Overcompressing JPEG images with Evolution Algorithms

Jacques Lévy Véhel[1], Franklin Mendivil[2] and Evelyne Lutton[1]

[1] Inria, Complex Team, 78153 Le Chesnay, France
jacques.levy-vehel@inria.fr,evelyne.lutton@inria.fr,
[2] University of Acadia, Department of Mathematics and Statistics,
Wolfville, Nova Scotia, Canada, B4P 2R6
franklin.mendivil@acadiau.ca>

Abstract. Overcompression is the process of post-processing compressed images to gain either further size reduction or improved quality. This is made possible by the fact that the set of all "reasonable" images has a sparse structure. In this work, we apply this idea to the overcompression of JPEG images: We reduce the blocking artifacts commonly seen in JPEG images by allowing the low frequency coefficients of the DCT to vary slightly. Evolutionary strategies are used in order to guide the modification of the coefficients towards a smoother image.

1 Statement of the problem

Various compression methods have been devised in view of reducing the size of image files for purposes of storing and transmission. In order to reach substantial compression rates, lossy techniques have to be used, *i.e.* the compressed/decompressed image is a degraded version of the original one. Most such compression methods allow to tune the size of the compressed file, so as to reach a trade-off between reduction and quality.

Many people have realized the following fact: The set of all "reasonable" images is extremely small as compared to the one of all "possible" images. Although the term "reasonable" is vague, the meaning is clear: If one chooses at random the gray level values of all pixels in a $N \times N$ image, where, say, $N = 512$ and the gray levels are coded on 8 bits, then the probability that the result looks like a meaningful image is ridiculously small. One may wonder if it is possible to improve the efficiency of the various compression methods by using this remark. We term attempts of this kind *overcompression*. Overcompression is not a new compression method, nor is it specific to a given compression scheme. Rather, overcompression tries either to further reducing the size of the compressed file or to improving of the quality of the decoded image, by taking advantage of the sparse structure of the set of images.

Although overcompression is by no means an easy task, it may be approached by a variety of methods. In this work, we propose an overcompression scheme for the case of JPEG compressed images.

The JPEG compression format [1] is the most popular image compression method to date. It has served as a standard until recently. Although JPEG has now been surpassed by a new standard, called JPEG 2000 [1], it is still widely used for several reasons. A major one is that a huge number of images are stored in this format, and it does not seem feasible nor desirable to acquire again all these data in order to process them with the new format. One may also mention the fact that while JPEG is public domain, JPEG 2000 uses some patented techniques, which reduces its diffusion.

It thus seems desirable to increase the efficiency of JPEG. As explained below, our overcompression method improves on the quality of JPEG images by reducing the blocking artifacts classically encountered with this compression method.

Several methods have already been proposed in view of post-processing JPEG-compressed images in order to remove the blocking artifacts. See [2] for a comprehensive list of references.

2 Overcompressing JPEG images

The general problem of restoring fidelity in a degraded image is almost impossibly difficult. However, since we are assuming that our image is a compressed JPEG image, we know how the information is lost. Our method is particularly adapted to the case of medium to high compression ratios, which translates into noticeable blocking effects.

The basic JPEG compression algorithm decomposes an image into 8×8 non-overlapping blocks and treats each such block independently of all others. The DCT (Discrete Cosine Transform) of each block is computed and then these frequency coefficients are quantized. Since the details of this quantization are important for our methods, we discuss them further below. After the quantization, the 8×8 table of coefficients is linearly ordered using a zig-zag traversal of the array, run length encoded and then finally some type of entropy coding is applied.

The 64 frequency coefficients are quantized individually by dividing by a *quantization value* and then rounding to the nearest integer. There are two important points to notice about this process. The first is that many different DCT values become quantized to the same integer value. The second is that these "equivalent" DCT values lie in an interval whose length is the size of the corresponding quantization value. Thus the higher this quantization value, the wider the corresponding interval (and the more information which is lost in the process).

