Reduced-Complexity Iterative Post-Filtering of Video

Mark A. Robertson and Robert L. Stevenson

***Abstract*—There are numerous methods of post-processing that make use of iterative techniques. Many of these schemes have been demonstrated to be very effective in removing artifacts from compressed video, producing better and better image estimates at each iteration. However, this artifact removal comes at the cost of a large computational burden. This paper introduces two methods for iterative post-processing of compressed video in an efficient manner. One of these methods is applicable to one particular maximum *a posteriori* scheme. The other method has direct application to other, more general, iterative post-processing schemes that make use of a convex *constraint set,* which is the set of all images that will recompress to yield the originally-received data. The combination of these two methods produces a post-processing algorithm that has the advantage of many iterative schemes (excellent visual results), while requiring a relatively low amount of computational effort to achieve these results.**

***Index Terms*—Compressed video post-processing, iterative filtering, MAP post-processing, quantization constraint set.**

I. Introduction

The BLOCK discrete cosine transform (BDCT) with motion compensation is widely used for the compression of video signals. It is commonly understood that the main drawback of using this coding method (at least in terms of reconstructed visual quality) is the presence of compression artifacts, which take the form of blocking and ringing. Numerous methods of removing or alleviating these artifacts exist. This paper will deal exclusively with those that are *iterative* in nature.

Many iterative schemes of post-processing have been proposed (e.g., [8], [13]–[16]). There are several advantages to using iterative methods. One advantage is that iterative schemes allow a high level of sophistication. For example, many post-processing formulations are quite involved from a theoretic standpoint, and do not lend themselves to a noniterative solution. Another advantage of iterative methods is that the image estimate is improving with each iteration (provided of course that the method was formulated in an intelligent manner). For the optimal solution, one would have to wait until the algorithm has converged to a solution. However, in a practical video post-processing application, the goal is to improve the visual quality, and actual convergence in not of the utmost importance. Thus, a second advantage for iterative schemes is that the video receiver can select the actual number of iterations to perform based on a tradeoff between the desired image quality and the computational resources available.

The main drawback of using iterative post-processing methods was hinted at in the previous paragraph: computational complexity. Thus, the very source of advantages is also the source of a major disadvantage—allowing a fairly high level of sophistication and using multiple iterations of an algorithm can result in computational requirements that may make the algorithm impractical for actual use.

The above points lead the would-be implementor of an iterative post-processing scheme to the following problem: how can this algorithm be implemented in a computationally efficient manner, while not compromising the visual quality of the result? This is the question that this paper addresses. In particular, two concepts will be presented. The first has been developed for a specific post-processing algorithm, namely the maximum *a posteriori* (MAP) scheme presented in [8]; it allows the MAP algorithm to be implemented with significantly less computational complexity than other reported methods. The second concept to be introduced is applicable to a larger class of iterative post-processing algorithms, including the MAP algorithm above. It deals with one aspect that is common to many post-processing schemes, namely those using a projection onto a *constraint space,* where the constraint space is defined as the set of images that re-code to match the originally-received image data. The work presented here has been presented in part in [12], [11], and is based upon the work in [10].

Section II gives a brief review of the MAP post-processing algorithm as used in this paper, and will provide necessary background and motivation for the rest of the paper. Section III gives one method of significantly decreasing the computational requirements of implementing this particular MAP filter. Section IV provides another method of increasing the efficiency of the MAP algorithm which can be used in conjunction with the method given in Section III, and also discusses the straightforward application of this method to other post-processing algorithms. Section V offers experimental results, followed by concluding remarks in Section VI.

II. MAP Background

Maximum *a posteriori* filters or estimators belong to a larger class of estimation schemes termed Bayesian estimation [6]. MAP filtering treats the image as a random field, with some prior probability distribution function. It then attempts to maximize the *a posteriori* probability of the random field given the observations, where the observations here are the compressed image data.

Assume the image is of dimension N by N. Let $\mathbf{z}$ be the vector random variable of length N^2 representing the image, and let $\mathbf{y}$ be the vector of observations (the received image/video data). For MAP estimation, the filtered image estimate $\hat{\mathbf{z}}$ will be

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z}}\{\log \Pr(\mathbf{y}|\mathbf{z}) + \log \Pr(\mathbf{z})\}. \tag{1}$$

Manuscript received October 18, 1999; revised July 12, 2001. This paper was recommended by Associate Editor O. K. Al-Shaykh.

The authors are with the Laboratory for Image and Signal Analysis, Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA.

Publisher Item Identifier S 1051-8215(01)09155-8.

An original image $\mathbf{z}$ is compressed as $\mathbf{y} = Q[H\mathbf{z}]$, where $Q[\cdot]$ is the quantization operation and H is the de-correlating transform, which is typically the BDCT. (Note that we are disregarding the motion-compensating offset common to video coding schemes for notational convenience.) In [8], it is shown that the compression model for $\mathbf{y}|\mathbf{z}$ for a single frame of video simplifies (1) to

$$\hat{\mathbf{z}} = \arg\max_{\mathbf{z}\in\mathcal{Z}}\{\log \Pr(\mathbf{z})\} \tag{2}$$

where $\mathcal{Z}$ is the set of all images that compress to $\mathbf{y}$, $\mathcal{Z} = \{\mathbf{z}\colon \mathbf{y} = Q[H\mathbf{z}]\}$.

