A NEW FAST METHOD FOR TRAINING STACK FILTERS

Ioan Tăbuş and Bogdan Dumitrescu

Signal Processing Laboratory, Tampere University of Technology
P.O. Box 553, SF-33101 Tampere, Finland
e-mail: tabus@cs.tut.fi

ABSTRACT

In this paper we introduce a new procedure for stack filter design with respect to MAE optimality criterion. While the procedure is only suboptimal, it differentiates from the existing methods by being extremely fast on sequential computers. Up to window length 19 this is by far the most efficient method, being an order of magnitude faster, while providing a MAE with only 0.1% greater than other existing methods.

Experiments performed on a normal workstation (Sun Ultra 1) shows that for a 512×512 image the execution time is less than a couple of seconds, for a 17 pixel window.

1. INTRODUCTION

The optimal or adaptive stack filter design with respect to MAE criterion was extensively studied, well motivated methods being presented in [1], [2], [3], [4], [7], [8]. While the principles are well established, the associated procedures encounter several practical difficulties, one of the most important being the design time, and the second the course of dimensionality, which makes the design next to impossible when using common computers, for templates larger than 22 pixels. We discuss here two recent methods for stack filter design, and propose a third one which proves much faster than the others, for a certain range of template sizes. We also find a partial remedy for the dimensionality issue by addressing another important issue, that of training set size, in connection with the size of the window. We show that extending the size of the training set, the design becomes not only more relevant for using the filter on a similar data set, but also makes the design stage better conditioned, and as a result the design time decreases considerably.

2. OPTIMAL STACK FILTER DESIGN USING TRAINING DATA

The existing methods referred here are those in [2], [7] which provide the most efficient results reported up-to-date. Both procedures start from the training set containing a clean signal and a corrupted version of it. The first step is to extract from the training set the sufficient statistics needed in the cost computation. The cost is the average of absolute erors between the clean signal and the filtered signal, denoted in the following MAE. The sufficient statistics are referred to as cost coefficients and denoted $c(\underline{v})$ where $\underline{v} \in \{0,1\}^N$. The cost coefficients can be computed in a very efficient way as presented in [7]. Latter we will present an even more efficient way, using running sorting of the values of the pixels in the template. For a stack filter corresponding to a positive Boolean function f^+, $MAE(f^+)$ can be evaluated as

$$MAE(f^+) = C_0 + \sum_{\underline{v} \in \{0,1\}^N} f^+(\underline{v}) c(\underline{v}). \qquad (1)$$

After the cost coefficients are computed, the procedures in [2], [7] select different approaches for finding the positive Boolean function f^+ minimizing (1).

In [2], a decision vector D^{opt} is initialized with the vector of cost coefficients $D^{opt}(\underline{v}) = -c(\underline{v})$. Then an iterative process start. Using a given

scanning order of the entries, those entries violating the stacking relationship are modified, such as the stacking property is restored. To continue the process, the cost coefficients are subtracted from $D^{opt}(\underline{v})$. If no violation of stacking constraint has occured, the iterations stop, and the positive Boolean function is obtained comparing the entries of D^{opt} to zero, and deciding $f^+(\underline{v}) = 0$ if $D^{opt}(\underline{v}) < 0$ and $f^+(\underline{v}) = 0$ otherwise. This iterative algorithm is very well suited to be performed in parallel, as shown in [2]. In sequential implementation the method is expensive due to the necessity to add cost coefficients even outside the undecided set.

In [7] the cost vector is used to find a Boolean function, and then two sets of binary vectors I_1, decided to 1, and I_0 decided to 0 are constructed. The undecided binary vectors are further processed, using an iterative procedure, denoted there Step III.5. If the template is not too large (say, less than 13 pixel wide), the exact linear programming (LP) solution can be computed. The time required by the training of the stack filter is small compared to the computation of coefficients. However for larger windows, LP cannot be computed, and the output of the algorithm is therefore the output of Step III.5. We propose here to replace the iterative procedure Step III.5 with a more efficient one, in view of obtaining better results on the range of windows 13-19 pixel wide.