Because of this information loss, many initial images all lead to the same final JPEG image (all their differences being removed by the quantization). We call this a *JPEG equivalence class* of images. Given a compressed JPEG image, clearly both the initial image and the JPEG image are in this equivalence class (as are many others).

Table 1. Example of *quantization values* for JPEG

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Our basic idea is to move around within this equivalence class to remove the blocking artifacts. The fact that the original image lies within this class ensures that there are some images which are visually better than the compressed one.

Searching within an equivalence class mearly necessitates perturbing the given JPEG DCT coefficients in such a way that each value remains in the given interval. This is simple to ensure by constraining the size of the perturbations according to their frequency component.

It is important to mention that we assume that the JPEG is reasonably close to the image. We cannot recover lost information, we only try to smooth out the blocking artifacts at the block boundaries without oversmoothing the entire image.

Each block is represented in the DCT domain by a matrix of 8×8 coefficients, where the frequency of coefficient (i, j) increases in the x (resp. y direction) direction as i (resp. j) increases. Thus, low-frequency (resp. high-frequency) coefficients are the ones with low (resp. high) value of $i + j$. Since we are only interested in smoothing the blocking artifacts, and not in recovering the lost high-frequency content, we need only modify the low-frequency coefficients. Experiments showed that tuning the 6 coefficients corresponding to the lowest frequency components (*i.e.* the ones with $i + j \leq 4$ - recall that i and j takes values in $\{1, \ldots, 8\}$) allowed to reach our purpose. More precisely, we ran the following test: We replaced these coefficients in highly compressed JPEG images with the "correct" coefficients from the uncompressed images. This resulted in images which were almost indistinguishable from the non-compressed ones. As a consequence, correcting these coefficients should be sufficient to restore most of the image quality.

We thus need to optimize 6 parameters per 8×8 block. For a 256×256 image, this still means processing 6144 values. This is beyond the capacity of any reasonable algorithm. However, we can take advantage of the following remark: JPEG ignores inter-block dependence. As a matter of fact, this is precisely the reason why blocking artifacts arise. This can be exploited in a very simple way, resulting in an algorithm which can easily be parallelized (we did not try this explicitly): Our algorithms decompose the image into a collection of overlapping "tiles", which are processed independently. The results for all the tiles are then blended together by using a convex combination of the various tiles at all places where the tiles overlapped.

This results in huge savings in computational effort and thus much better solutions: For a 256×256 image, using 24×24 tiles and shifting by 8 pixels in each direction results in 900 tiles, so 900 optimizations with 54 parameters each. This contrasts to a single optimization with 6144 parameters. The value 24 is justified by the fact that, with this size, each block is processed once along with its 8 neighbors, allowing an efficient treatment of the blocking effects.

The optimizations on each tile are very fast and result in good local solutions. The blended global solution was found to be much better than any of the solutions we obtained to the global problem, even with higher iteration counts.

Even though the solutions obtained this way are probably sub-optimal, the justification is that the necessary adjustment to the DCT coefficients in disjoint tiles are approximately independent, with this approximation becoming more true as the distance between the tiles increases.

3 Fitness Function

As always, the choice of the fitness function is of the utmost importance. This is particularly crucial in our case, because there is no obvious way to measure the adequacy of a solution.

Since the aim is to reduce the blocking artifacts of a highly compressed JPEG image, the fitness function should provide a quantitative measurement of these artifacts. There are many possible ways of trying to do this, and we discuss several that were tried, along with some which were tried but then discarded.

As previously mentioned, both the original and the JPEG image are in the given equivalence class. The fitness function should obviously yield a smaller (more optimal) value on the original image than on the JPEG image (since we wish for images which look more like the original image than the JPEG image). This simple criterion eliminates several candidate fitness functions. In particular, one might be tempted to use fitness functions that provide a measure of the smoothness of the image, with the idea that a highly compressed image will not be as smooth as the original image, due to the blocking artifacts. One such metric is the *total variation norm* [3], a commonly used metric in image processing. However, this metric is rather ill-adapted to our case: Indeed, a highly compressed JPEG image tends to be almost piecewise constant, because it consists of large almost flat regions bounded by a comparatively small number of edges between blocks. This is precisely the type of images that is favored by total variation. As a consequence, compressed images will often tend to have smaller fitness values than their original version.