A Markov random field (MRF) is incorporated in Pr($\mathbf{z}$) to model the individual frames of video. Details of the derivation can be found, for example, in [8]. Since this paper deals primarily with computational issues, the resulting optimization will simpmly be stated:

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}\in\mathcal{Z}} \sum_{1\le m,\, n\le N}\ \sum_{j,\, l\in\mathcal{N}_{m,\, n}} \rho_T(\mathbf{z}_{m,\, n} - \mathbf{z}_{j,\, l}) \tag{3}$$

where $\mathcal{N}_{m,\, n}$ is some subset of the eight nearest neighbors of the pixel located at coordinates $(m,\, n)$. The first summation over $m,\, n$ is a sum over all pixels in the image, and the second summation over $j,\, l$ is a sum over the neighbors of the pixel located at coordinates $(m,\, n)$. This function to be minimized will be referred to as the cost or objective function, and will be denoted by $C_{\mathcal{N},\, T}(\mathbf{z})$.

The function $\rho_T(\cdot)$ is the Huber penalty function [4], given as

$$\rho_T(u) = \begin{cases} u^2, & |u| \le T \\ T^2 + 2T(|u| - T), & |u| > T \end{cases} \tag{4}$$

and shown in Fig. 1. The Huber function has been used extensively in image processing applications due to its edge-preserving properties; see, for example, the work in [13]. Due to the convexity of $\rho_T(\cdot)$, the MAP estimate becomes a convex constrained minimization problem, which can then be solved by iterative techniques.

A. Minimization of the Objective Function

An iterative method is chosen to solve the nonlinear constrained minimization problem in (3). The initial estimate $\mathbf{z}^{(0)}$ is chosen as the standard decompression of $\mathbf{y}$

$$\mathbf{z}^{(0)} = H^{-1}Q^{-1}[\mathbf{y}].$$

The image is then successively filtered in such as way as to decrease the cost function at each iteration until some form of stopping criteria have been reached.

The gradient-projection method [1] is one possibility that may be used in the minimization of (3), and this is the approach taken in [8], [13]. Given the image estimate at the kth iteration, $\mathbf{z}^{(k)}$, the gradient of the objective function is taken to find the steepest direction $\mathbf{g}^{(k)}$ toward the minimum

$$\mathbf{g}^{(k)} = \nabla\left(\sum_{1\le m,\, n\le N}\ \sum_{j,\, l\in\mathcal{N}_{m,\, n}} \rho_T\left(\mathbf{z}_{m,\, n}^{(k)} - \mathbf{z}_{j,\, l}^{(k)}\right)\right). \tag{5}$$

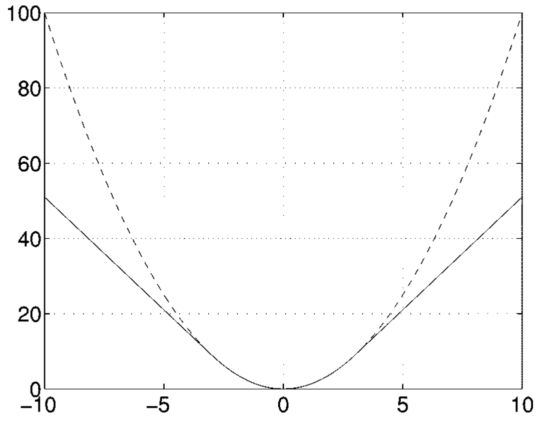


Fig. 1. $\rho_T(\cdot)$ superimposed on quadratic $T = 3.0$.

Evaluation of (5) requires the use of the first derivative of the Huber function

$$\rho'_T(u) = \begin{cases} 2T, & u > T \\ 2u, & |u| \le T \\ -2T, & u < -T. \end{cases} \tag{6}$$

There are any number of methods for determining a step size $\alpha^{(k)}$ to be used in conjunction with the direction found in (5). In [8] and [13], it is suggested that by using a quadratic approximation to the nonquadratic function in (3), an optimal value of $\alpha^{(k)}$ be found. However, since it is only an approximation, the resulting step size may be too large, which could result in actually increasing the cost function. Thus, the step size is successively divided by two until the cost function is in fact decreased.

A different way of determining $\alpha^{(k)}$ that does not make use of derivatives is to use the golden section search (GSS), an approximation to the Fibonacci search method [5]. The Fibonacci search method is the optimal method for minimization along a line given a fixed number of search points, whereas the GSS method does not require a fixed number of search points. The GSS search method starts by assuming that $\alpha^{(k)}$ belongs to some range, say $[\alpha_1,\, \alpha_2]$. By evaluating the cost function for various values of $\alpha^{(k)}$ within the allowable range, it successively refines the range of possible $\alpha^{(k)}$ until a value is found to satisfy some desired tolerance. If a value of $\alpha^{(k)}$ cannot be found that reduces the cost function, it is assumed that the minimum has been attained.

