Exact JPEG recompression

Andrew B. Lewis, Markus G. Kuhn

Computer Laboratory, University of Cambridge,
15 JJ Thompson Avenue, Cambridge CB3 0FD, UK

ABSTRACT

We present a variant of the JPEG baseline image compression algorithm optimized for images that were generated by a JPEG decompressor. It inverts the computational steps of one particular JPEG decompressor implementation (Independent JPEG Group, IJG), and uses interval arithmetic and an iterative process to infer the possible values of intermediate results during the decompression, which are not directly evident from the decompressor output due to rounding. We applied our exact recompressor on a large database of images, each compressed at ten different quality factors. At the default IJG quality factor 75, our implementation reconstructed the exact quantized transform coefficients in 96% of the 64-pixel image blocks. For blocks where exact reconstruction is not feasible, our implementation can output transform-coefficient intervals, each guaranteed to contain the respective original value. Where different JPEG images decompress to the same result, we can output all possible bit-streams. At quality factors 90 and above, exact recompression becomes infeasible due to combinatorial explosion; but 68% of blocks still recompressed exactly.

1. INTRODUCTION

Lossy perceptual coding algorithms (JPEG, MPEG, etc.) were designed to compress raw audio-visual data captured by sensors. Ideally, such data should only ever go through a single lossy compression/decompression cycle. In practice, however, an image is often compressed already at the beginning of its editing history, such as in a digital camera, and repeatedly compressed later, after decompression and editing. Only very limited manipulation is practical without first decompressing (e.g., cropping and copying along certain boundaries, rotation by 90°, mirroring, scaling by certain factors[1]).

A *recompressor* is an alternative compression algorithm, designed to process data that was output by a decompressor. Depending on how strictly the decompression process was standardized, a recompressor may have to be designed specifically for a given decompressor implementation. We call a recompressor *exact* if its output is either identical to the input of the preceding decompressor, or equivalent to it, such that it decompresses to the same result and is not longer (Figure 1); *complete* if it generates the set of all equivalent inputs; and *stable* if localized edits on the data after decompression lead to only localized further loss of information during recompression. *Naïve recompression* is the application of standard lossy compression algorithms to decompressed data, which in practice rarely achieves exact or stable recompression, even with identical quantization and subsampling parameters.[2] This is due to rounding, range limiting, and the fact that some compressors do not apply the inverse functions of the corresponding decompression steps.

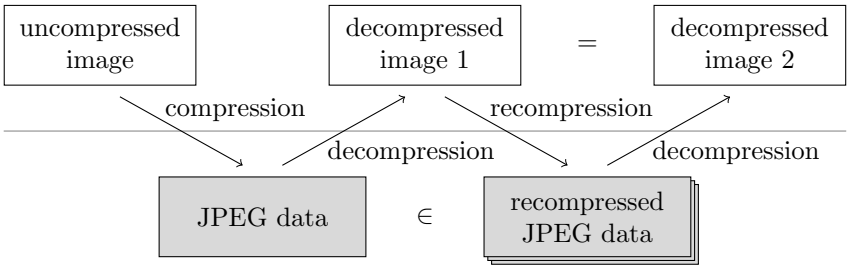

Figure 1. Exact recompression maps a decompressed image onto a set of equivalent possible original JPEG bit-streams.

1.1 Relationship to previous work

Various techniques have been proposed for quantization-table estimation,[3,4] which use the forward discrete cosine transform (DCT) and histograms of its coefficients to estimate the quantization factors used. As this does not, in practice, exactly invert the corresponding inverse DCT (IDCT) implementation, the resulting small perturbations blur the peaks in the resulting DCT coefficient histograms, which hinders the estimation of quantization factors and warrants probabilistic techniques. In contrast, we invert the implementation of both the inverse DCT and the chroma-interpolation operations exactly. Where we lack the information needed to unambiguously identify a single input integer, we output the smallest interval guaranteed to contain the corresponding original value.

Neelamani et al.[2] have described methods for finding the JPEG compression history of an image, that is, the colour-space, chroma-interpolation method and quantization tables used during compression. They could be used as heuristics to speed up selection of the appropriate exact recompressor for an image where the compressor and its options are unknown.

1.2 Applications

One application for exact, stable recompression are editors that read and write compressed data. Plugins exist already for some image editors to keep track of which blocks have not been modified since a JPEG image was opened, so that these blocks can be copied directly in compressed form when the image is saved.[5] The complexity of integrating this into the application could be avoided with an exact, stable recompressor, which can be applied without auxiliary information.

