matrix

$$\begin{bmatrix} A \\ -I \end{bmatrix}$$

is *totally unimodular* and $(0_m, -1_{2^d})$ is an integral vector, all basic feasible solutions of (3) are integral. This fact allows us to drop the integrality constraint in (3) (see, for instance, [24]).

We can also reduce the size of A by considering transitivity, i.e., we may associate a row corresponding to the pair $u < v$ if and only if there is no $w \in \mathcal{R}$ such that $u < w < v$. Although this is a more compact representation, any attempt of solving problem (3) directly (even when the window dimension is modest) is impractical, despite the efficiency of current IP/LP solvers.

B. Duality

Even though the LP formulation given by (3) is quite interesting, in particular, due to unimodularity, more structure is unveiled when we look at its dual problem. Duality is a very important tool in linear programming. It associates to an optimization problem a *dual* problem that classically can give new insights and interpretations to the original formulation [25]. Moreover, in the case of LP, the solution of the dual problem allow us to find a solution of the original problem.

In this section, we show that the dual of the LP relaxation of (3) corresponds to a minimum cost network flow (MCNF) problem. These problems are very important because they admit a very efficient specialization of the original Simplex method, called the *network Simplex algorithm.*

In order to start the discussion, let us recall the definition of a MCNF problem.

Definition 1: Let $D = (V, A)$ be a *directed graph*. This means that V is a set of *vertices* and the *arcs* in A are ordered pairs $(u,v) \in V \times V$ with $u \neq v$. It will be convenient to denote the arc (u,v) by uv. Let $z \in \mathbb{Z}^V$ be a vector such that z_v is the *demand* of the vertex v. We will call z the *demand vector* of D. A *feasible flow* in D is a vector $x \in \mathbb{Z}^A$ that satisfies the demands, i.e., for every $v \in V$

$$\sum_{u \in V : uv \in A} x_{uv} - \sum_{w \in V : vw \in A} x_{vw} = z_v.$$

Let $y \in \mathbb{Z}^A$ be such that y_a is the cost of passing one unit of flow through the arc a. We say that y is the *cost vector* of D. The total *cost of a flow* x is defined as $\sum_{uv \in A} y_{uv} x_{uv} = \langle y, x \rangle$.

The *minimum cost network flow problem (MCNF)* consists in finding a feasible flow x^* with minimum cost.

We proceed by showing that the dual problem associated to the LP relaxation of (3) can be interpreted as a MCNF problem. This dual problem is given by

$$\begin{aligned} &\min \langle -b, z \rangle \\ &\text{subject to } [\, A^T \; -I_{2^d} \,] z \le c \\ &z \ge 0 \end{aligned} \tag{4}$$

where $b = (0_m, -1_{2^d})$ and m is the number of rows in the matrix A.

After introducing slack variables, (4) becomes

$$\begin{aligned} &\min \langle -b, z \rangle \\ &\text{subject to } [\, A^T \quad -I_{2^d} \quad I_{2^d} \,] \begin{bmatrix} z \\ s \end{bmatrix} = c \\ &z, s \ge 0. \end{aligned} \tag{5}$$

It is easy to see that the componentwise sum of the rows of the constraint matrix in (5) is the vector $w := [0_m \;\; -1_{2^d} \;\; 1_{2^d}]^T$. Hence, if we append $-w$ as a new row in the constraint matrix and extend the right-hand-side accordingly, we obtain

$$\begin{aligned} &\min \langle -b, z \rangle \\ &\text{subject to } \begin{bmatrix} A^T & -I & I \\ 0 & 1_{2^d} & -1_{2^d} \end{bmatrix} \begin{bmatrix} z \\ s \end{bmatrix} = \begin{bmatrix} c \\ -\sum_{u \in \mathcal{R}} c_u \end{bmatrix} \\ &z, s \ge 0. \end{aligned} \tag{6}$$

In (6), the constraint matrix can be seen as the incidence matrix of a directed graph $D = (V, A)$, in which the set of vertices is $V = \mathcal{R} \cup \{r\}$ (the node r corresponds to the appended row, that is, $-w$) and the set of arcs A is such that, for every constraint $x_u \le x_v$ given by the lattice structure, we have an arc $uv \in A$. We also have arcs $sr, rs \in A$ for every $s \in \mathcal{R}$. Since r is connected to every other node of D, we call it the *root* node.

Observe that (6) is a linear programming formulation for the MCNF problem in the network defined by the directed graph D with demand vector $(c, -\sum_{u \in \mathcal{R}} c_u)$. It also has a very particular cost function. All the arcs having r as the destination node have unit cost and all others have zero cost. Hence, the cost vector can be represented by $(\delta_{v,r})_{uv \in A}$, where $\delta_{i,j}$ is the Kronecker delta.

C. Minimum Cost Network Flow Problems

MCNF problems are usually solved using a specialization of the simplex method called the *network Simplex algorithm.* This algorithm moves along *tree solutions* trying to decrease the objective value.[1]

Definition 2: A feasible flow x is called a *tree solution*, or *basic tree*, if there is a spanning tree $T \subseteq A$ such that $x_a = 0$ for every $a \notin T$. An arc will be called *basic* if it is contained in a basic tree.