Regardless of how $\alpha^{(k)}$ is found, the updated estimate $\mathbf{w}^{(k+1)}$ is found as

$$\mathbf{w}^{(k+1)} = \mathbf{z}^{(k)} - \alpha^{(k)}\mathbf{g}^{(k)}.$$

Thus, for a particular iteration, the new cost function should be decreased, i.e., $C_{\mathcal{N},\, T}(\mathbf{w}^{(k+1)}) \le C_{\mathcal{N},\, T}(\mathbf{z}^{(k)})$.

Since there is no guarantee that $\mathbf{w}^{(k+1)}$ will lie within the constraint space $\mathcal{Z}$, $\mathbf{w}^{(k+1)}$ must be projected onto $\mathcal{Z}$ to give the image estimate at the $(k+1)$th iteration,

$$\mathbf{z}^{(k+1)} = P_{\mathcal{Z}}\left(\mathbf{w}^{(k+1)}\right).$$

The projection of $\mathbf{w}^{(k+1)}$ onto $\mathcal{Z}$ finds the $\mathbf{z}^{(k+1)} \in \mathcal{Z}$ such that $\|\mathbf{z}^{(k+1)} - \mathbf{w}^{(k+1)}\|$ is minimized. Obviously, if $\mathbf{w}^{(k+1)} \in \mathcal{Z}$, then $\mathbf{z}^{(k+1)} = \mathbf{w}^{(k+1)}$ and $\|\mathbf{z}^{(k+1)} - \mathbf{w}^{(k+1)}\| = 0$.

The transformation H in this case is the BDCT, which is just the transformation matrix that treats image blocks individually, and applies the DCT to each. Since the BDCT is a unitary transformation

$$\left|\left|H\mathbf{z}^{(k+1)} - H\mathbf{w}^{(k+1)}\right|\right| = \left|\left|\mathbf{z}^{(k+1)} - \mathbf{w}^{(k+1)}\right|\right|$$

and the projection can be carried out in the transform domain.

While the projection operation may seem a rather difficult concept when viewed from a spatial-domain standpoint, it is actually quite simple to understand when viewed from a frequency-domain standpoint. When the updated estimate $\mathbf{w}^{(k+1)}$ is determined, the forward BDCT is applied. Based on the compressed image data $\mathbf{y}$ and the quantization parameters for each block, there will be a range of allowable values that each transform-domain coefficient from $\mathbf{w}^{(k+1)}$ can assume. If any of these coefficients are outside of the allowable range, their values are clipped to the nearest boundary value to make them valid. The inverse BDCT is then applied to yield the constrained image block.

B. Difficulties of These Methods

The MAP filtering process described above has two main sources of computation—decreasing the objective function, and applying the quantization constraints. For the case of decreasing the objective function, the main source of computation comes from the necessity of multiple objective-function evaluations to ensure that the objective function is in fact decreased. If L represents the number of neighbors per pixel in $\mathcal{N}_{m,n}$ of the prior image model, then a single evaluation of the objective function will require N^2L evaluations of the Huber function; as can be seen from (4), this could get very expensive computationally if many evaluations of the objective function are required for each iteration. Many evaluations of the objective function may in fact be necessary, for although the "optimal" $\alpha^{(k)}$ derived in [8] and [13] gives an explicit expression for the step size, this step size is quite often too large, and the successive divisions-by-two reduce this method to a simple search for a step size to decrease the cost function. For the GSS, by its very definition it requires multiple evaluations of the objective function.

Application of the quantization constraints is the other main source of computation in the MAP algorithm. For each iteration, the forward BDCT must be applied prior to clipping the transform coefficients, followed by the inverse BDCT. Assuming 8×8 blocks and the Chen DCT [3], each block requires 416 additions and 256 multiplications for the forward transform, and the same number of computations for the inverse transform. This process is applied for each iteration.

The next section will provide an efficient method of decreasing the cost function, one which requires no evaluations of the objective function. The subsequent section will discuss a way to decrease the computational burden of applying the quantization constraints.

III. Decreasing the Objective Function

The approach taken here to reduce the objective function is a form of Seidel relaxation [9], where an iteration of the Seidel relaxation algorithm is attempting to solve the nonlinear equation $\mathbf{g}(\mathbf{z}) = \mathbf{0}$, i.e., the gradient of the objective function equal to the zero vector.

Keeping with the notation of Section II-A, it is assumed that the kth iterate $\mathbf{z}^{(k)} = [z_1^{(k)}, \ldots, z_{N^2}^{(k)}]^T$ and the first $(i-1)$ elements of the $(k+1)$th iterate $\mathbf{w}^{(k+1)} = [w_1^{(k+1)}, \ldots, w_{i-1}^{(k+1)}]^T$ have been determined. If $\mathbf{x}^{[\mathbf{i}]}$ then denotes the image whose elements consist of the first $(i-1)$ elements of $\mathbf{w}^{(k+1)}$, with the remaining elements from $\mathbf{z}^{(k)}$, the Seidel algorithm attempts to update the ith pixel $x_i^{[\mathbf{i}]}$ such that $g_i(\mathbf{x}^{[\mathbf{i}]}) = 0$, where $g_i(\mathbf{x}^{[\mathbf{i}]})$ is the ith element of the gradient of the objective function for the image $\mathbf{x}^{[\mathbf{i}]}$. This value of $x_i^{[\mathbf{i}]}$ is then substituted for $w_i^{(k+1)}$. This process is performed for $i = 1, \ldots, N^2$, which then completes one iteration of the Seidel algorithm. The final updated image $\mathbf{w}^{(k+1)}$ is then projected onto the quantization constraint set to yield $\mathbf{z}^{(k+1)}$, which completes one iteration of the MAP algorithm.