Many copy-protection mechanisms keep the compressed bit-stream confidential, via encryption, giving end-users plaintext access only to the output of the decompressor. Their reliance on the unavailability of exact recompression motivates its study, to understand both its practicality and how best to modify decompressors to prevent it without reducing signal quality.

As a by-product, exact recompression can also reveal information that may be of interest in forensic analysis of uncompressed data. It recovers parameters used during previous compression, which may give clues about which compressor was used before. It can also reveal whether decompressed data (or which unmodified sections of it) could possibly have been the output of a particular given decompressor implementation or not, providing further clues about the processing history and which parts of the data were modified since the last decompression.

Exact recompression can also be used to hinder forensic analysis. Naïve recompression can leave clear traces in the data, such as the JPEG double-compression artifacts described by Popescu and Farid,[6] whereas exact recompression adds no further information.

For some data, a recompression algorithm may even lead to a more compact representation, as it will search for those quantization parameters that lead to the smallest output without altering content. Recompression algorithms could also be extended to assist in compressing the data more aggressively (using different quantization parameters) without introducing recompression artifacts, that is, as if the lower quality setting had been chosen initially.

1.3 Contribution

In Section 2, we describe an exact, complete, stable recompressor for the DCT-based JPEG baseline algorithm,[7] as implemented in the widely-used open-source Independent JPEG Group (IJG) decompressor[8] (version 6b). It reads an RGB image and either returns one or more JPEG bit-streams that on IJG decompression will result in the input image, or it identifies regions in the input that could not have been output by the IJG decompressor. The recompression is exact because our algorithm, when provided with a decompressed image, returns a (normally singleton) set of bit-streams, including one with the original JPEG data (Figure 1).

The JPEG decompressor algorithm consists of a lossless entropy decoding stage followed by (a) dequantization of values in 8×8 blocks based on two quantization matrices (luma and chroma) given in the bit-stream, (b) application of the inverse discrete cosine transform (IDCT) to each block, (c) up-sampling of the two chroma planes to the same size as the luma plane and (d) conversion of colour values from the $\mathrm{YC_bC_r}$ space to the RGB space*.

*Other colour-spaces and subsampling schemes are possible, but here we consider the default settings.

The input to the recompressor is an uncompressed RGB image. Our recompressor implementation inverts each stage in turn, refining sets of intermediate values in an iterative process, until it reaches a fixed point. At any stage during execution, the sets are guaranteed to contain the intermediate values that actually occurred during the preceding decompression. If a set becomes empty (for example, due to intersection with a disjoint set), this indicates that the input image was not produced by the exact recompressor's corresponding decompressor implementation, or that the image was modified after decompression.

2. EXACT RECOMPRESSION OF JPEG IMAGES

All scalars in this paper are integer, unless otherwise stated. Scalars, vectors and matrices over integers carry no accent, whereas those over sets of integers are marked as $\ddot{x}$ and those over intervals of integers are marked as $\bar{x}$. The interval $\bar{x} = [x_\perp, x_\top]$ contains the integers from $x_\perp$ up to $x_\top$ inclusive, with element membership, "union" and intersection defined as

$$\begin{aligned} y \in \bar{x} &:= x_\perp \leq y \leq x_\top \\ \bar{x} \cup \bar{y} &:= [\min\{x_\perp, y_\perp\}, \max\{x_\top, y_\top\}] \\ \bar{x} \cap \bar{y} &:= \begin{cases} \text{empty} & \text{if } x_\perp > y_\top \vee y_\perp > x_\top, \\ [\max\{x_\perp, y_\perp\}, \\ \ \min\{x_\top, y_\top\}] & \text{otherwise.} \end{cases} \end{aligned}$$

The input RGB image has width w and height h divisible by sixteen:[†]

$$u_{x,y} \in \{0, \ldots, 255\}^3 \qquad \text{with } 0 \leq x < w = 16 \cdot w_b \text{ and } 0 \leq y < h = 16 \cdot h_b. \tag{1}$$

2.1 Colour-space conversion

The final step in the decompressor applies the colour-space conversion function C to each pixel, which converts each 3×8-bit luma and chroma ($\mathrm{YC_bC_r}$) value into a 3×8-bit RGB value.