The second type of fitness function is one which explicitly uses the 8×8 blocking structure of JPEG. There many possibilities for this as well. Among those tried were:

1. Summing the absolute differences across all the interior 8×8 block boundaries.

2. For each interface between two blocks, compute the average value on both sides of the edge and sum the absolute value of these differences.
3. Same as 2) except normalize by some block measure (like the mean).
4. For each interface between two blocks, compute the vector of differences along the edge and accumulate the L1 norm of these vector differences divided by the variance of the vector.

In each case, the idea is that one wishes to penalize large inter-block differences but allow for the fact that blocks may consist of texture regions.

4 Evolutionary Algorithms

To find optimal perturbation of the JPEG coefficients, we tested two algorithms: a $(1+1)$ EA and a genetic algorithm.

If we write the JPEG image as a sum over the blocks, $I = \sum_B I_B$, we can represent the individual as a sum of perturbations, $\delta = \sum_B \delta_B$. We are looking for the best δ, that is so that $I + \delta$ has smoothed out the blocking artifacts. Thus each $\delta_B \in \mathbb{R}^6$, with each component appropriately constrained (by the condition that $I_B + \delta_B$ still lie in the correct quantization interval).

Modifying a component consists of adding a random perturbation (uniformly distributed in an interval) to that component, wrapping around if the new value lies outside the particular quantization interval. We use wrap-around rather than clipping since the former ensures a uniform distribution over the interval (experiments showed that clipping yields lower quality results).

After testing various possibilities, the fitness function used was item 1) from the list above, simply summing the absolute differences across all (internal) block boundaries. Because only the lowest frequency coefficients were modified, the resulting images were not too irregular: Since we do not touch the coefficients (i, j) such that $i + j > 4$, we have no risk of introducing spurious higher frequencies, and there is no need to introduce a term measuring this in the fitness function.

4.1 $(1+1)$ EA

We tried a simple $(1+1)$ EA, that is, each generation consists of a single individual which produces one mutation which may become the next generation.

We mutate δ by independently modifying each component with some probability p_c, obtaining δ'.

If $f(\delta') < f(\delta)$, we always replace δ with δ'. On the other hand, with probability p_r we replace δ with δ' even if $f(\delta) < f(\delta')$.

4.2 Genetic Algorithm

A classical genetic algorithm was used, with the following parameters: uniform selection, elite count = 2, crossover fraction = 0.8, uniform mutation with probability = 0.1 and range two thirds of the quantization bin (*i.e.* the size of the