For the case at hand, for the ith pixel, the Seidel algorithm is attempting to replace $x_i^{[\mathbf{i}]}$ such that $g_i(\mathbf{x}^{[\mathbf{i}]}) = 0$, or

$$\sum_{l=1}^{L} \rho_T'(x_i^{[\mathbf{i}]} - x_{i,l}^{[\mathbf{i}]}) = 0 \tag{7}$$

where $x_{i,l}^{[\mathbf{i}]}$ is the lth neighbor of the pixel $x_i^{[\mathbf{i}]}$, and there are L total neighbors being used.

One way to solve (7) would be to apply Newton's method, which would result in

$$x_i^{[\mathbf{i}]} = x_i^{[\mathbf{i}]} - \omega \frac{\sum_{l=1}^{L} \rho_T'\left(x_i^{[\mathbf{i}]} - x_{i,l}^{[\mathbf{i}]}\right)}{\sum_{l=1}^{L} \rho_T''\left(x_i^{[\mathbf{i}]} - x_{i,l}^{[\mathbf{i}]}\right)} \tag{8}$$

where ω is a relaxation parameter. The procedure in (8) could then be repeated m times in an attempt to solve (7). Following the nomenclature of [9], this will be referred to as an m-step successive overrelaxation-Newton (SOR-Newton) method.

The problem with using Newton's method as above is that the second derivative of the Huber function $\rho_T''(u)$ is zero for $|u| > T$, which could result in the denominator of the term in (8) becoming zero. To avoid this potential problem, it is proposed here to modify $x_i^{[\mathbf{i}]}$ as

$$x_i^{[\mathbf{i}]} = x_i^{[\mathbf{i}]} - \frac{1}{2L} \sum_{l=1}^{L} \rho_T'\left(x_i^{[\mathbf{i}]} - x_{i,l}^{[\mathbf{i}]}\right). \tag{9}$$

Equation (9) is the parallel-chord (PC) method [9], which also can be repeated m times, resulting in an m-step SOR-PC method. It can be shown that using the m-step SOR-PC method is *guaranteed* to decrease the objective function, thus eliminating any need to actually evaluate the objective function [10].

The actual computational cost is an important issue to address as well. For the one-step SOR-PC method, if the "2" is factored out of $\rho_T'(\cdot)$ in (6) to cancel the "2" in the $1/(2L)$ term of (9), then the above procedure for decreasing the cost function will require $2L$ additions/subtractions, $2L$ comparisons, and $\log_2 L$

shifts; all these numbers are operations per pixel per iteration. The actual number of iterations and neighbors required for an application will be discussed later. Note that there are no multiplications or divisions in this smoothing operation, and the computational cost of an iteration is considerably less than with any other approach.

IV. Application of the Constraints

For each block, application of the quantization constraints requires computing and clipping the 64 DCT coefficients followed by the computation of the spatial-domain values via the inverse transform. It is proposed that in order to reduce the computational requirements of applying the quantization constraints, only a *subset* of the 64 DCT coefficients be evaluated and clipped. Similarly, only that subset of DCT coefficients will be used for performing the inverse BDCT. Since not all DCT coefficients are being used, this paper will use the term "partial constraints" to refer to this process.

A. Partial Constraints

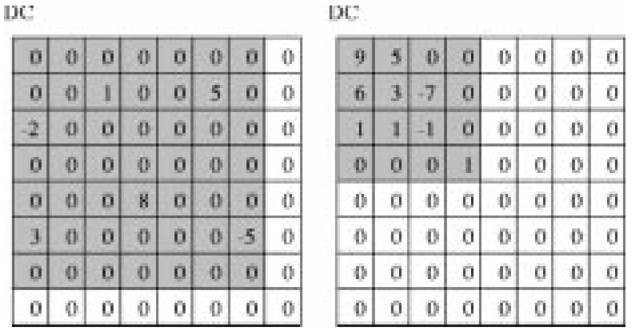
To be more precise, the subset of DCT coefficients to be evaluated and constrained is determined as follows. For the data originally received, take the smallest-sized square block of coefficients, with one corner of the block anchored on the DC component, such that all of the nonzero DCT coefficients lie within that block. This block then represents the coefficients to be evaluated and constrained. Thus, if $H(i, j)$ represents the original transform values at location (i, j) for a single block, and if $S \times S$ is the size of the sub-block to be used, then S is defined as

$$S = \arg\min_{k}: H(i, j) = 0 \qquad \forall\, i > k, \forall\, j > k \tag{10}$$

which is the smallest number such that $H(i, j) = 0$ for $i > S$ or $j > S$. What the partial constraint process is essentially doing, then, is to constrain only the lowest-frequency coefficients of a block; it does so in a manner such that all of the coefficients that were nonzero in the originally-received image data get constrained. As the constraint process proceeds from block to block, the size of the sub-block being constrained is varying. Fig. 2 shows example sub-blocks of sizes 7 and 4.