Our recompressor inverts C. C^{-1} maps each RGB triple in the range of C to all $\mathrm{YC_bC_r}$ values in the domain of C that map to this RGB colour. C^{-1} therefore associates with each RGB colour a set of possible $\mathrm{YC_bC_r}$ colours. We apply C^{-1} to each input RGB pixel $u_{x,y}$, resulting in a set ($\ddot{v}_{x,y}$) for each pixel:

$$\begin{gathered} \ddot{v}_{x,y} = C^{-1}(u_{x,y}) \\ C^{-1} : \{0, \ldots, 255\}^3 \to \mathcal{P}\left(\{0, \ldots, 255\}^3\right) \text{ and} \\ C^{-1} : (r, g, b) \mapsto \left\{(y, cb, cr) \in \{0, \ldots, 255\}^3 : C(y, cb, cr) = (r, g, b)\right\}. \end{gathered}$$

In our implementation, the inversion of C is tabulated in a 2^{24}-entry look-up table, mapping RGB values to sets of $\mathrm{YC_bC_r}$ values.

The RGB triple $(0, 255, 0)$ is associated with 29714 possible $\mathrm{YC_bC_r}$ values, the largest set in the domain of C^{-1}. Around 10^7 RGB triples are associated with an empty set, as the decompressor can never output them.

2.2 Chroma downsampling

In the default sub-sampling mode, both chroma channels are stored at half the horizontal and vertical resolution of the luma channel. In the up-sampling calculation, each sample of the input chroma plane is located in the centre of a 2×2 square of output samples. By default, the IJG decompressor uses the up-sampling filter shown in Figure 2.2. The down-sampled planes are padded by repetition at each edge so that values on the border of

[†]The information loss caused by post-decompression cropping across a block boundary can be modelled using intervals, but our initial implementation does not support this.

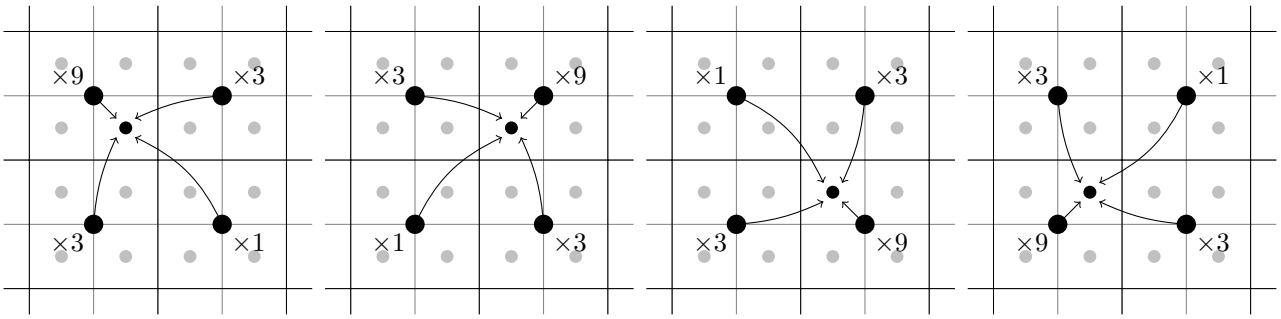

Figure 2. The chroma up-sampling filter in the IJG decompressor weights contributions from neighbouring samples by $\frac{1}{16}(1,3,3,9)$ in order of increasing proximity.

the image can be calculated in the same way as inner samples. Given a down-sampled plane of pixels $w_{i,j}$ (with $-1 \le i \le \frac{w}{2}$ and $-1 \le j \le \frac{h}{2}$) the up-sampled values are calculated as

$$v_{x,y} = \left\lfloor \frac{1}{16}\left(8 + \alpha \cdot w_{i-1,j-1} + \beta \cdot w_{i,j-1} + \gamma \cdot w_{i-1,j} + \ \delta \cdot w_{i,j}\right) \right\rfloor, \tag{2}$$

with weights

$$(\alpha,\beta,\gamma,\delta) = \begin{cases} (1,3,3,9) & \text{if } x = 2i,\ y = 2j \\ (3,1,9,3) & \text{if } x = 2i-1,\ y = 2j \\ (3,9,1,3) & \text{if } x = 2i,\ y = 2j-1 \\ (9,3,3,1) & \text{if } x = 2i-1,\ y = 2j-1. \end{cases} \tag{3}$$

The division by sixteen and subsequent rounding causes information loss, but we mitigate this by exploiting the redundancy introduced when the up-sampling process quadruples the number of pixels. We iteratively recover as much information as possible about the down-sampled chroma planes, using interval arithmetic to keep track of uncertainty. The algorithm represents knowledge of possible down-sampled values $w_{i,j}$ in the form of intervals $\bar{w}_{i,j}$. These are initialized to $[0, 255]$ and then refined repeatedly using the known sets of possible values in the up-sampled planes $\ddot{v}^{\mathrm{Cb}}_{x,y}$, $\ddot{v}^{\mathrm{Cr}}_{x,y}$ and our current estimates for the down-sampled planes $\bar{w}_{i,j}$, by rearranging (2).