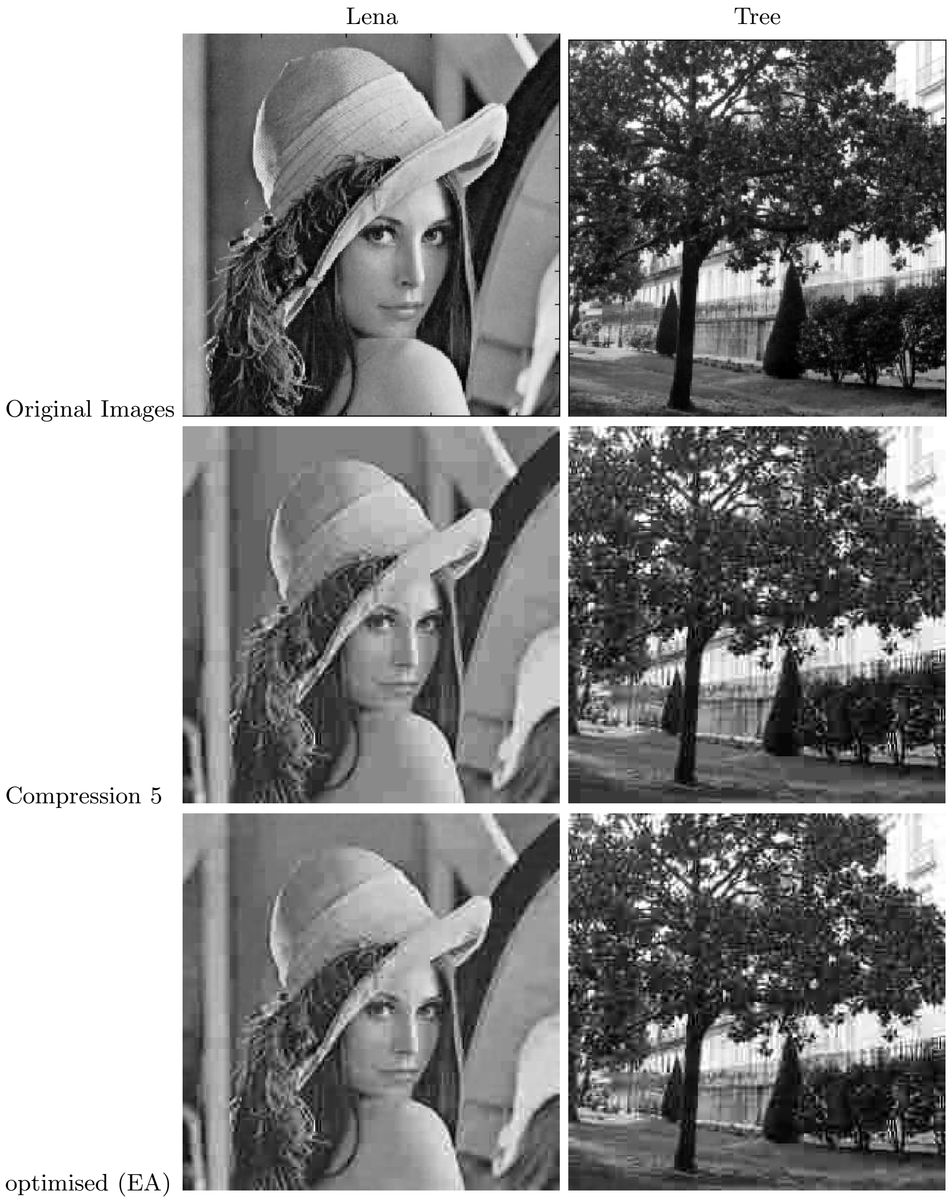

Fig. 1. Compression results, compression factor 5

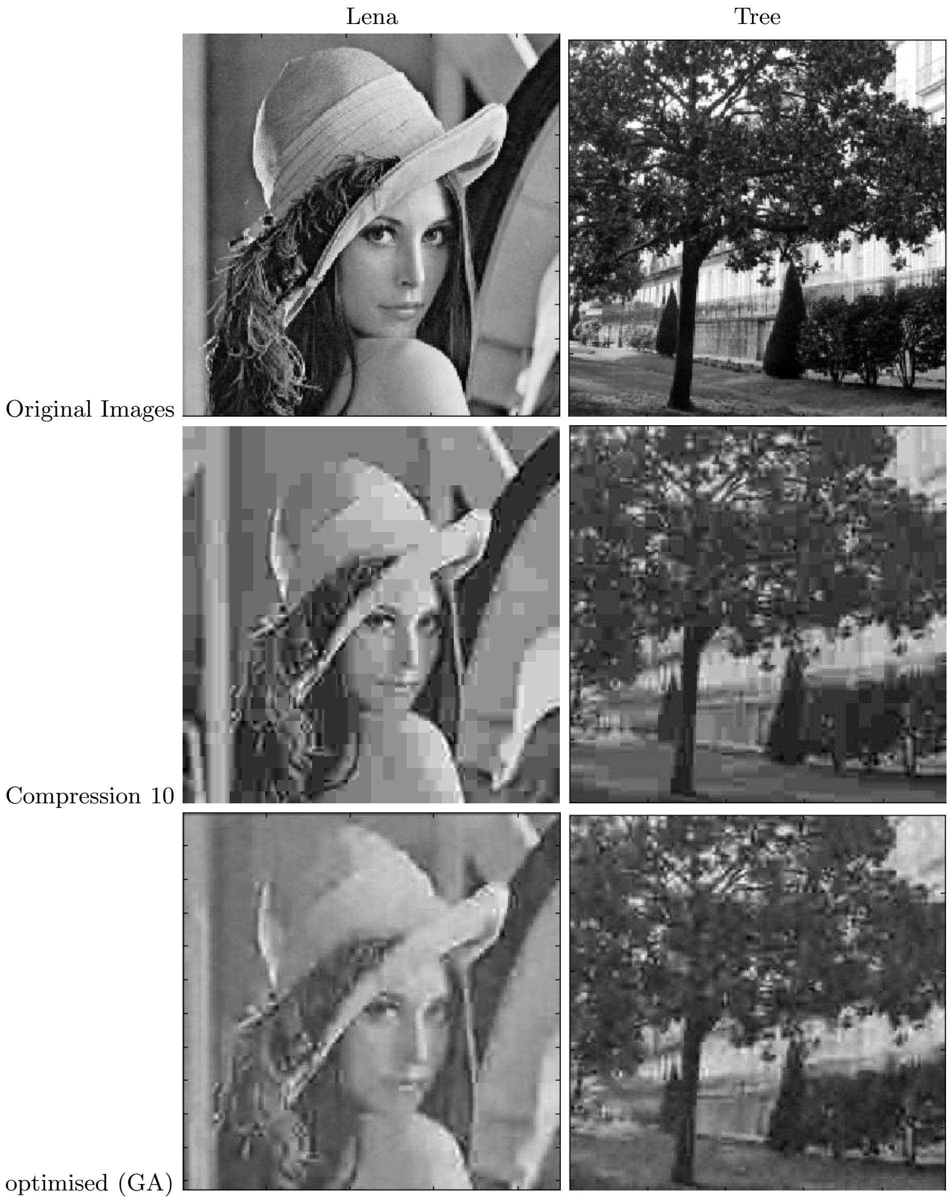

Fig. 2. Compression results, compression factor 10

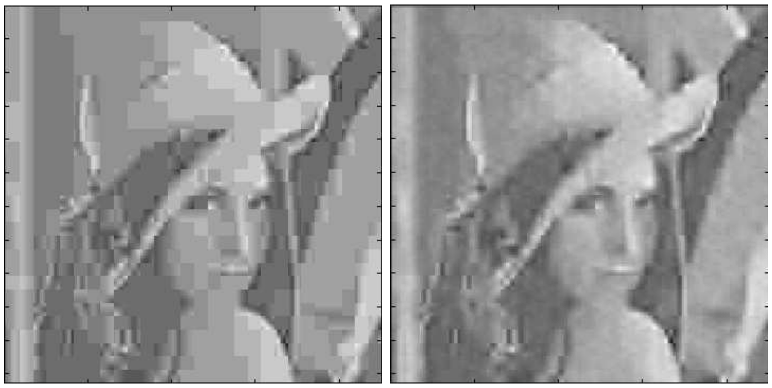

Fig. 3. Lena with compression factor 15 (left), optimized with the GA (rigth)

interval where the uniform perturbation is drawn depends on the quantization bin, and is equal to two thirds of the length of this bin), population size = 40, number of generations = 200.

5 Experimental results

We display in figures 1 to 3 results on two images: The well-known lena image, and an image of a tree. Both images are 256×256 with the grey levels coded on 8 bits. We show optimization with both the $(1+1)$ EA and the genetic algorithm. Three compression ratios have been considered: The compressed images are obtained by using the quantization values in table 1 multiplied by 5, 10, and 15.

References

1. Home site of the JPEG and JBIG committees: http://www.jpeg.org/
2. Nosratinia, A. *Enhancement of JPEG-Compressed images by re-application of JPEG*, Journal of VLSI Signal Processing, **27**, (2001), 69-79.
3. Chan, T.F., Shen, J., Zhou, H-M. *Total Variation Wavelet Inpainting*, Journal of Mathematical Imaging and Vision, **25**-1, (2006), 107–125.