Every MCNF problem has an optimal solution that is a tree solution. Given an initial tree solution, the network simplex algorithm moves among trees by adding a new arc to the tree and eliminating a different arc of the (unique) cycle created by such

[1]A reference to network flows and combinatorial optimization is [24].

new arc. This is done in such way that the cost never increases and, eventually, decreases. The tree changes have a very low computational cost as they are based on pivot operations (see [24, Ch.4]).

The network Simplex converges in a finite number of iterations to an optimal flow (dual optimal solution). Furthermore, we may easily recover a basic solution of the original filter design problem from such optimal flow. In particular, its integrality as assured by unimodularity.

D. Decomposition Principle

In the last section, we showed that the network Simplex algorithm may be used in the design of monotone filters with optimal MAE. However, the size of the network graph is still exponential in the window size, posing real challenges to its applicability. This fact may be mitigated if we are able to find a way to decompose the original problem in smaller subproblems that are easier to solve. We show next how to use the Boolean lattice structure present in the filter design problem to induce such a decomposition. First, we need some definitions before presenting the main result of this section.

Definition 3: An *ideal* is a set $I \subseteq \mathcal{R}$ such that for all $u \in I$ we have $\{v \in \mathcal{R} \mid v < u\} \subseteq I$.

Definition 4: Given a set X of elements of the Boolean lattice $\mathcal{R}$, we define the *cost of a Boolean operator* ψ relative to X as

$$c(\psi, X) := \sum_{u \in X} c_u \psi(u).$$

If $c(\psi^*, X)$ is minimum among all monotone Boolean operators then ψ^* is an *optimal solution (or operator) relative to* X, and we define the *optimal cost* of the set X as $\mathrm{opt}(X) := c(\psi^*, X)$.

Note that any monotone operator $\psi: X \to \{0,1\}$ can be extended to a monotone operator on the whole lattice $\mathcal{R}$. Our main decomposition principle is stated in the next theorem.

Theorem 5: (Decomposition Principle) Let $S \subseteq \mathcal{R}$ and $I \subseteq S$ be ideals. If $\mathrm{opt}(I) = 0$, then there is an optimal operator ψ relative to S such that $\psi(I) = \{0\}$. Furthermore, if ψ is an optimal solution relative to S, then

$$J = \ker(\psi) = \{u \in \mathcal{R} \mid \psi(u) = 0\}$$

is an ideal satisfying $\mathrm{opt}(J) = 0$.

Proof: Let ψ^* be an optimal solution relative to S and $I \subseteq S$ be an ideal such that $\mathrm{opt}(I) = 0$. Define the operator ψ' as

$$\psi'(u) := \begin{cases} \psi^*(u), & \text{if } u \in S \setminus I \\ 0, & \text{if } u \in I. \end{cases}$$

Note that the above operator is monotone. Indeed, since I is an ideal, whenever $u \in I$ and $v \in S \setminus I$, we have $v \not\leq u$ (hence, either u and v are not comparable or $v \geq u$). Since $\psi'(u) = 0 \leq \psi'(v)$, the monotonicity constraint is fulfilled. If both u and v belong to I the conclusion is trivial, and if both u and v belong to $S \setminus I$ then, by our assumption that ψ is monotone, the conclusion is again trivial.

Therefore, the cost $c(\psi', S)$ must be the same as $c(\psi^*, S)$; otherwise, $c(\psi^*, I) < 0 = \mathrm{opt}(I)$, a contradiction. This proves the first assertion of the theorem.

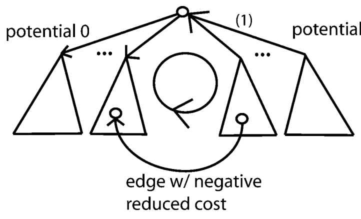

Fig. 1. Simplified diagram of a tree solution. Notice that, in order to reduce the cost of the solution, one must push flow in the direction of the cycle depicted above.

In order to prove the second assertion, notice that, since ψ is monotone, $J = \ker(\psi)$ must be an ideal. Now suppose that $\mathrm{opt}(J) < 0$. There must be a nontrivial solution ψ^* that is optimal relative to J. Define

$$\psi'(u) := \begin{cases} 1, & \text{if } u \in S \setminus J \\ \psi^*(u), & \text{if } u \in J. \end{cases}$$

By our assumptions, it is clear that ψ' is a monotone operator and its cost is given by

$$\begin{aligned} c(\psi', S) &= \sum_{u \in S \setminus J} c_u + \mathrm{opt}(J) \\ &= c(\psi, S) + \mathrm{opt}(J) < c(\psi, S) = \mathrm{opt}(S) \end{aligned}$$

which is a contradiction. ■

Due to the symmetry of the Boolean lattice, one can easily see that, with a symmetric definition of ideals, we can obtain a symmetric version of Theorem 5. A set $I \subseteq \mathcal{R}$ is called a *sup-ideal* if, for every $u \in I$, we have $\{v \in \mathcal{R} \mid v > u\} \subseteq I$.

Corollary 6: Let $S \subseteq \mathcal{R}$ and $I \subseteq S$ be sup-ideals. If $\mathrm{opt}(I) = \sum_{u \in I} c_u$, then there is an operator ψ that is an optimal solution relative to S such that $\psi(I) = \{1\}$. Furthermore, if ψ is an optimal solution relative to S, then

$$J = \{u \in \mathcal{R} \mid \psi(u) = 1\}$$

is a sup-ideal satisfying $\mathrm{opt}(J) = \sum_{u \in J} c_u$.

