This document is published in:

Expert Systems with Applications, (2011), 38 (9), 10999–11010.
DOI:http://dx.doi.org/10.1016/j.eswa.2011.02.143

MIJ2K Optimization using evolutionary multiobjective optimization algorithms

Alvaro Luis Bustamante *, José M. Molina López, Miguel A. Patricio

Univ. Carlos III de Madrid, Avda. Univ. Carlos III, 22, 28270 Colmenarejo, Madrid, Spain

* Corresponding author. Tel.: +34 918561338.

E-mail addresses: aluis@inf.uc3m.es (A.L. Bustamante), molina@ia.uc3m.es (J.M. Molina López), mpatrici@inf.uc3m.es (M.A. Patricio).

Abstract: This paper deals with the multiobjective definition of video compression and its optimization. The opti-mization will be done using NSGA-II, a well-tested and highly accurate algorithm with a high conver-gence speed developed for solving multiobjective problems. Video compression is defined as a problem including two competing objectives. We try to find a set of optimal, so-called Pareto-optimal solutions, instead of a single optimal solution. The two competing objectives are quality and compression ratio maximization. The optimization will be achieved using a new patent pending codec, called MIJ2K, also outlined in this paper. Video will be compressed with the MIJ2K codec applied to some classical vid-eos used for performance measurement, selected from the Xiph.org Foundation repository. The result of the optimization will be a set of near-optimal encoder parameters. We also present the convergence of NSGA-II with different encoder parameters and discuss the suitability of MOEAs as opposed to classical search-based techniques in this field.

Keywords: Multi-objective, Optimization, Video, Encoder

1. Introduction

Nowadays, digital video is widely used for many purposes, ranging from mere entertainment, such as TV, video conferencing or video on demand, to more professional environments, such as remote video surveillance. This wide range of applications is possible thanks to recent advances in digital video technology, like broadband connections to the Internet, computing capacity and the digital storage space of new devices.

However, the most important part of a digital video system is the codec. The codec enables digital video compression (with the encoder) and/or decompression (with the decoder). This reduces the data necessary for representation. This is necessary because uncompressed digital video still exceeds common network bandwidths for transmission and storage spaces for digital archiving.

Compression usually employs lossy data compression. Lossy data compression reduces a file by permanently eliminating certain information, especially redundant information. When the file is uncompressed, only a part of the original information is still there (although this may go unnoticed to the user, especially in video and sound compressions). It inherently implies a reduction of the quality in exchange for a reduction in the final amount of information.

Systems like these introduce a complex trade-off between the quality and quantity of data needed to represent the video (also referred to as bit rate). The best we could expect is to get the highest video quality with the smallest file size, but these objectives are in conflict since better qualities inherently imply a greater bit rate.

Thus, when compressing digital video, the user usually establishes encoder objectives or constraints, i.e. the maximum bit rate (normally used when there are bandwidth limits or storage space constraints (Jiang, 2006; Wang & Leou, 2003)), or video quality (rated on a 0 to 100 scale by some quality metric, etc., to ensure some quality of service (Ng, Leung, & Hui, 2005; Zhang, Zhu, & Ya-W, 2005)).

The encoder should compress the digital video according to these objectives, but it is not usually easy to get a direct correlation between these high-level objectives and low-level encoder parameters, because video encoder bit rate and video quality depend on several coding parameters, such as quantization parameter (Czuni, Csaszar, & Licsar, 2006), coding mode (kuang Chen, Vetro, & Sun, 1997), macroblock sizes (Tu, Yang, Shen, & Sun, 2003), or motion compensation algorithms (Hang, Chou, & Cheng, 1997). Each of these parameters has its own attributes or thresholds and may have different effects. We have developed MIJ2K, a new video codec, currently patent pending, and we need to optimize some parameters before its release.