The justification for doing this partial constraint procedure is as follows. The quantization constraints are being used in conjunction with a smoothing operation; the constraints serve the purpose of preventing too much smoothing from being done (this is also true of other post-filtering methods that make use of a quantization constraint set, to be discussed in more detail later). It stands to reason, then, that it is unlikely for the smoothing operation to actually introduce high-frequency energy into the image, and thus one would feel justified in not constraining high-frequency components that were zero to begin with. However, for both high- and low- frequency DCT coefficients that were *not* zero originally, these still certainly need to be constrained because they represent valuable information about the image from the data stream which could possibly get oversmoothed if not constrained. With these considerations, the definition of the sub-block to be constrained makes sense.

In addition to the above intuitive justification for using the partial constraints, experimental evidence also suggests that it



0	0	0	0	0	0	0	0
0	0	1	0	0	5	0	0
-2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	8	0	0	0	0
3	0	0	0	0	0	-5	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

9	5	0	0	0	0	0	0
6	3	-7	0	0	0	0	0
1	1	-1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 2. Examples of sub-squares to be constrained.

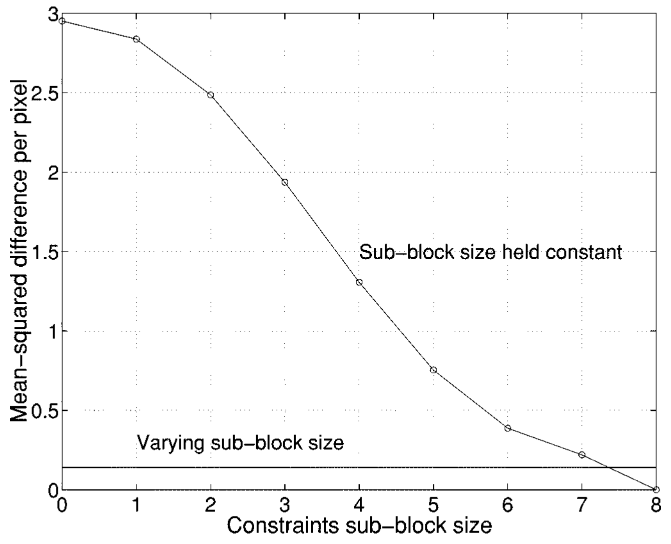


Fig. 3. Mean-square difference per pixel between full constraints and partial constraints.

is effective. Using the iterative MAP procedure discussed in Section III, objective comparisons were made between the algorithm's results when using the full (normal) constraints, and when using the partial constraints as outlined here. The reference sequence was generated by compressing a QCIF sequence to an H.263 bitstream of 64 kbps. Five iterations of the MAP algorithm were performed, which is enough to effectively remove nearly all blocking and ringing artifacts. Several comparisons were performed. First, sub-block sizes were held *constant* for the entire sequence, and then compared to the reference sequence. For this method, the sub-block sizes were varied from size 0 (constraining no coefficients) through size 8 (constraining all coefficients). Second, the sub-block sizes were varied from block to block depending on the size of the original sequence's sub-blocks, as defined in (10). The results for filtering the luma plane are shown in Fig. 3.

The smooth line in Fig. 3 is for holding the sub-block sizes constant, and it makes intuitive sense: as the number of DCT coefficients being constrained increases, the error between full and partial constraints decreases, until there is zero error for a "partial constraint" of size 8. The horizontal line in Fig. 3 is the error for using the partial constraints with sub-block sizes as in (10). The mean-squared difference per pixel is roughly 0.2 by using this partial constraint procedure, and is less than always constraining the 7×7 lowest-frequency sub-block. Results

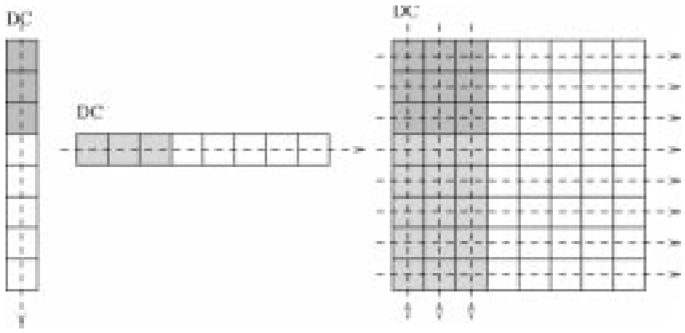


Fig. 4. Evaluation of 2-D partial DCT using separable extension of 1-D partial DCT.

are similar when sequences are compressed to other bit-rates as well, as are when the chroma planes are also filtered.