E. Decomposition and the Network Simplex

The decomposition principle tells us that, after solving the subproblem defined by some ideal, every element mapped to 0 in the optimal solution of this subproblem can be fixed to 0 in a global optimal solution. We will use this idea in an algorithm that iteratively solves subproblems using the network Simplex as its basic tool.

In order to apply the decomposition in the development of an algorithm, there must be a simple way to choose ideals from the Boolean lattice. For instance, one can define a total order on the Boolean lattice that respects the canonical partial order. Given such total order, all elements of $\mathcal{R}$ can be linearly ordered so that any initial (final) segment in this order forms an ideal (sup-ideal).

Here, we propose a total order based on the support of each pattern. To break the ties, we use the associated costs and the lexicographical order. Formally, let $w(u) := |\mathrm{supp}(u)|$, and let $\prec$ denote the lexicographic order. For any Boolean lattice $\mathcal{R}$

TABLE I
COMPUTATIONAL TIME FOR WINDOW SIZE 25. THE COMPUTER IS BASED ON A 2.8-GHz INTEL XEON AND HAS 4 GB OF MEMORY

Test	Description of noise and training data	`stackfd`	`train` (100 iter.)
Binoise	Binary, simple shapes. 15% salt and 15% pepper noise. 10 256x256 training images.	156.32s	335.94s
BooleanA	Binary Boolean model (mean=11 and variance=4 pixels length square grains with 0.2% of occurrence). 2% additive and 1% subtractive noise (random subsets of size between 2 and 5 within the 3×3 square). 10 512x512 training images.	207.93s	333.74s
BooleanB	Similar to BooleanA. 3% additive and 3% subtractive noise. 10 512x512 training images.	194.07s	340.88s
EinsteinA	Gray-level, Einstein photo. 5% additive and 5% subtractive impulse noise (both with amplitude 200, and maximum and minimum saturated at 255 and 0, respectively) plus horizontal line segments of intensity 255 with probability of occurrence 0.35%, with length following a normal distribution with mean 5 and variance 49 pixels). 1 512x512 training image.	416.94s	356.16s
EinsteinB	Similar to EinsteinA. 1.5% additive and 1.5% subtractive impulse noise plus 0.1% of horizontal line segments. 1 512x512 image.	234.93s	353.05s
EinsteinC	Gray-level, Einstein photo. 3% additive and 3% subtractive impulse noise. 1 512x512 training image.	254.04s	357.49s
EinsteinD	Gray-level, Einstein photo. 0.5% additive and 0.5% subtractive noise similar in shape to the noise in BooleanA, with amplitude 200. 1 512x512 training image.	242.80s	361.64s
EinsteinE	Gray-level, Einstein photo. 6% additive and 6% subtractive impulse noise. 1 512x512 training image.	330.29s	359.55s

and any cost function $c: \mathcal{R} \to \mathbb{Z}$, we may define a total order, denoted $<$, by the following.

1) If $w(u) < w(v)$, then $u < v$.
2) If $w(u) = w(v)$ and $c_u < c_v$, then $u < v$.
3) If $w(u) = w(v)$, $c_u = c_v$ and $u \prec v$, then $u < v$.

Notice that this order respects the canonical partial order $\leq_{\mathcal{R}}$.

Given an ideal I, we can solve the problem restricted to I and obtain an optimal operator ψ_I. The method for finding such restricted operators is described in the next section. The set $J = \ker(\psi_I)$ is a subset of I. By the decomposition principle, there exists an optimal operator for the whole lattice that maps every element of J to 0. Also, for every ideal $K \supseteq J$, the set $K \setminus J$ is an ideal of $\mathcal{R}' = \mathcal{R} \setminus J$. The decomposition principle also states that, if we have an operator ψ_I that is optimal restricted to the ideal I, and an operator $\psi_{\mathcal{R}'}$ that is optimal restricted to $\mathcal{R}'$, we can define

$$\psi^*(u) := \begin{cases} \psi_{\mathcal{R}'}(u), & \text{if } u \in \mathcal{R}' \\ 0, & \text{otherwise} \end{cases}$$

as a global optimal operator.

Hence, the complete filter design problem may be solved iteratively by 1) starting with an ideal I; 2) solving the associated MCNF problem; 3) fixing the values of those patterns in the kernel of the optimal operator restricted to I; and 4) increasing the ideal with new nodes following the total order given above. The larger problem is not necessarily more difficult than the previous one, as many patterns may be already fixed.

This strategy would be particularly interesting if we knew how to use the solution for a smaller ideal to find an initial feasible tree needed by the network Simplex method when solving some larger problem. To achieve this, let us show a feasible tree solution T_0 for the whole network.

Let v be a node corresponding to some pattern p. If p has preference for output 1 (that is, $c_p = f_{0,p} - f_{1,p} < 0$, in the notation of Section III-A), then vr is selected as basic arc, otherwise (if $c_p \geq 0$), rv is selected as basic (recall that r is the root node of the network, defined in Section III-B). This defines a tree solution where every node is connected directly to r.

Given a partial network, a corresponding tree solution and a new node v to be added, we extend the tree solution selecting either rv or vr (the one belonging to T_0) as basic.