In this paper, we present an algorithm to dynamically optimize the two conflicting objectives discussed above (video quality/bit rate) as well as translate the objectives to the video encoder parameters. Such algorithms are known as multiobjective optimization (MO) algorithms. MO optimization problems (Steuer, 1986; Sawaragi, Nakayama, & Tanino, 1985) are very common in many complex engineering situations, and can be found in many fields: product and process design, finance, aircraft design, the oil and gas industry, automobile design, or wherever optimal

decisions need to be taken in the presence of more than one, generally conflicting objective, preventing the simultaneous optimization of each objective. Basically there are two major approaches for solving such MO problems. The first is to combine the individual objective functions into a single composite function and optimize this function only. This could be done with techniques such as the weighted sum method (Koski, 1988), utility theory (Thurston et al., 2006), etc.

The second major approach is to determine an entire Pareto optimal solution set or a representative subset, i.e. a series of solutions that are non-dominated with respect each other. Pareto optimal solution sets are often preferred to single solutions because they can be practical when considering real-life problems and give the decision maker (DM) the option of evaluating the trade-offs between different solutions.

The use of Multiobjective Evolutionary Algorithms (MOEAs) to output the Pareto front between these two objective functions (quality and bit rate) will satisfy all DM requirements, since the Pareto solution set will offer a wide range of solutions ranging from low quality, low bit rate to high quality, high bit rate, all of which are optimal in some sense. DMs could use the solution set for many applications with different constraints, such as real-time streaming, controlling the bit rate according to available bandwidth; high definition video, establishing high qualities; video storage with space constraints, etc.

We will use the NSGA-II algorithm to obtain the Pareto front (Deb, Pratap, Agarwal, & Meyarivan, 2002). NSGA-II should maximize two objective functions: the quality measured by the peak-to-signal noise ratio (PSNR), and the compression ratio (CR), which is a ratio between compressed video and original video sizes. To evaluate each set of parameters, we will test the video compressor with classical sequences used to evaluate encoders performance, like 'Hall Monitor' or 'Akiyo', selected from the well-known Xiph.org Foundation repository (Lora, 1994-2008), which is suitable for evaluating video compression codecs.

Achieving compression and decompression to meet the fitness function for the quality and bit rate objectives is a tedious process due to the amount of data that has to be managed. Classical search-based techniques will take a long time, whereas MOEAs are well suited for such optimization problems.

The paper is organized as follows. Section 2 gives a general description of how the MIJ2K works. Section 3 defines the multiobjective problem. Section 4 specifies the conflicting objective functions and the decision variables for optimization. Finally, Section 5 presents the tests performed with the NSGA-II algorithm.

2. MIJ2K codec

This section outlines the MIJ2K codec to give an understanding of how it works and the internal parameters needed to use evolutionary algorithms for optimization purposes. The basics of video compression rely on two major methods: intra-frame and inter-frame compression techniques (Shi & Sun, 2000). In intra-frame methods each video frame is an independent entity encoded with a still image compressor, usually JPEG (Pennebaker & Mitchell, 1993; Wallace, 1991) or JPEG2000 (Christopoulos, Skodras, & Ebrahimi, 2000; Rabbani & Joshi, 2002). This technique is extre-mely useful in real-time environments, like video surveillance, due to its low computing complexity. On the other hand, inter-frame techniques employ advanced coding methods, using algo-rithms like motion compensation. These techniques are more bandwidth-efficient than intra-frame methods at the expense of more complexity.

In this paper, we optimize the internal parameters of the MIJ2K codec, which is based on the JPEG2000 image compression standard, and a mixture of intra and inter-frame techniques. JPEG2000 is a wavelet-based (Adams & Ward, 2001) image compression standard created by the Joint Photographic Experts Group committee in the year 2000, with the aim of superseding their original discrete cosine transform-based JPEG standard (dating from 1992).

JPEG2000 offers a modest increase in compression performance compared with JPEG, but its main benefit is significant code-stream flexibility. The code stream obtained after compression of an image with JPEG2000 is scalable, meaning that it can be decoded in a number of ways. For instance, by truncating the code stream at any point, we can get a representation of the image at a lower resolution or signal-to-noise ratio. By ordering the code stream in various ways, applications can achieve significant performance increases (Adams, 2001).