To evaluate formulae with intervals as variables, we use

$$[q_\perp, q_\top] + [r_\perp, r_\top] = [q_\perp + r_\perp, q_\top + r_\top] \tag{4a}$$

$$[q_\perp, q_\top] \times \alpha = [q_\perp \times \alpha, q_\top \times \alpha] \tag{4b}$$

and to rearrange these formulae, we use

$$[p_\perp, p_\top] = [q_\perp, q_\top] + \alpha \implies [q_\perp, q_\top] = [p_\perp - \alpha, p_\top - \alpha] \tag{5a}$$

$$[p_\perp, p_\top] = [q_\perp, q_\top] \times \alpha \implies [q_\perp, q_\top] = \left[\left\lceil \frac{p_\perp}{\alpha} \right\rceil, \left\lfloor \frac{p_\top}{\alpha} \right\rfloor\right] \tag{5b}$$

$$[p_\perp, p_\top] = \left\lfloor [q_\perp, q_\top] \times \frac{1}{\alpha} \right\rfloor \implies [q_\perp, q_\top] = [p_\perp \times \alpha, p_\top \times \alpha + (\alpha - 1)]. \tag{5c}$$

Note that in (5b), if $p_\perp$ or $p_\top$ is not divisible by α, we set the wanted interval $\bar{q}$ to the smallest interval that on multiplication by α will contain all the multiples of α in $\bar{p}$.

Rearranging equation (2) gives the interval for a particular down-sampled value $w_{i',j'}$ in terms of the other down-sampled values in the formula and the up-sampled output value $v_{x,y}$. For example, applying the rules in (5) to solve (2) for a down-sampled value $w_{i,j}$ in relation to an up-sampled value $v_{x,y}$ with $x = 2i,\ y = 2j$, we get

$$\begin{aligned} \bar{w}_{i,j} = \Bigg[& \left\lceil \frac{1}{\delta}\left(v_{x,y} \times 16 - \left(8 + \alpha \cdot \bar{w}_{i-1,j-1} + \beta \cdot \bar{w}_{i,j-1} + \gamma \cdot \bar{w}_{i-1,j}\right)\right) \right\rceil, \\ & \left\lfloor \frac{1}{\delta}\left(v_{x,y} \times 16 + 15 - \left(\alpha \cdot \bar{w}_{i-1,j-1} + \beta \cdot \bar{w}_{i,j-1} + \gamma \cdot \bar{w}_{i-1,j}\right)\right) \right\rfloor \Bigg] \end{aligned} \tag{6}$$

where $v_{x,y} \in \ddot{v}^c_{x,y}$ is an integer and the $\bar{w}_{i',j'}$ variables are intervals. Where $\ddot{v}^c_{x,y}$ contains several possible values $v_{x,y}$, we have to evaluate the right-hand side of (6) for each and assign their union to $\bar{w}_{i,j}$. Each sample in each down-sampled colour plane c is the subject of 16 equations, of which (6) is one example, because each down-sampled value $w_{i',j'}$ influenced the surrounding 16 pixels when the decoder up-sampled them. The overdetermined system of equations is solved iteratively using Algorithm 1. We change the scan order in each iteration of the algorithm to accelerate convergence.

Algorithm 1 Down-sample $\ddot{v}^c_{x,y}$

$k \leftarrow 0$
$\bar{w}^0_{i,j} \leftarrow [0, 255]$ at all positions $-1 \le i \le \frac{w}{2}, -1 \le j \le \frac{h}{2}$
repeat
  $k \leftarrow k + 1$
  change scan order of (x, y) ($\rightrightarrows$, $\leftleftarrows$, $\vdots\!\rightrightarrows$, $\vdots\!\leftleftarrows$)
  for each sample position (x, y) in the up-sampled plane **do**
    set (i, j) based on (x, y) using Equation (3)
    for $(i', j') \in \{(i-1, j-1), (i, j-1), (i-1, j), (i, j)\}$ **do**
      $\bar{w}^k_{i',j'} \leftarrow \bar{w}^{k-1}_{i',j'} \cap \bigcup_{s \in \ddot{v}^c_{x,y}} \bar{a}$, where $\bar{a}$ is the smallest possible interval that satisfies Equation (2) if we substitute $\bar{a}$ for $w_{i',j'}$, s for $v_{x,y}$ and the current estimates $\bar{w}^{k-1}$ for the other w values.
    end for
  end for
until $\bar{w}^k = \bar{w}^{k-1}$
return $\bar{w}^k$

Since the two chroma channels were up-sampled independently, we also down-sample them independently in Algorithm 1. As the next step (inverting the IDCT) also relies on interval arithmetic, we convert the sets of possible luma values (at the original resolution) to an interval representation: $\bar{w}^{\mathrm{Y}}_{x,y} = [\min\{\ddot{v}^{\mathrm{Y}}_{x,y}\}, \max\{\ddot{v}^{\mathrm{Y}}_{x,y}\}]$.