F. Column Generation

Finally, we show how to solve the problem restricted to a given ideal. A naive approach would be to create an instance of the respective MCNF problem and use the network Simplex algorithm to solve it. Even though this is very effective for small window sizes, it becomes more and more difficult as the window size grows. In particular, the number of arcs in the network associated to each LP becomes huge. To alleviate this problem, we propose a classical technique that allows us to keep in main memory only the current tree solution and generate a new arc only when it enters the next tree. Such techniques that keep in memory only a minimal number of variables and generate a new variable when needed are usually called *column generation* [25].

In order to do this in the optimal MAE filter design problem, we start recalling an important definition:

Definition 7: Let $D = (V, A)$ be a directed graph. Let T be a rooted tree of D with root r. Suppose x is a tree solution having T as its support and $y \in \mathbb{Z}^V$ be such that, for every arc uv of T, we have $y_v = y_u + c_{uv}$. We call y_v the *potential* of node v. It will be convenient to set $y_r = 0$.

Recall the special structure of problem (6). Since $c_{uv} = \delta_{r,v}$ and y_r is set to 0, the only possible potentials for nodes in the graph are in $\{-1, 0\}$. In order to calculate the potential y_v of some node v, we may find the (unique) undirected path $P(v, r)$ connecting v to the root r. Let w be the last node before r in $P(v, r)$. We set $y_v = -1$, if the directed arc wr is basic or $y_v = 0$, if the directed arc rw is basic (clearly, one of these arcs must be part of the basic tree).

The only interesting arcs to enter the current tree solution and generate a new tree with smaller objective value are the arcs with negative reduced costs. As noted in Section III-C, the arc costs of our graph are always in $\{0, 1\}$ and, as we saw above, the node potentials are always in $\{-1, 0\}$. This means that an arc uv has negative reduced cost if and only if $y_u = -1, y_v = 0$ and $v \neq r$ (see Fig. 1). This simple characterization allows us to identify the candidate arcs to enter the tree solution and generate them only when needed. Given an entering arc, the update of the tree solution can be easily performed by a classical pivot operation.

G. Putting Everything Together

Let us now gather the ideas presented in this section in a full outline of the proposed algorithm. Let k be an integer used to determine how large the subproblems might be. It is also useful to define predicates `head`, and `next` for linked lists and its elements (with obvious meaning). The sentinels of the linked list are set to `nil`. Our algorithm is composed of the following steps.

1) Sort the patterns according to the total order $<$ into a linked list l.
2) Create a network (graph) containing only the root node r. Set $p \leftarrow \texttt{head}(l)$.
3) While the current network has $<k$ nodes and $p \neq \texttt{nil}$
   a) add a node v to the network corresponding to p;
   b) add arcs rv and vr to the network;
   c) extend the tree solution by adding either the arc vr or rv as described in Section III-E;
   d) let $p \leftarrow \texttt{next}(p)$.
4) Run the network Simplex using column (arc) generation for the current network.
5) If l is not empty then
   a) for every node of the network with potential 0 we remove the corresponding pattern from the list l and fix its final value as 0 (decomposition principle);
   b) delete the current network and goto 2.
6) If l is empty, we fix the final value of all the remaining patterns.

The actual implementation can be obtained from the website http://www.vision.ime.usp.br/nonlinear/stackfd. It is more involved than the algorithm outlined above. In particular, it exploits the symmetry of the Boolean lattice, applying the decomposition principle to both ends of the linked list.

In our tests, we have limited the number of nodes to $k = 2\binom{d}{\lfloor d/2 \rfloor}$. This number corresponds to an upper bound on the number of elements in the two middle levels of the Boolean lattice. This value proved to be a good compromise between speed and memory consumption.

TABLE II
MAE, WINDOW SIZE 25

Test	Training		Testing	
	`stackfd`	`train`	`stackfd`	`train`
Binoise	0.0008	0.0071	0.0027	0.0296
BooleanA	0.0034	0.0097	0.0046	0.0123
BooleanB	0.0078	0.0151	0.0101	0.0210
EinsteinA	2.5962	3.0321	3.1428	3.3453
EinsteinB	1.2831	1.5148	1.3402	1.5486
EinsteinC	1.6126	1.8556	1.7935	1.9418
EinsteinD	1.8670	2.1670	2.1382	2.3895
EinsteinE	2.3774	2.8053	2.8489	3.0267

IV. Computational Results

We have implemented the ideas described above and tested them using binary and gray-level images. Due to space limitation, we can only present part of these tests, in particular, just a few images. The tests presented here were carefully chosen to reflect the overall results in all tests. Many more details can be seen on the website http://www.vision.ime.usp.br/nonlinear/stackfd. We will refer to our implementation as `stackfd`.

Our main objective is to estimate the computational resources needed by `stackfd`. We will also compare our results to a borrowed implementation [26] of the `train` heuristic [27]. This suboptimal algorithm is very fast and can deal with large window sizes. Other heuristics, like the ones presented in [14] and [22] are not included, as their computational time is much larger than the ones presented here.