Finally, note that the objective measures between using full constraints and partial constraints say nothing about *subjective* comparisons; when sequences post-processed using full constraints are visually compared with those post-processed with the partial constraints, the two are virtually indistinguishable.

B. Computational Savings

In order to examine the computational savings of using partial constraints, the details of its implementation must be explained. In practice, the 8×8 2-D DCT is typically evaluated by performing 16 1-D DCTs in a separable fashion. Here, it is proposed to compute the partial DCT similarly; however, rather than evaluating all 8 of 8 1-D DCT values 16 times as in the full constraints, only S of the 8 1-D DCT values need to be evaluated $(8+S)$ times. In other words, if a 1-D partial DCT of size S is defined to be the S lowest frequency components of the length-8 DCT, then the 2-D partial DCT with sub-block size S can be accomplished by performing $(8+S)$ 1-D partial DCT's. Fig. 4 shows an example of this.

Computing only S of 8 1-D DCT coefficients requires less computational effort than does computing all 8. If the Chen DCT is used, then the required computations are as shown in Table I. The computations for the 1-D case are the result of merely not performing any operations that are unnecessary to compute the S of 8 coefficients. Computations for the 2-D case are $(8+S)$ times the computations for the 1-D case.

Similar results are obtained for the partial IDCT. The main difference between the partial DCT and the partial IDCT is that whereas the partial DCT is taking 64 spatial-domain values and computing S^2 transform-domain values, the partial IDCT is taking S^2 transform-domain values and computing 64 spatial-domain values.

Unfortunately, however, the partial IDCT is not quite as straightforward as the partial DCT. While the original DCT coefficients are all within the sub-block being used (by definition), there is no guarantee that all of the *filtered* DCT coefficients will be within the sub-block. Thus merely doing a partial IDCT on the constrained coefficient sub-block and assuming the rest of the coefficients are zero will not necessarily be a good idea. To rectify this potential problem, the procedure shown in Fig. 5 is proposed.

TABLE I
OPERATIONS PER 8 BY 8 BLOCK FOR THE PARTIAL DCT. NOTE THAT FOR SUB-BLOCKS OF SIZE 0, THE OPERATION COUNT IS 0

Operation	Operations for sub-blocks of size:							
	1	2	3	4	5	6	7	8
1-D ×'s	1	5	7	9	10	12	14	16
1-D +'s	7	16	19	22	23	24	25	26
2-D ×'s	9	50	77	108	130	168	210	256
2-D +'s	63	160	209	264	299	336	375	416

The input in Fig. 5 is the input block to be constrained. The partial DCT is taken as described previously, and the resulting DCT coefficients are clipped to their allowable ranges. Next, the un-clipped partial DCT coefficients are subtracted from the clipped coefficients, where the result represents the transform-domain difference between the constrained and unconstrained image block. The partial IDCT is then taken, with the result being the spatial-domain difference between the constrained and unconstrained image block. To arrive at the constrained block, the result is added to the original unconstrained block. For an $S \times S$ sub-block size, these extra operations introduce $64 + S^2$ additions per image block to the operation count, in addition to the number of operations required for the actual partial IDCT. Using this procedure allows the partial constraint process to retain frequency information in a filtered image block that is actually outside of the sub-block.

The next important consideration for determining the computational savings of using partial constraints is the expected size of the sub-blocks. For video applications, due to motion compensation many DCT blocks will have very few high-frequency transform coefficients, and hence small sub-block sizes. This results in the need to evaluate only a few DCT values, which leads to considerable computational savings. Bit-rate affects these savings (lower bit-rate video has smaller sub-blocks than high bit-rate video, due to more severe quantization), and so does the choice of whether or not the chroma planes are being post-filtered (chroma blocks typically have smaller sub-block sizes than do luma blocks). To get an idea of the expected sub-block sizes, three separate QCIF sequences were compressed to various bit-rates, with the sub-block sizes recorded for each. Fig. 6 shows results for sequences compressed to 64 kbps, for both luma and chroma planes. The three separate bars for each sub-block size represent the percentages of each of the three sequences with that particular sub-block size. As can be seen from the figure, a significant portion of the sub-blocks are in fact of size zero, and hence need not be constrained at all; this trend is especially evident for the chroma planes. The results for 32 and 96 kbps are similar to those presented in Fig. 6, but with the number of zero-size sub-blocks slightly more and less, respectively.

If the average of the results of the three test sequences for sub-block sizes from Fig. 6 are used as actual expected block sizes, then expected computational savings can be determined. Table II shows expected savings if only the luma plane is filtered, and Table III shows expected savings if both the luma and chroma planes are post-filtered. It has been assumed that the chroma planes are sub-sampled by a factor of two in each spatial dimension relative to the luma plane. The results of the

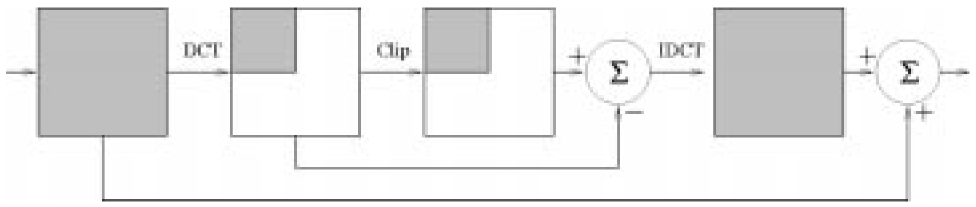


Fig. 5. IDCT in the partial-constraint process.