In the following we are using the notations introduced in [7], to which the reader will refer for more details on various ways of stating the training problem.

2.1. The new algorithm

We observe that in Step III.5 of FASTAF algorithm [7], which is by far the dominant cost in the procedure, two actions are repeteadly performed: first set to 1 the group of vectors stacking under $\underline{v}$ having overall negative cost, and second, set to 0 the vectors stacking over $\underline{v}$ if their overall cost is positive. The improvements in the cost obtained by this heuristic are significant, but the time to perform iteratively these two steps over the undecided set, I_u, is often very large.

We propose here to use a different approach, to set to 1 groups of vectors (from the undecided set I_u) stacking under $\underline{v}$ having overall negative cost and immediately to delete them from I_u. But we have to decide to 1 only the "safe" group of vectors with overall negative cost, i.e. to gradually grow groups of units starting from the bottom of the undecided set. This will render unnecessary, in the majority of cases, the second iteration from FASTAF III.5, of setting to 0 some groups of vectors, and therefore we may declare decided all the vectors to be set to 1. The undecided set will go diminish along the iterations, making the growing of units in the undecided set extremely efficient. The "safe" groups of pixels to be set to 1 will be determined using the heuristics explained below.

In order to have the undecided set organized by levels in the Hasse diagram, we first determine the sets of undecided binary vectors with the same Hamming weight, $w_H(x) = k$, as $I_{u_k} = \{x \in I_u | w_H(x) = k\}$ and denote k_{min}, k_{max} the minimum respectively maximum k for which $I_{u_k} \neq \emptyset$.

The heuristics of the new algorithm are the following:

1. Include in the set decided to 1, I_1, a "safe" group of vectors stacking under $\underline{v}$, including $\underline{v}$, with overall negative cost. This is equivalent to adding one minterm at a time to the disjunctive-conjunctive form of the boolean function, but what is essential, this minterm has to correspond to the smallest group of vectors which are worth adding.

   Algorithmic description: Start from the next to lowest level, $k_{max}-1$, of the undecided set in the Hasse diagram. For a vector $\underline{v}$ in the set $I_{u_{k_{max}-1}}$ with a negative cost, compute the cost of putting to 1 all vectors $\underline{w} \in I_u$ stacking under the vector $\underline{v}$. If this cost is negative, then put to 1 all $\underline{w}$'s stacking under the vector $\underline{v}$ and delete them from the undecided set I_u. Continue with nodes in the upper level in Hasse diagram until the upper level k_{min} is processed as well.

2. Include in the set decided to 1, I_1, a "safe" group of vectors stacking under $\underline{v}$, excluding

$\underline{v}$, with overall negative cost. This is equivalent to adding more minterms at the same time to the disjunctive-conjunctive form of the boolean function, and again we will select the groups in increasing order of their cardinality.

Algorithmic description: Start from the second next to lowest level, $k_{max}-2$, of the undecided set in the Hasse diagram. For a vector $\underline{v}$ in the set $I_{u_{k_{max}-2}}$ (no matter the sign of its cost), compute the cost of putting to 1 groups of sons of $\underline{v}$. Select the largest group of sons having negative overall cost. Then put to 1 all $\underline{w}$'s stacking under the selected sons of the vector $\underline{v}$ and delete them from the undecided set I_u. Continue with nodes in the upper level in Hasse diagram until the upper level k_{min} is processed as well.

In Figure 1 we list the new algorithm, using the notations from [7]. A single new notation is needed: $W_{down}(Sons)$ is the set of all vectors stacking under the vectors in the set $Sons$.

The fast evaluation of the cost in $Cost_{\text{diff}}$ is realized by a recursive procedure which computes recursively in the levels of Hasse diagram the cost of descendents of $\underline{v}$, and marks the descendents which have already contributed to $Cost_{\text{diff}}$ (note that some descendens of $\underline{v}$ are common descendents of sons of $\underline{v}$, since Hasse diagram is not a tree).