With the default limitation on memory usage (see Section III-G), we were able to train filters with window size 25. Our program used 3 GB of memory for that dimension. In

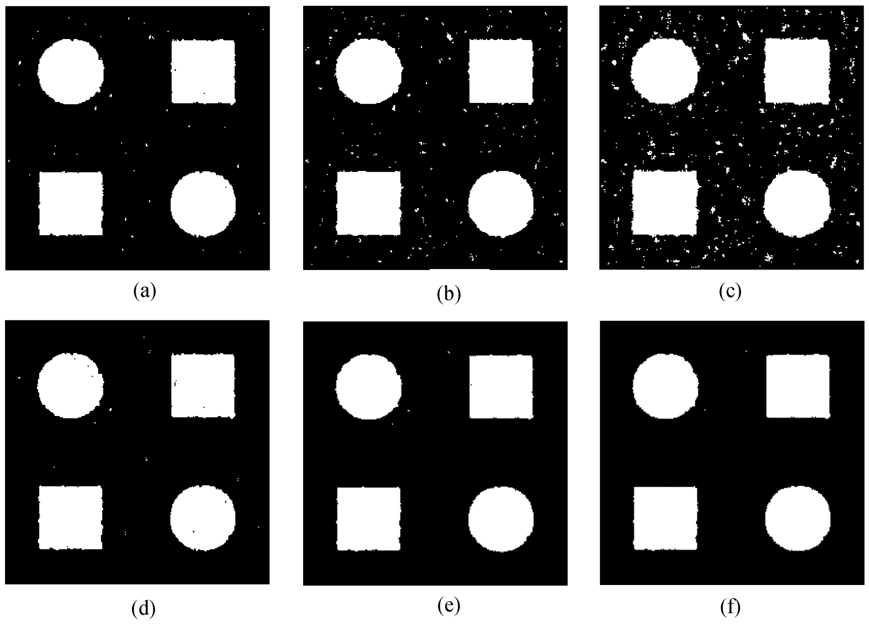

Fig. 2. Quality evolution with increasing window size. The testing image is equal to the training image, but with an independent realization of the noise. In the top row are the results of `train`: (a) window size 15 (MAE 0.0046); (b) window size 21 (MAE 0.0109); (c) window size 25 (MAE 0.0296). In the bottom row are the results of `stackfd`: (d) window size 15 (MAE 0.0041); (e) window size 21 (MAE 0.0028); (f) window size 25 (MAE 0.0027).

this respect, the `train` heuristic performs much better, using around 256 MB.

On the other hand, regarding training time, our code outperformed `train` in most of the tests. Table I presents the training time for `stackfd` and `train` for different tests. The difference can be up to twice as fast in simple data, like the binary images. In gray-level images, the difference decreases, with `train` taking around 40% more time in four tests and about 9% in EinstenE. The `train` implementation is faster only in EinsteinA, where it takes 85% of the time used by `stackfd`. Remember that `stackfd` computes a filter with minimum MAE, while `train` is suboptimal.

A. MAE and Visual Quality

Given that `stackfd` is usually faster than `train`, another important variable is the difference between the MAE of an optimal filter and a suboptimal one.

For small window size, like 15, the `train` heuristic can actually find an almost optimal filter. However, as window size grows, the gap in the MAE of the filter computed by `train` and an optimal filter computed by `stackfd` increases, specially in binary images. Table II summarizes the MAE obtained with window size 25 both on training and on independent testing images. The testing images are the same used in the respective training phase, only with an independent realization of the noise.

We can see that, for binary images, `stackfd` performs much better than `train`. The MAE in both testing and training images is at least half of the MAE obtained by the `train` filter. The reason for this is that the suboptimality of the `train` method is precluding it to find a better filter as window size grows. Actually, in the first test, Binoise, the quality of filtered image clearly decreases with the window size (see Fig. 2). On the other hand, the `stackfd` method can better explore large windows and, thus, the quality of the image improves with window size up to 25 pixels. This may indicate that the number of iterations for `train` is insufficient. Increasing this number would allow it to find better filters. However, the computational time would increase proportionally and the performance gap with `stackfd` would increase.

For gray-level images, the differences between the MAE of the filters computed by `stackfd` and `train` are relatively smaller, but again `stackfd` consistently outperforms `train` on test data. While on binary images the differences are easily perceived visually, on gray-level images the visual differences are more subtle. A careful visual examination shows that images filtered by `stackfd` tend to keep more impulse noise in the filtered image but look sharper than the ones obtained by a filter computed by `train`, thus better preserving edge details (see Figs. 3 and 4). Once again, we could increase the number of iterations in `train` in order to improve the sharpness of the filtered images. In this case, however, the resulting image would contain more impulse noise as in the image obtained by `stackfd`.

We conjecture that the excess of noise in the `stackfd` image is a consequence of the excess of freedom given to the optimal method. For large window size, most of the possible patterns are not present in the training data. For example, for window size 25, in the gray-level images we have tested, only about 5% of the patterns were present in the training data. We believe that this sparseness gives too much freedom to the optimal method to choose the values of 95% of the patterns in a way that do not affect the MAE value in the training process.

To test this conjecture we decided to preprocess the training data. If a pattern is not present in it, we fill the pattern cost using the mean cost of the neighbor patterns that were actually observed in the original data. This process is repeated until there is no pattern which contributes arbitrarily with zero in the objective function.