2.3 Discrete cosine transform

The IJG decompressor implements the inverse DCT as

$$\mathrm{IDCT}(X) = \max\left\{0, \min\left\{255, \left\lfloor \frac{1}{2^{18}} \left(\left\lfloor \frac{1}{2^{11}} \left(TX + 2^{10}\right) \right\rfloor T^{\mathrm{T}} + 2^{17} \right) \right\rfloor \right\}\right\} \tag{7}$$

where X is an 8×8 matrix of DCT coefficients, T is the underlying 1-D IDCT matrix used (see Appendix A[‡]) and addition of a matrix and a scalar applies element-wise. Due to rounding and clipping, (7) is not linear and not exactly inverted by the DCT in the IJG compressor.

We need to solve $\mathrm{IDCT}(X) = x$ for X, knowing only that the 8×8 matrix result x lies within a given interval matrix $\bar{x}$. This will result in a frequency-domain interval matrix $\bar{X}$. We can later refine the result to

$$\bar{X}' = \{X \in \bar{X} : \mathrm{IDCT}(X) \in \bar{x} \wedge X \text{ matches quantization constraints}\}. \tag{8}$$

The interval matrices $\bar{x}$ stem from the previously down-sampled chroma ($\bar{w}^{\mathrm{Cb}}_{i,j}$, $\bar{w}^{\mathrm{Cr}}_{i,j}$) and luma ($\bar{w}^{\mathrm{Y}}_{x,y}$) planes, tiled into non-overlapping 8×8 blocks.

To invert (7), we apply the rules in (5), in conjunction with an additional rearrangement rule for matrix pre-multiplication: if $TX = \bar{Y}$, given an interval matrix $\bar{Y}$ and a non-singular fixed matrix T, we can efficiently find an interval matrix $\bar{X}$ guaranteed to contain X as

$$\begin{aligned} X_{i,j\ \perp} &= \left\lfloor \sum_k T^{-1}_{i,k} \cdot \begin{cases} Y_{k,j\ \perp}, & \text{if } T^{-1}_{i,k} \ge 0 \\ Y_{k,j\ \top}, & \text{if } T^{-1}_{i,k} < 0 \end{cases} \right\rfloor \\ X_{i,j\ \top} &= \left\lfloor \sum_k T^{-1}_{i,k} \cdot \begin{cases} Y_{k,j\ \top}, & \text{if } T^{-1}_{i,k} \ge 0 \\ Y_{k,j\ \perp}, & \text{if } T^{-1}_{i,k} < 0 \end{cases} \right\rfloor, \end{aligned} \tag{9}$$

[‡]IJG uses the Loeffler/Ligtenberg/Moschytz fast IDCT[9] by default.

and similarly for matrix post-multiplication $XT = \bar{Y}$. Because the elements of the matrix T^{-1} are fractions of very large integers, the multiplications in (9) require arbitrary-precision arithmetic over rational numbers. We also use the rules

$$\begin{aligned} \max\{\bar{x}, a\} = \bar{y} \implies [x_\perp, x_\top] &= \left[\begin{cases} -\infty, & \text{if } y_\perp \le a \\ y_\perp, & \text{otherwise} \end{cases}, y_\top \right] \\ \min\{\bar{x}, a\} = \bar{y} \implies [x_\perp, x_\top] &= \left[y_\perp, \begin{cases} \infty, & \text{if } y_\top \ge a \\ y_\top, & \text{otherwise} \end{cases} \right] \end{aligned} \tag{10}$$

2.4 Determining possible quality factors

The JPEG bit-stream header includes 8×8 quantization matrices for luma and chroma. Each matrix element specifies a divisor (quantization factor) for linear quantization of the corresponding coefficient in every transformed 8×8 block.