3. EXPERIMENTAL RESULTS

3.1. Fast computation of cost coefficients

The pixels in a sliding window may be put into two categories. Some of them, say n_o, are inherited from the previous window; the others, a number of $n_n = N - n_o$ of them, appear for the first time in a window. In order to efficiently update the costs, pixel values must be ordered; an algorithm which sorts all the N pixels is obviously not appropriate since the n_o values from the previous window are already sorted. The problem of sorting values in a sliding window is known as running ordering and Pitas [6] gave an efficient solution by insertion: each of the new n_n values is inserted in the ordered list containing initially n_o values; this algorithm performs a small number of comparisons, about $O(n_n \log N)$, but is rather costly in memory shifts ($O(N^2 - n_o^2)$). An other possibility is to sort separately the new n_n values, and then to merge the old and new values; the cost is $O(N + n_n^2)$ comparisons and memory shifts. We present in table 1 the theoretical values for the usual case $n_n = O(\sqrt{N})$ and some experimental times for usual window sizes; for comparison, the third column shows the times obtained when all values were sorted from scratch; all times were obtained on a 512×512 image Airfield, corrupted at 6dB with contaminated Gaussian distribution, with contamination 0.1 [7]. It can be noticed that the "sort and merge" variant gives the best results.

3.2. Training the stack filter

We compare in Table 2 the results of 4 procedures. The first, FASTAR is an improved version of FASTAF, where the solution is computed recursively in the size of the window; the implementing program was available starting from August 1997 at [5]. The second is an improved version of the former, FASTAR*, making use of recursive procedures for evaluating the cost of the minterms to be added. The third is the new procedure introduced in this paper, and the fourth is our implementation of TRAIN[2].

In Table 3 we compare the results of the new procedure and of TRAIN[2], for larger window sizes, when we increase the size of the training set, by keeping the same clean (target) image, but provide many corrupted versions of it. We observe that extending the size of the training set the design stage is better conditioned, and as a result the design time decreases considerably.

4. REFERENCES

[1] E. J. Coyle and J.-H. Lin. Stack filters and the mean absolute error criterion. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-36:1244–1254, Aug. 1988.

[2] K.L. Fong, G.B. Adams III, E.J. Coyle, and J. Yoo. Synthesis of a parallel optimal stack

N	n_n	Running ordering		Usual sort
		Insertion	Sort + merge	
13	5	8.95	8.58	10.24
15	5	10.23	9.60	12.43
17	5	11.57	10.92	14.90
19	5	13.60	12.50	18.04
21	5	15.34	13.97	21.24
Theoretical complexity for $N = O(n_n^2)$:				
$O(n_n^2)$	n_n	$O(n_n \log n_n)$ comparisons $O(n_n^3)$ memory shifts	$O(n_n^2)$ comparisons $O(n_n^2)$ memory shifts	$O(n_n^4)$ comparisons $O(n_n^4)$ memory shifts

Table 1: Complexity and experimental times (in seconds) for running ordering.