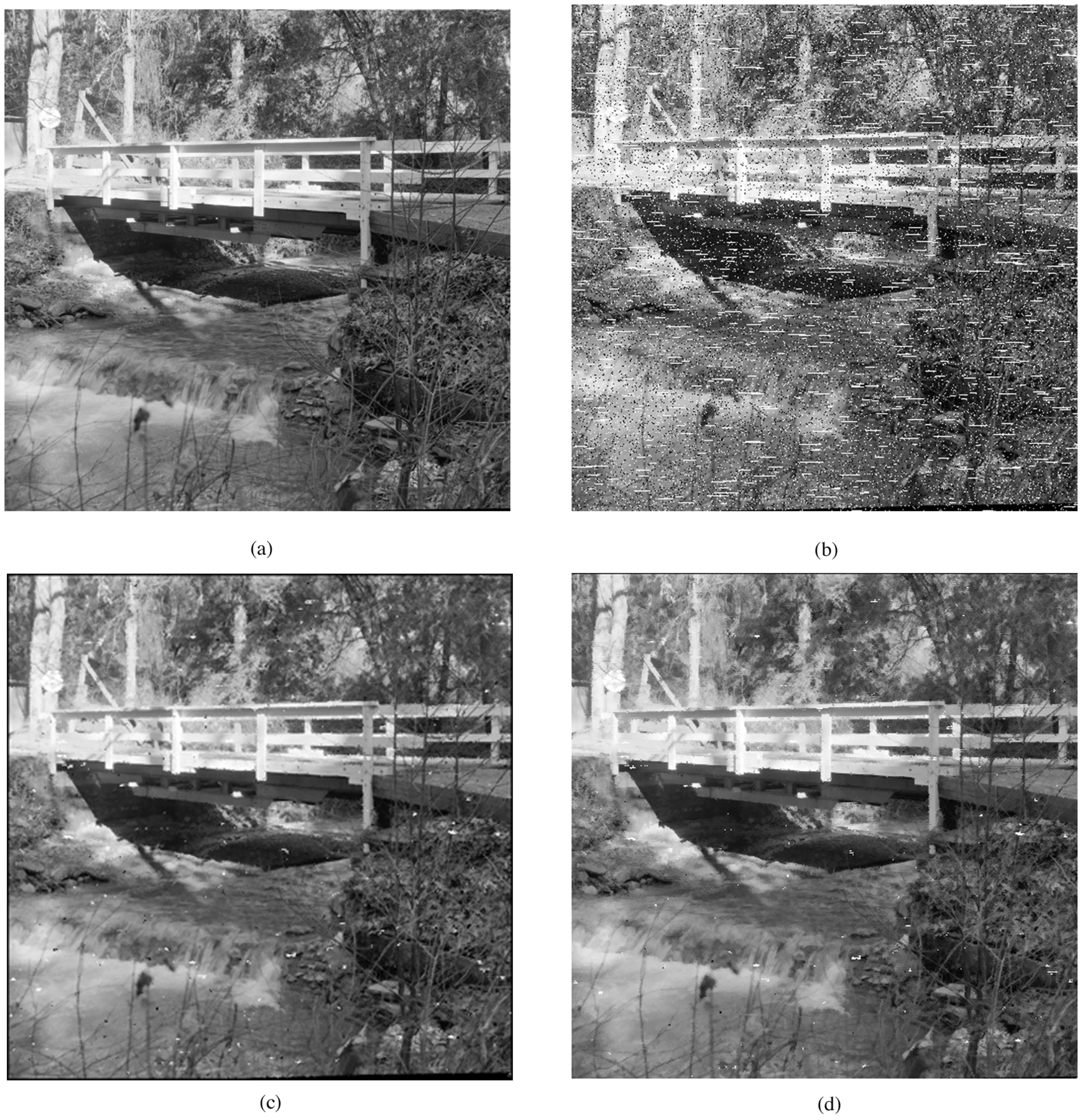

Fig. 3. Visual comparison of gray-level images filtered by `stackfd` and `train`: (a) ideal image, (b) noisy image, (c) `stackfd` filtered (MAE 4.334580), and (d) `train` filtered (MAE 4.716431). The training images used to obtain the filters are those contained in the EinsteinA test, described in Table I. The test image is a completely different picture, "stream and bridge," with an independent realization of the same type of noise. Note that the `stackfd` image in (c) contains more noise; however, it is sharper. The `train` image in (d) also seems to increase the average gray-level intensity in some areas of the image, producing a slightly brighter image than the ideal one.

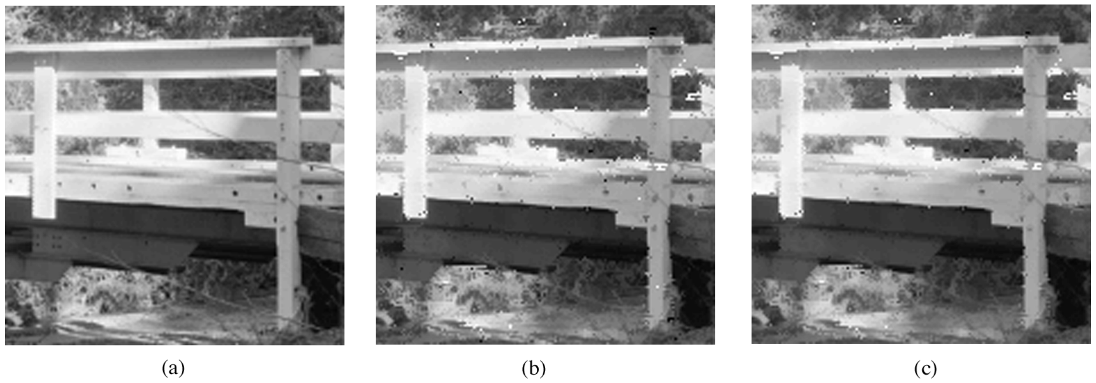

Fig. 4. Detail of Fig. 3: (a) ideal image, (b) `stackfd` filtered, and (c) `train` filtered. The image computed using `stackfd` is sharper. Look at the three nails in the bottom of the bridge to the left. The `stackfd` image in (b) still leaves traces of the tree nails, while `train` in (c) almost completely erases them.