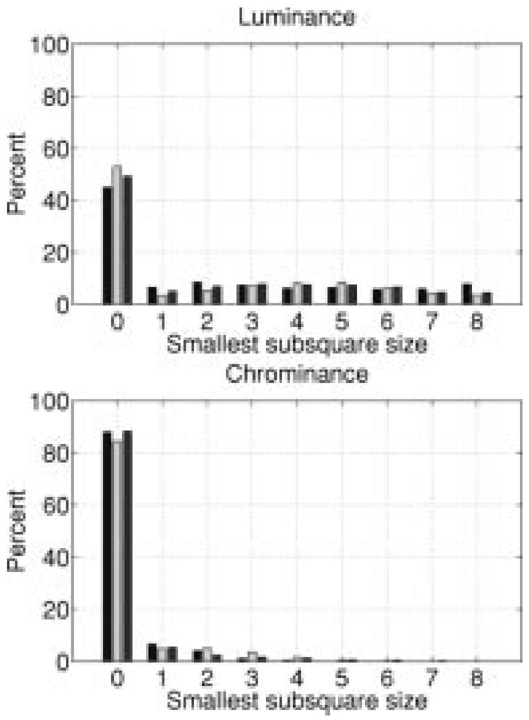


Fig. 6. Sub-block sizes for three sequences compressed to 64 kbps.

TABLE II
COMPARISON OF OPERATIONS PER PIXEL PER CONSTRAINT FOR FULL AND PARTIAL CONSTRAINTS, LUMA ONLY

Operation	Full Constraints	Partial Constraints		
		32 kbps	64 kbps	96 kbps
Additions	15.0	3.0	5.5	7.0
Multiplies	8.0	0.9	1.9	2.7
Compares	2.0	0.2	0.4	0.6

two tables are in terms of operations per pixel per application of the constraints.

C. Other Potential Applications

The particular MAP algorithm discussed in this paper is not the only method that could benefit from using partial constraints. For example, a MAP formulation similar to the one discussed in this paper, but which uses a different prior image model, would also result in a constrained optimization problem and could benefit from using the partial constraints as discussed here. For example, if the absolute value MRF discussed in [7], or the generalized Gaussian MRF of [2] were formulated in the same image and video post-processing context as discussed in this paper, they would both require application of the quantization constraints and could benefit from using partial constraints.

TABLE III
COMPARISON OF OPERATIONS PER PIXEL PER CONSTRAINT FOR FULL AND PARTIAL CONSTRAINTS, LUMA AND CHROMA

Operation	Full Constraints	Partial Constraints		
		32 kbps	64 kbps	96 kbps
Additions	22.5	3.2	5.9	7.7
Multiplies	12.0	1.0	2.0	2.8
Compares	3.0	0.2	0.4	0.6

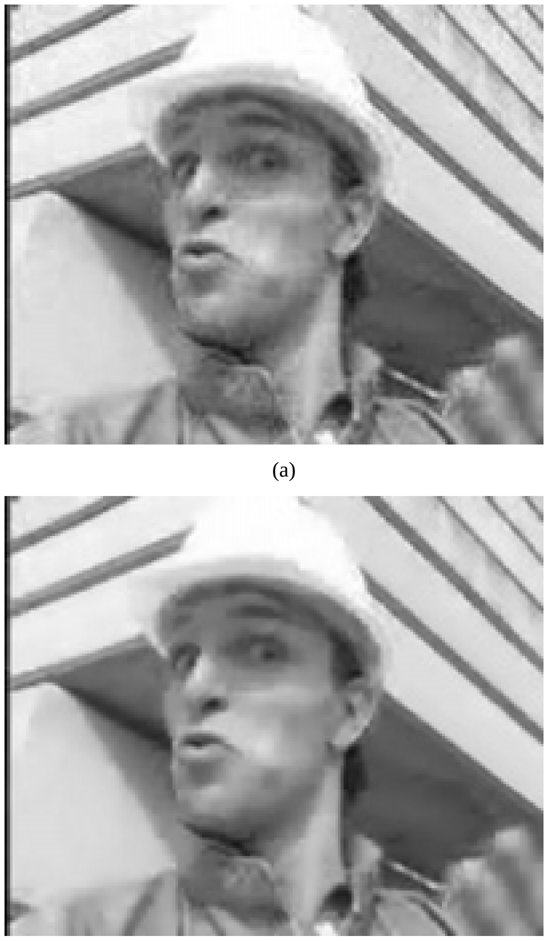
Another example would be post-processing algorithms that make use of the theory of projection onto convex sets (POCS). These POCS methods often define one of the convex sets to be the quantization constraint set discussed here [14]–[16]. Strictly speaking, the partial constraints process is not a true projection. However, if one is not concerned about absolute convergence of the algorithm (which would be the case for an actual video post-processing situation), the POCS algorithms could benefit from using the partial constraints.