III.5.0 Determine the sets $I_{u_k} = \{\underline{v} \in I_u | w_H(\underline{v}) = k\}$ and indices k_{min}, k_{max}.
III.5.1 Do
$Improve = 0$
For $k = k_{max} - 1 : -1 : k_{min}$
For all $\underline{v} \in I_{u_k}$ with $c(\underline{v}) < 0$
$Cost_{\text{diff}}(\underline{v}) = \sum_{\substack{\underline{x} \in I_u \\ \underline{x} \geq \underline{v}}} c(\underline{x})$
If $Cost_{\text{diff}}(\underline{v}) < 0$
For all $\underline{x} \in \{\underline{x} | \underline{x} \geq \underline{v}\} \bigcap I_u$
$f^+(\underline{x}) = 1$; $I_{u_k} \leftarrow I_{u_k} - \{\underline{x}\}$; $Improve = 1$
until $(Improve = 0)$
III.5.2 For $k = k_{max} - 2 : -1 : k_{min}$
For all $\underline{x} \in I_{u_k}$
$Sons = \emptyset$
For all $\underline{v} \in H_{down}(\underline{x})$
$Sons = Sons \bigcup \{\underline{v}\}$
$Cost_{\text{diff}}(Sons) = \sum_{\underline{w} \in I_u \bigcap W_{down}(Sons)} c(\underline{w})$
If $Cost_{\text{diff}}(Sons) < 0$
For all $\underline{w} \in W_{down}(Sons) \bigcap I_u$,
$f^+(\underline{w}) = 1$; $I_{u_k} \leftarrow I_{u_k} - \{\underline{w}\}$
$Sons = \emptyset$

Figure 1: The new algorithm for deciding the nodes in the undecided set of the Boolean function f. The Steps I,II,III1.-III4. are the same as in FASTAF algorithm [7].

	$N = 13$	$N = 14$	$N = 15$	$N = 17$
FASTAR $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.955}{1.45}$	$\frac{7.888}{10.6}$	$\frac{7.855}{93.4}$	$\frac{7.725}{4200}$
FASTAR* $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.955}{0.63}$	$\frac{7.888}{0.71}$	$\frac{7.855}{3.50}$	$\frac{7.725}{1340}$
New algorithm $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.955}{0.22}$	$\frac{7.887}{0.41}$	$\frac{7.856}{0.70}$	$\frac{7.727}{137}$
TRAIN[2] $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.955}{0.59}$	$\frac{7.887}{3.40}$	$\frac{7.855}{14.22}$	$\frac{7.718}{337}$

Table 2: Performance and time complexity for four procedures designed on the image Airfield corrupted at 6 dB.

	$N = 15$ $n_{im} = 10$	$N = 17$ $n_{im} = 20$	$N = 19$ $n_{im} = 40$	$N = 21$ $n_{im} = 80$
New algorithm $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.434}{0.28}$	$\frac{7.485}{1.89}$	$\frac{7.436}{51.4}$	$\frac{7.379}{5700}$
TRAIN[2] $(\frac{\text{MAE}}{\text{time}})$	$\frac{7.434}{6.92}$	$\frac{7.484}{129}$	$\frac{7.434}{613}$	$\frac{7.376}{2740}$

Table 3: Performance and time complexity for two procedures designed on the training set formed by the clean Airfield image and n_{im} versions of it, each corrupted at 6 dB.

filter training algorithm. In *NSIP'97, IEEE Workshop on Nonlinear Signal and Image Processing*, Mackinac Island,Ma, USA, Sep. 1997.

[3] J.-H. Lin and Y.-T. Kim. Fast algorithms for training stack filters. *IEEE Transactions on Signal Processing*, SP-42:772–781, April 1994.

[4] J.-H. Lin, T.M. Sellke, and E.J. Coyle. Adaptive stack filtering under the mean absolute error criterion. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-38:938–954, June 1990.

[5] A package for stack filter design. http://www.cs.tut.fi/~tabus/course//stack_design.html.

[6] I. Pitas. Fast algorithms for running ordering and min/max calculation. *IEEE Transactions on Circuits and Systems*, 36:795–804, June 1989.

[7] I. Tăbuş, D. Petrescu, and M. Gabbouj. A training framework for stack and Boolean filtering-Fast optimal design procedures and robustness case study. *IEEE Transactions on Image Processing Special Issue on Nonlinear Image Processing*, IP-5:809–826, June 1996.

[8] B. Zeng, M. Gabbouj, and Y. Neuvo. A unified design method for rank order, stack and generalized stack filters based on classical Bayes decision. *IEEE Transactions on Circuits and Systems*, CAS-38:1003–1020, Sept. 1991.