Apart from the above features, the main benefit of using JPEG2000 for video streaming is that, unlike other video compressors including MPEG-4, compression could be done in real-time (Luis & Patricio, 2000) because it is an intra-frame codec. Thanks to this feature, JPEG2000 can be used in events that require real-time transmission, like video surveillance.

As far as our proposal is concerned, however, the main advantage is that the image can, optionally, be partitioned into smaller independent non-overlapped rectangular blocks called *tiles* (ISO/IEC, 2000). We will exploit this exceptional feature, provided by this compressor alone, to perform real-time inter-frame compression using the proposed conditional tile replenishment method, which we optimize in this paper. We will employ a new block-based difference coding technique (Shi & Sun, 2000) for this task, having tiles assume the role of blocks.

Tiles can be of any size, and the whole image can even be considered as one single tile. Once the size has been chosen, though, all the tiles will be of the same size (except, optionally, tiles on the right and bottom borders). Dividing the image into tiles is advantageous in that the encoder/decoder will need less memory to encode/decode the image. Also it can opt to encode/decode only selected tiles to achieve a partial coding/decoding of the image. It will provide full control of whatever area of the image is being compressed, decompressed, transmitted, etc.

Fig. 1 shows an example of how the J2K code stream is structured, and how an image is divided using tiles. The first marker present is the start of code stream (SOC). This is followed by a main header (MH), which includes the common parameters required for image decoding. The tile-part header (TH) contains the necessary information for decoding each tile. It is followed by the corresponding tile-part bit-stream. Finally, the end of code-stream (EOC) marker denotes the termination of a J2K code stream.

Notice that each region of the image occupies a definite region in the J2K code stream. Thanks to the header definition method (ISO/IEC, 2000), each region can also be accessed randomly.

The inter-frame technique adopted in MIJ2K is a real-time specific block-based difference coding, adapted to JPEG2000 streams. This technique is useful in the real-time transmission scheme because it provides a low computational complexity. It also preserves the real-time latency provided by the native JPEG2000 intra-frame architecture.

The MIJ2K architecture designed for this task is outlined in Fig. 2 and explained in more detail in the following sections. The general operating procedure is as follows.

A common JPEG2000-based streaming system is basically divided into three steps. The first step is related to frame acquisition and compression. It is followed by the transmission of the resulting compressed frames. Finally, it ends with the reception and display of each frame. The acquisition and compression step in these systems is not complex. Each frame is acquired and compressed separately, and can be transmitted as soon as it is compressed.

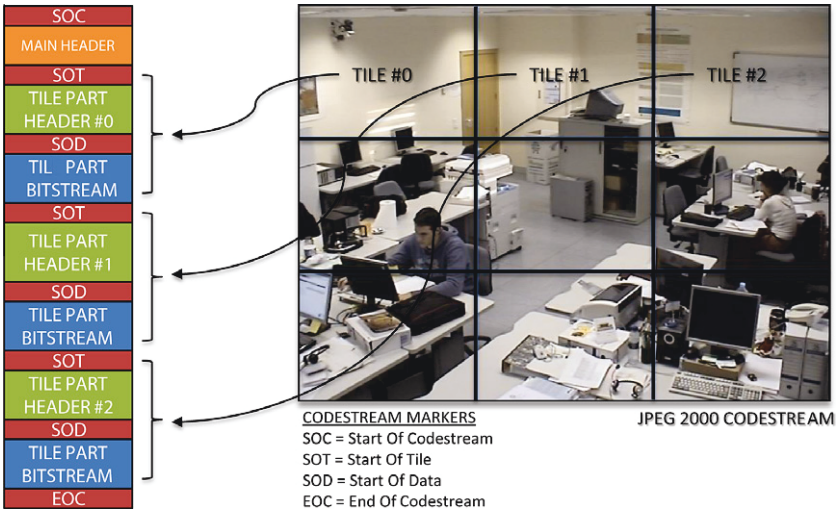


Fig. 1. JPEG2000 code-stream structure.