The visual results using the optimal filter based on this heuristic filling of missing costs are encouraging. They usually keep the good properties of the original `stackfd` in binary images while improving the noise reduction in gray-level images without affecting sharpness too much (see Fig. 5). This opens up the possibility of trying more sophisticated methods

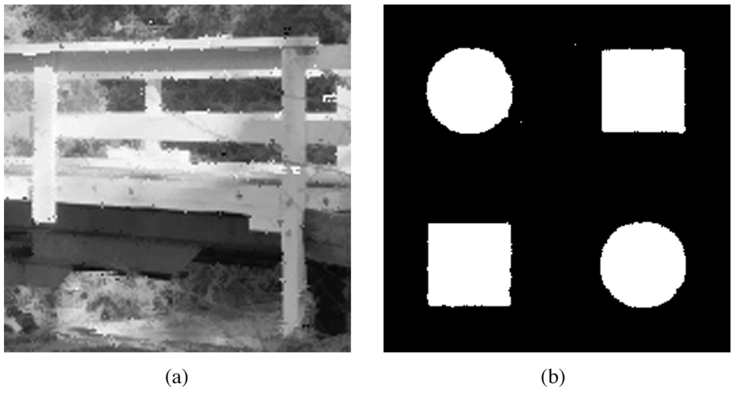

Fig. 5. `stackfd` with heuristic filling. The heuristic decreases the amount of noise in the resulting image, without decrease in sharpness as in (a) and in the performance in binary images as in (b).

for the filling, for example, methods based on multiresolution techniques. This is subject of ongoing research.

V. Concluding Remarks

In this paper, we have presented a new optimal algorithm to find a minimum MAE stack filter. In our tests, the proposed method is usually faster than the fastest heuristic to date. These results show that the combination of the network Simplex method with decomposition and column generation techniques lead to a very efficient algorithm. On the other hand, our solution is very memory demanding.

Our computational tests have also unveiled some problems associated to the use of heuristics, where the quality of the computed solution is not easily comparable to the optimum. In some tests on binary images, the heuristic presented a poor performance with large window size when compared to our optimal method. In gray-level images, the optimal filter seems to better preserve original characteristics of the image, like sharpness, at the expense of leaving more visual noise in the filtered image.

We have also proposed a naive way to fill the costs of the patterns that are not present in the training data. That helped to reduce noise in the filtered images while preserving the strengths of our solution. We believe that the quest for a better strategy to estimate the cost of the missing patterns, based on multiresolution ideas [28], is very promising and is the subject of ongoing research.

Acknowledgment

The authors would like to thank the anonymous referees for many useful remarks.

References

[1] P. D. Wendt, E. J. Coyle, and N. C. Gallagher Jr., "Stack filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 4, pp. 898–911, Aug. 1986.

[2] E. J. Coyle and J.-H. Lin, "Stack filters and the mean absolute error criterion," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, no. 8, pp. 1244–1254, Aug. 1988.

[3] J.-H. Lin, T. M. Sellke, and E. J. Coyle, "Adaptive stack filtering under the mean absolute error criterion," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 6, pp. 938–954, Jun. 1990.

[4] P.-T. Yu and E. J. Coyle, "Convergence behavior and n-roots of stack filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 9, pp. 1529–1544, Sep. 1990.

[5] P.-T. Yu, "Some representation properties of stack filters," *IEEE Trans. Signal Process.*, vol. 40, no. 9, pp. 2261–2266, Sep. 1992.

[6] P.-T. Yu and E. J. Coyle, "The Classification and associative memory capability of stack filters," *IEEE Trans. Signal Process.*, vol. 40, no. 10, pp. 2483–2497, Oct. 1992.

[7] L. Yin, J. T. Astola, and Y. A. Neuvo, "Adaptive stack filtering with application to image processing," *IEEE Trans. Signal Process.*, vol. 41, no. 1, pp. 162–184, Jan. 1993.

[8] J. H. Lin and Y.-T. Kim, "Fast algorithms for iraining stack filters," *IEEE Trans. Signal Process.*, vol. 42, no. 4, pp. 772–781, Apr. 1994.

[9] L. Yin, "Stack filter design: A structural approach," *IEEE Trans. Signal Process.*, vol. 43, no. 4, pp. 831–840, Apr. 1995.

[10] I. Tăbuş, D. Petrescu, and M. Gabbouj, "A training framework for stack and Boolean filtering—Fast optimal design procedures and robustness case study," *IEEE Trans. Image Process.*, vol. 5, no. 6, pp. 809–826, Jun. 1996.

[11] C.-C. Han and K.-C. Fan, "Finding of optimal stack filter by graphic searching methods," *IEEE Trans. Signal Process.*, vol. 45, no. 7, pp. 1857–1862, Jul. 1097.