The recompressor DCT described in Section 2.3 outputs intervals for (dequantized) DCT coefficients, where each interval contains the result of a dequantization operation. For each coefficient position $(i, j) \in \{0, \ldots, 7\}^2$, we initially assume that any quantization factor between 1 and 255 is possible (baseline JPEG uses 8-bit precision for these values), then eliminate any factor which is inconsistent with the interval we obtain for the coefficient in any block. The set of possible quantization factors for the DCT coefficient position (i, j) in component $c \in \{\mathrm{Y}, \mathrm{Cb}, \mathrm{Cr}\}$ is given by

$$\begin{aligned} P^c_{i,j} = \big\{ q \in \{1, \ldots, 255\} : \nexists b.\, \big(\bar{W}^c_b\big)_{i,j} = [w_\perp, w_\top] \wedge w_\perp \operatorname{div} q = w_\top \operatorname{div} q \wedge \\ ((w_\perp < 0 \wedge q \nmid w_\top) \vee (w_\perp \ge 0 \wedge q \nmid w_\perp)) \big\} \end{aligned} \tag{11}$$

where $\bar{W}^c_b$ denotes the b-th block of intervals in component c and

$$x \operatorname{div} q = \begin{cases} \lfloor x/q \rfloor & x \ge 0 \\ \lceil x/q \rceil & x < 0. \end{cases} \tag{12}$$

In our implementation, the set of possible quantization factors is stored efficiently in two 8×8 tables of 256-bit masks, which describe the set of quantization factors possible at each position in each quantization matrix.

In the IJG compression utility, the quantization matrices are selected based on a quality factor $f \in \{1, \ldots, 100\}$ which selects what scaling factor will be applied to the matrices suggested by Annex K of the JPEG standard[7]. Therefore, when the user specifies a quality factor, only 100 quantization-matrix pairs are possible. Given the possible quantization factors at each coefficient position, the set of possible quality factors is given by

$$Q = \left\{ f \in \{1, \ldots, 100\} : \forall (i, j) \in \{0, \ldots, 7\}^2.\, \big(\mathrm{Q}^{\mathrm{Y}}_f\big)_{i,j} \in P^{\mathrm{Y}}_{i,j} \wedge \big(\mathrm{Q}^{\mathrm{C}}_f\big)_{i,j} \in P^{\mathrm{Cb}}_{i,j} \wedge \big(\mathrm{Q}^{\mathrm{C}}_f\big)_{i,j} \in P^{\mathrm{Cr}}_{i,j} \right\}, \tag{13}$$

where $\mathrm{Q}^{\mathrm{Y}}_f$ and $\mathrm{Q}^{\mathrm{C}}_f$ are the luma and chroma quantization matrices associated with quality factor f, respectively.

2.5 Quantization and exhaustive search

If an image was compressed with quality factor f, the set of possible quality factors in (13) will include f and also higher quality factors whose quantization bins are a superset of those associated with quality factor f.[§]

In addition, our uncertainty in the DCT coefficient values may allow quality factors lower than f which also give quantization bins that lie within all DCT coefficient intervals, but where the refinement (8) yields an empty set for one or more blocks. In this case, we remove f from the set of possible quality factors and try the next higher candidate for f.

[§]For example, quality factor 100 implies no quantization, which means all values $s \in \{-1024, \ldots, 1023\}$ are possible for each DCT coefficient.

For each block number b in component c, the quantized intervals with quality factor f are given by

$$\begin{aligned}
\left(\hat{\bar{W}}_b^c\right)_{i,j} &= [\hat{w}_\perp, \hat{w}_\top] \\
\hat{w}_\perp &= \begin{cases} w_\perp \operatorname{div} \left(\mathrm{Q}_f^c\right)_{i,j} & w_\perp < 0 \vee \left(\mathrm{Q}_f^c\right)_{i,j} \mid w_\perp \\ \left(w_\perp \operatorname{div} \left(\mathrm{Q}_f^c\right)_{i,j}\right) + 1 & \text{otherwise} \end{cases} \\
\hat{w}_\top &= \begin{cases} \left(w_\perp \operatorname{div} \left(\mathrm{Q}_f^c\right)_{i,j}\right) + 1 & w_\top < 0 \vee \left(\mathrm{Q}_f^c\right)_{i,j} \mid w_\top \\ w_\perp \operatorname{div} \left(\mathrm{Q}_f^c\right)_{i,j} & \text{otherwise} \end{cases}
\end{aligned} \tag{14}$$

The number of possibilities for a given quantized block is given by

$$\prod_{i,j} \left| \left(\hat{\bar{W}}_b^c\right)_{i,j} \right| \tag{15}$$

where $|\bar{w}|$ denotes the number of integers in the interval $\bar{w}$.