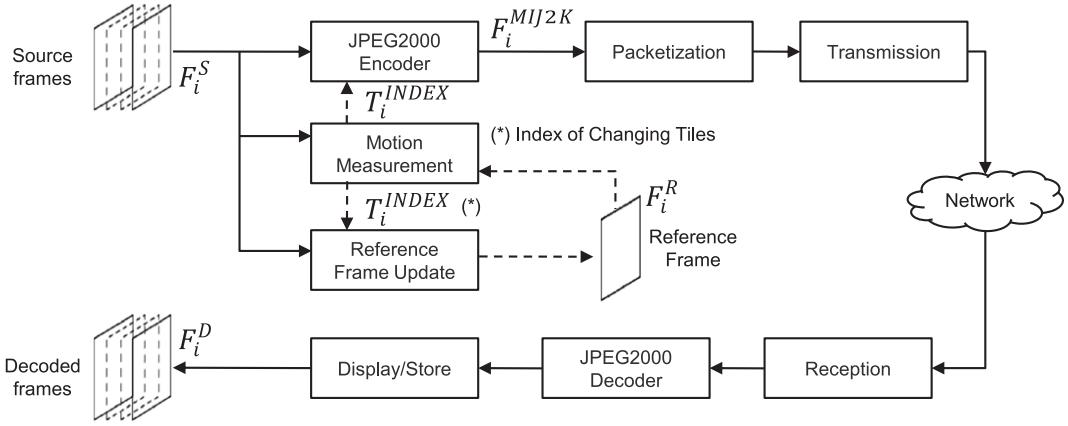


Fig. 2. Overall functioning of the MIJ2K architecture, performing real-time selective tile compression and transmission.

In the proposed MIJ2K streaming architecture, an extra process is inserted in the compression step. Instead of compressing each whole frame, it compresses and transmits only the areas that are different (changing tiles) from the previously transmitted frame. Compressing only changing tiles improves compression, transmission, and decoding performance, since it hugely reduces the total amount of data for management.

For example, Fig. 3 shows the tiles detected as changing in frame 127 of the 'Akiyo' sequence. They are marked with a green rectangle.[1] Notice how only 17% of the tiles in this frame are detected as changing. When the MIJ2K method is applied to the whole 'Akiyo' sequence, it saves around 87% of bandwidth compared with a native JPEG2000 streaming system.

Changing tiles are detected using a reference frame F_i^R that stores a representation of the last transmitted frame. Each new frame F_i^S to be transmitted is compared tile by tile with the reference frame F_i^R in order to detect the tiles that differ with their counterparts. F_i^R is updated frame by frame with the tiles that are detected as 'changing' to create a real representation of what the client is viewing.

The client receiving this modified stream (a JPEG2000 code stream containing just some of the tiles) will have to decode each tile received and use it to update the displaying frame F_i^D. The client can easily locate each tile position since they are identified by a tile number. Knowing that all tiles (except image boundaries) are

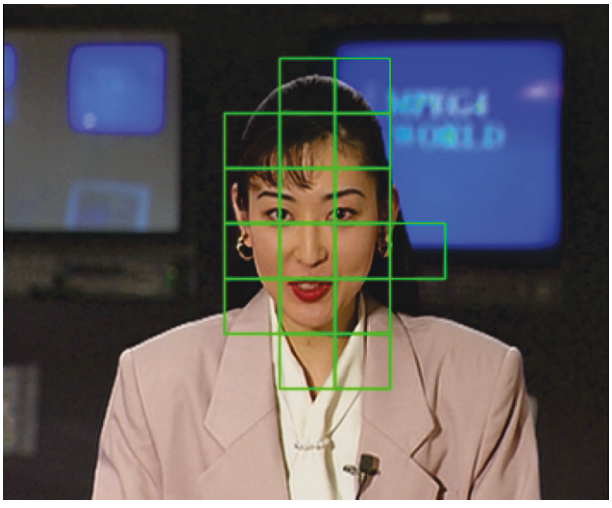


Fig. 3. 'Akiyo' sequence. Tiles detected as changing in frame 127. In this frame, a bandwidth saving of around 83% is achieved.