[12] J. Yoo, K. L. Fong, J.-J. Huang, E. J. Coyle, and G. B. Adams III, "A fast algorithm for designing stack filters," *IEEE Trans. Image Process.*, vol. 8, no. 8, pp. 1014–1028, Aug. 1999.

[13] W.-L. Lee, K.-C. Fan, and Z.-M. Chen, "Design of optimal stack filters under the MAE criterion," *IEEE Trans. Signal Process.*, vol. 47, no. 12, pp. 3345–3355, Dec. 1999.

[14] I. Tăbuş and B. Dumitrescu, "A new fast method for training stack filters," presented at the IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, Antalya, Turkey, Jun. 1999.

[15] M. K. Prasad, "Stack filter design using selection probabilities," *IEEE Trans. Signal Process.*, vol. 53, no. 3, pp. 1025–1037, Mar. 2005.

[16] N. S. T. Hirata and J. Barrera, "A unifying view for stack filter design based on graph search methods," *Pattern Recognit.*, vol. 38, no. 11, pp. 2088–2098, 2005.

[17] P. Maragos and R. W. Schafer, "Morphological filters: Part II: Their relations to median, order statistic, and stack-filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, no. 8, pp. 1170–1184, Aug. 1987.

[18] J. P. Fitch, E. J. Coyle, and N. C. Gallagher Jr., "Median filtering by threshold decomposition," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-32, no. 6, pp. 1183–1188, Dec. 1984.

[19] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: A tutorial," *IEEE Trans. Circuits Syst II, Analog Digit. Signal Process.*, vol. 43, no. 3, pp. 157–192, Mar. 1996.

[20] R. P. Loce, "Morphological filter mean-absolute-error representation theorems and their application to optimal morphological filter design," Ph.D. dissertation, Center Image Process., Rochester Inst. Technol., Rochester, NY, 1993.

[21] A. V. Mathew, E. R. Dougherty, and V. Swarnakar, "Efficient derivation of the optimal mean-square binary morphologicalfilter from the conditional expectation via a switching algorithm for discrete power-set lattice," *Circuits, Syst., Signal Process.*, vol. 12, no. 3, pp. 409–430, 1993.

[22] N. S. T. Hirata, E. R. Dougherty, and J. Barrera, "A switching algorithm for design of optimal increasing binary filter sover large windows," *Pattern Recognit.*, vol. 33, no. 6, pp. 1059–1081, Jun. 2000.

[23] E. Coyle and M. Gabbouj, “On the LP which finds a MMAE stack filter,” *IEEE Trans. Signal Process.*, vol. 39, no. 11, pp. 2419–2424, Nov. 1991.

[24] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorid Optimization*. New York: Wiley, 1998.

[25] A. M. Geoffrin, Ed., *Perspectives on Optimization*. Reading, MA: Addison Wesley, 1972.

[26] B. Dumitrescu and I. Tăbuş, “Optimal stack and boolean filter design,” [Online]. Available: http://www.cs.tut.fi/~tabus/course/Stack-design.html Nov. 1998, [Online]. Avaliable

[27] K. L. Fong, G. B-Adams III, E. J. Coyle, and J. Yoo, “Synthesis of a parallel optimal stack filter training algorithm,” presented at the IEEE Workshop on Nonlinear Signal and Image Processing, 1997.

[28] E. R. Dougherty, J. Barrera, G. Mozelle, S. Kim, and M. Brun, “Multiresolution analysis for optimal binary filters,” *Math. Imag. Vis.*, no. 14, pp. 53–72, 2001.

Domingos Dellamonica, Jr., is pursuing the M.S. degree at the Computer Science Department, University of São Paulo, São Paulo, Brazil.

His interests are in optimization and combinatorics.

Paulo J. S. Silva is an Assistant Professor in the Computer Science Department, University of São Paulo, São Paulo, Brazil. His current research interests include linear and nonlinear optimization and their applications.

Carlos Humes, Jr., received the degree in electrical engineering and the M.Eng. degree in systems from the Escola Politécnica, Universidade de São Paulo (USP), São Paulo, Brazil, and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley.

Presently, he is a Full Professor in the Computer Science Department of the Institute of Mathematics and Statistics, USP, with interests in optimization and in decentralized control.

Nina S. T. Hirata is an Assistant Professor in the Computer Science Department of the Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil.

Her current research interests include machine learning, multiple classifier systems, and nonlinear image processing.

Junior Barrera received the Ph.D. degree in automatic control and systems from the Department of Electrical Engineering, University of São Paulo (USP), São Paulo, Brazil, in 1992.

From 1985 to 1992, he was with the Brazilian Space Research Institute. In 1992, he joined the Department of Computer Science of the Institute of Mathematics and Statistics, USP, where he is currently a Full Professor. His career was focused on the study of lattice operator representation and design, producing several theoretical and applied contributions. Recently, he also worked on the representation and design of lattice dynamical systems. For many years, he has centered his interests on image processing applications, but, in the recent years, he has also worked on bioinformatics and computational biology applications. His performance in this field lead him to some administrative positions. He is the President of the USP Center for Bioinformatics.

Dr. Barrera is currently Vice President of the Brazilian Society for Bioinformatics and Computational Biology.