2.5.1 IDCT refinement

Based on running our recompressor on many images with different quality factors, we found that the number of possibilities for most blocks allows for an exhaustive search when the quality factor is less than about 90. For higher quality factors, the intervals output by the IDCT are normally sufficiently large that a search is infeasible. In our implementation we set the limit on the maximum allowable possibilities to $L = 2^{20}$ block values[¶].

If the number of possibilities for a block in (15) is less than the threshold L, for each possible value of the block we dequantize the quantized coefficients, perform an IDCT according to (7), and check whether outputs lie within the intervals $\bar{w}_b^c$. We approximate the set of blocks that meet this condition by an 8×8 interval matrix. If exactly one value for the block meets the condition, the block is marked 'exact'. If more than one value meets the condition, the block is marked 'ambiguous' and passes to the next stage of the search.

If the number of possibilities for a block is greater than L, we mark the block as infeasible to search.

$$\left(\hat{\bar{W}}_b'^c\right)_{i,j} \subseteq \left(\hat{\bar{W}}_b^c\right)_{i,j} \tag{16}$$

2.5.2 Impossible blocks

If the IDCT refinement stage finds that none of the possibilities for a block are consistent with the unquantized intervals $\bar{w}_b^c$, the block is marked 'impossible'. This indicates that the block was not output by the IJG decompressor, so we assume that the chosen quality factor was incorrect, and return to the quantization stage with the next quality factor in (13).

2.5.3 Full decompression refinement of 'ambiguous' blocks

Blocks which were marked as 'ambiguous' and are co-located with exact blocks in the other colour channels are decompressed to the RGB representation and checked against the uncompressed input data. We tighten the intervals to bound those values which decompress to the original input data. If exactly one value remains, the block is marked as 'exact'.

$$\left(\hat{\bar{W}}_b''^c\right)_{i,j} \subseteq \left(\hat{\bar{W}}_b'^c\right)_{i,j} \tag{17}$$

After this step, remaining ambiguous blocks represent multiple possible bit-streams which decompress to the same image.

[¶]This gives a worst case search with a roughly comparable number of IDCTs as required to decompress one second of 720p60 high definition video in a block/DCT-based video codec such as MPEG-2.

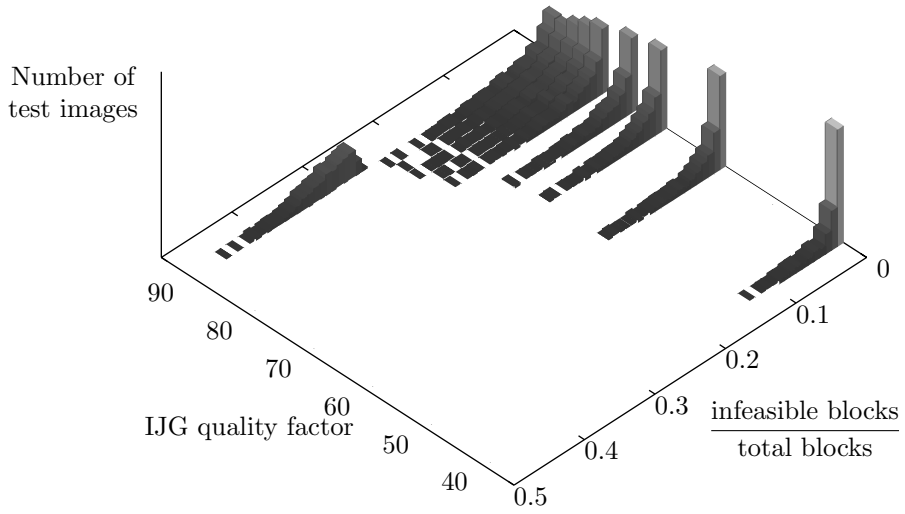

Figure 3. Recompression performance for a dataset of uncompressed images from the UCID.[10] 1338 colour images were compressed and decompressed at quality factors $f \in \{40, 60, 70, 80, 82, 84, 86, 88, 90\}$, then recompressed. The histogram shows the proportion of infeasible blocks in each image, with zero indicating that the image was recompressed perfectly.

2.6 Colour-space conversion refinement

The colour-space conversion is the first source of uncertainty in the exact recompressor, as we derive sets of $\mathrm{YC_bC_r}$ values for each pixel in the original image. By propagating back the spatial domain values from the IDCT refinement and up-sampling the chroma as in the decompressor using (2), we find which $\mathrm{YC_bC_r}$ triples are not consistent with the refined values, remove these from the sets and run another iteration of the recompressor beginning with chroma downsampling. We repeat this process until no further refinement of the $\mathrm{YC_bC_r}$ triples is possible.