[1] For interpretation of colour in Fig. 3, the reader is referred to the web version of this article.

of the same size, it is easy to calculate the position of the tile inside the full image.

Some of the subsystems illustrated in Fig. 2 (necessary for understanding which parameters to optimize), and their design details are explained in more detail in the following sections.

2.1. JPEG2000 encoder

This module compresses the JPEG2000 still images. It basically takes and compresses the source frame F_i^S using tile partitioning. The encoder should be modified to assure that only the tiles specified by the T_i^{INDEX} parameter, and not all the tiles of the frame are compressed. This is feasible since the tiles in JPEG2000 images can be compressed and decompressed separately. Ideally, then, only the changing tiles are compressed. This will save compression time, improving overall system operation. All frames will be compressed with the same quality, given by bpp_i.

Fig. 4 is a diagram of this module. It shows all the required inputs and outputs. These are described in more detail in the following:

- Source frame F_i^S: this is the last image acquired by the digitizer board or digital camera, that is, the frame that is going to be transmitted. It should be formatted in some J2K encoder-understandable format, for instance, a RAW 8*bpp* or 24*bpp* RGB image (depending on whether it is a gray-scale or color picture).
- Quality bpp_i: this parameter is related to compression quantity, or target quality after each still image has been compressed. This parameter is usually expressed as the quantity of bits used to represent each pixel in the generated JPEG2000 code stream, that is, bits per pixel (*bpp*).
- Tile size T_{SIZE}: this parameter defines the size of the tiling performed in the J2K compressed image. Once the tile size has been set, all images will be compressed in separate squared regions of the same size. Tile size is variable, but, theoretically, small tile sizes can achieve a better fitting to moving objects. This is illustrated in Fig. 5, where the soccer player is a moving object in the sequence, and little tiles are a better fit for the player.
- Changing tiles index T_i^{INDEX}: this input is delivered by the motion measurement subsystem (this process is detailed later), as shown in Fig. 2. It indicates the index of the tiles that contain some movement, that is, the tiles that should be compressed. In this case, the JPEG2000 encoder should compress these tiles (regions) of the image only. This will improve the compression delay since the encoder does not have to compress the whole image.
- J2K code stream F_i^{MIJ2K}: this is the J2K code stream output after compression. It should contain only the changing tiles specified in the T_i^{INDEX} input. The tiles bit-stream should be between the main header and end of code-stream markers, as shown in Fig. 4. In this case, only tiles 3, 7 and 15 have been detected as changing. The F_i^{MIJ2K} output will be passed directly to the packetization subsystem, as illustrated in Fig. 2, which will transmit the frame over the network.

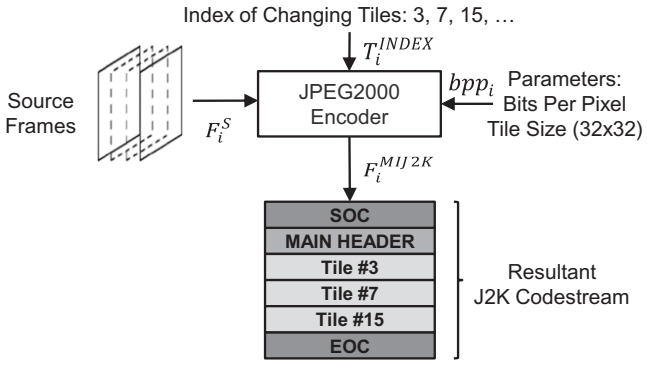


Fig. 4. MIJ2K encoder operation.

This process is defined as in Eq. (1), where the J2K function represents the encoder, and the inputs of the function are source frame F_i^S, index of tiles for compression T_i^{INDEX}, quality bpp_i and size of tiles T_{SIZE} with the default value 32×32.

$$F_i^{MIJ2K} = J2K\left(F_i^S, bpp_i, T_i^{INDEX}, T_{SIZE}\right) \tag{1}$$

Notice how the compression module introduces two variables for optimization: bpp_i and T_{SIZE}. *A priori* we do not know what is the best value for the two variables, since we do not know how quality or tile size affect the motion compensation algorithm and the final video output.