V. EXPERIMENTAL RESULTS

Before giving actual visual demonstrative results, parameters of the MAP algorithm which affect its performance need to be discussed. In particular, there are three degrees of freedom for the MAP filter: 1) the parameter T in the Huber function; 2) the number of neighbors used, L; and 3) the number of iterations. These three parameters essentially determine what the quality of the filtered image will be. Aside from visual quality, the number of iterations and the number of neighbors used also have a direct impact on the algorithm's computational complexity.

Here, the Huber parameter is chosen as $T = 3$, which has provided good visual results in experiments. To keep computational costs low, a neighborhood structure with four pixels—the two horizontal and two vertical neighbors of a pixel—is used, implying $L = 4$. Experiments show that using three iterations of the MAP algorithm provides a good tradeoff between computational requirements and visual results.

The test sequence used here was the QCIF sequence *foreman*. The *foreman* sequence was compressed according to the H.263 standard, with constant quantization parameter $QP = 10$, resulting in a bit rate of 64 kbps. No H.263 advanced coding options were used. Fig. 7 shows visual results for a single frame of the sequence. As can be seen from the figure, the picture quality has been significantly improved. For processing the luma of this particular sequence, the average operations per pixel for the constraint procedure were 0.38, 0.27, and 0.20 of the additions, multiplications, and comparisons of the full constraints. If the chroma planes are also processed, the average operations per pixel were 0.27, 0.18, and 0.13 of the additions, multiplications, and comparisons of the full constraints. The filtering procedure



(a)

(b)

Fig. 7. Example post-processing results. (a) Decoded frame. (b) Post-processed frame.

of Section III introduces an additional 24 additions, 24 comparisons, and 6 shifts per pixel.

VI. Conclusion

In conclusion, this paper discussed two methods for reducing the computational requirements of iterative post-filtering of video. The first of these methods is particular to the MAP formulation in [8]. By using a form of Seidel relaxation, a computationally efficient method of reducing the objective function is achieved. Using this method guarantees a decrease in the objective function, thus eliminating the need to actually evaluate the objective function. The second method introduced here was referred to as "partial constraints." This method involved constraining only a sub-block of DCT coefficients in each block. For low- to moderate-bitrate video, sub-blocks are typically quite small, which results in significant computational savings. This partial constraints procedure is applicable not only to the MAP algorithm discussed in detail here, but is also applicable to other post-processing schemes. Due to the potentially large computational requirements of some iterative post-filters, the computational savings can be quite important.

Using the two methods discussed in this paper together results in a computationally efficient iterative algorithm for reducing the compression artifacts in video post-processing applications.

References

[1] M. Bazaraa, H. Sherali, and C. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. New York: Wiley , 1993.

[2] C. Bouman and K. Sauer, "A generalized Gaussian image model for edge-preserving MAP estimation," *IEEE Trans. Image Processing*, vol. 2, pp. 296–310, July 1993.

[3] W. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004–1009, 1977.

[4] P. Huber, *Robust Statistics*. New York: Wiley, 1981.

[5] S. Jacoby, J. Kowalik, and J. Pizzo, *Iterative Methods for Nonlinear Optimization Problems*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[6] S. Kay, *Statistical Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[7] L. Onural, M. Alp, and M. Gürelli, "Gibbs random field model based weight selection for the 2-D adaptive weighted median filter," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 831–837, Aug. 1994.

[8] T. O'Rourke and R. Stevenson, "Improved image decompression for reduced transform coding artifacts," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 490–499, Dec. 1995.

[9] J. Ortega and W. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic, 1972.

[10] M. Robertson, "Computationally-efficient post-processing of compressed video streams," Master's thesis, University of Notre Dame, Notre Dame, IN, 1998.

[11] M. Robertson and R. Stevenson, "Reducing the complexity of iterative post-processing of video," in *Proc. Midwest Symp. Circuits Syst.*, Aug. 1998, pp. 399–402.

[12] M. Robertson and R. Stevenson, "Reducing the computational complexity of a MAP post-processing algorithm for video sequences," in *Proc. Int. Conf. Image Processing*, vol. 1, Oct. 1998, pp. 372–376.

[13] R. Stevenson, "Reduction of coding artifacts in low-bit rate video coding," in *Proc. 38th Midwest Symp. Circuits and Systems*, Rio de Janeiro, Brazil, Aug. 1995, pp. 854–857.

[14] Y. Yang, N. Galatsanos, and A. Katsaggelos, "Regularized reconstruction to reduce blocking artifacts of block discrete cosine transform compressed images," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 421–432, Dec. 1993.

[15] Y. Yang and N. Galatsanos, "Removal of compression artifacts using projections onto convex sets and line process modeling," *IEEE Trans. Image Processing*, vol. 6, pp. 1345–1357, Oct. 1997.

[16] A. Zakhor, "Iterative procedures for reduction of blocking effects in transform image coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 91–95, Mar. 1992.