3. RESULTS

We tested our recompressor on 1338 images from the uncompressed image database UCID.[10] Using the IJG command-line utilities `cjpeg` and `djpeg` for compression and decompression, we performed one compression cycle on each image using each of the quality factors $f \in \{40, 60, 70, 75, 80, 82, 84, 86, 88, 90\}$, giving 13380 test images.

As the test images are the result of a decompression operation using the IJG decompressor, no blocks were classified as 'impossible' by our recompressor, meaning that all blocks in the images are either exactly recompressed, ambiguous (meaning that multiple JPEG bit-streams produce the input on decompression) or infeasible to search.

We ran our recompressor on each image, calculating the proportion of blocks in the image which were infeasible to search. Figure 3 shows that, at lower quality factors, there are almost no blocks which are infeasible to search, whereas at quality factors higher than 85, the number of infeasible blocks sharply increases. At quality factor 90, more than a quarter of all blocks were infeasible to search.

Images that were previously compressed at high quality factors are difficult to recompress because the quantization step does not reduce the number of possibilities sufficiently to allow a search. At lower quality factors (i.e., higher quantization factors) only a small number of possibilities normally remain inside the intervals resulting from IDCT reversal. In cases where the quality factor is low, infeasible blocks are generally caused by saturated

values on inputs to the IDCT reversal (i.e., spatial domain samples equal to 0 or 255) which must be mapped to large intervals as the last operation of the IDCT is to clip the range.

Our exact recompressor is able to match or outperform naïve recompression in all cases, as infeasible blocks can be replaced with naïvely recompressed blocks, and other blocks are either ambiguous or exact, so result in an identical image block to the co-located input block on decompression.

REFERENCES

[1] Salazar, C. and Tran, T., "A complexity scalable universal DCT domain image resizing algorithm," *Circuits and Systems for Video Technology, IEEE Transactions on* **17**, 495–499 (April 2007).

[2] Neelamani, R., de Queiroz, R. L., Fan, Z., and Baraniuk, R., "JPEG compression history estimation for color images," *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on* **3**, III–245–8 vol.2 (September 2003).

[3] Fridrich, J., Goljan, M., and Du, R., "Steganalysis based on JPEG compatibility," *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **4518**, 275–280 (2001).

[4] Fan, Z. and de Queiroz, R. L., "Identification of bitmap compression history: JPEG detection and quantizer estimation," *Image Processing, IEEE Transactions on* **12**, 230–235 (February 2003).

[5] Better JPEG, "Lossless resave plug-in for Adobe Photoshop and Adobe Photoshop Elements," *http://www.betterjpeg.com/jpeg-plug-in.htm* .

[6] Popescu, A. C. and Farid, H., "Statistical tools for digital forensics," in [*6th International Workshop on Information Hiding*], 128–147, Springer-Verlag, Berlin-Heidelberg (2004).

[7] "ISO/IEC 10918-1: Digital compression and coding of continuous-tone still images (JPEG)," *CCITT Recommendation T* **81**.

[8] Lane, T. G., "Independent JPEG Group library," *http://www.ijg.org* .

[9] Loeffler, C., Ligtenberg, A., and Moschytz, G. S., "Practical fast 1-D DCT algorithms with 11 multiplications," *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on* , 988–991 vol.2 (May 1989).

[10] Schaefer, G. and Stich, M., "UCID – an uncompressed colour image database," *Storage and Retrieval Methods and Applications for Multimedia 2004* **5307**, 472–480.

APPENDIX A. IJG INVERSE DISCRETE COSINE TRANSFORM

$$T = \begin{bmatrix} 8192 & 11363 & 10703 & 9633 & 8192 & 6437 & 4433 & 2260 \\ 8192 & 9633 & 4433 & -2259 & -8192 & -11362 & -10704 & -6436 \\ 8192 & 6437 & -4433 & -11362 & -8192 & 2261 & 10704 & 9633 \\ 8192 & 2260 & -10703 & -6436 & 8192 & 9633 & -4433 & -11363 \\ 8192 & -2260 & -10703 & 6436 & 8192 & -9633 & -4433 & 11363 \\ 8192 & -6437 & -4433 & 11362 & -8192 & -2261 & 10704 & -9633 \\ 8192 & -9633 & 4433 & 2259 & -8192 & 11362 & -10704 & 6436 \\ 8192 & -11363 & 10703 & -9633 & 8192 & -6437 & 4433 & -2260 \end{bmatrix} \tag{18}$$