2.2. Motion measurement

This subsystem manages the motion measurement between two consecutive frames and detects the tile index of images that contain some movement. The complexity of the algorithm proposed for this task is low with a view to meeting the needs of real-time transmissions rather than aiming for the highly efficient motion detection performed by common inter-frame encoders. Such sophisticated techniques, as used in H.264/MPEG-4 AVC, are not usually feasible for critical real-time environments, such as video surveillance.

This algorithm, as shown in Fig. 6, takes two frames, F_i^S and F_i^R (source and reference), for comparison. It also has to know the selected tile size T_{SIZE} used in the compression step.

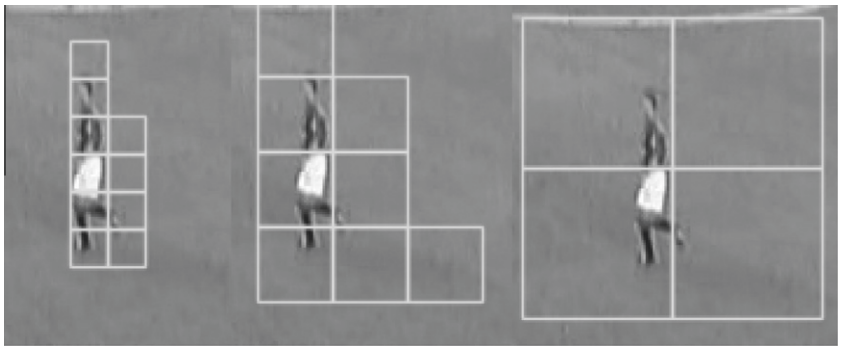


Fig. 5. Overlay areas with different tile sizes. From left to right 16×16, 32×32 and 64×64.

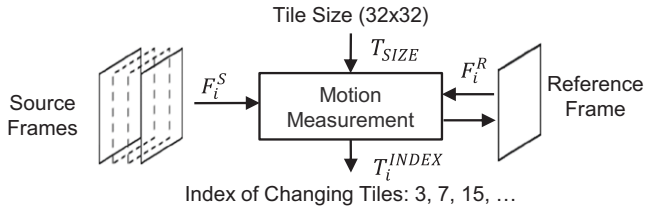


Fig. 6. Motion measurement input/output.

All the inputs and outputs of this system are described below:

- Source frames F_i^S and F_i^R: F_i^S should be the same image as used in the J2K encoder in some affordable format where the pixel values of the image can be directly manipulated. The other required image is the reference frame F_i^R. For comparison purposes, it should have the same format as the source frame F_i^S.
- Tile size T_{SIZE}: as compression is performed using the concept of tiling, motion should be measured in tile units. So, this subsystem must know the working tile size T_{SIZE}.
- Changing tiles index T_i^{INDEX}: as mentioned in the JPEG2000 encoder section, this subsystem should provide the index of tiles containing some movement. The indexes could range from 0 to the total amount of tiles in any order and without any restriction. If no tiles with movement are detected (the current frame is almost equal to the last frame), this subsystem should somehow notify this circumstance and then send no tile for the current F_i^S frame (but the client side must be notified).

$$T_i^{INDEX} = Motion\left(F_i^S, F_i^R, T_{SIZE}\right) \tag{2}$$

So, this subsystem can be defined as in (2). In this case, we will specify what the 'Motion' function does in more detail, since this is the most important part of the adopted inter-frame technique. Fig. 7 is a detailed illustration of this function. We also describe all the tasks involved as follows:

- Preprocessing: this task prepares the source frame F_i^S for motion measurement. The conversion performed is related to the extraction of image intensity information, that is, the Y-luminance component of the YUV color space. The conversion specified in (3) is applied for RGB images. This way we can work with one simple representation of the image, leading to a faster analysis of source frames.

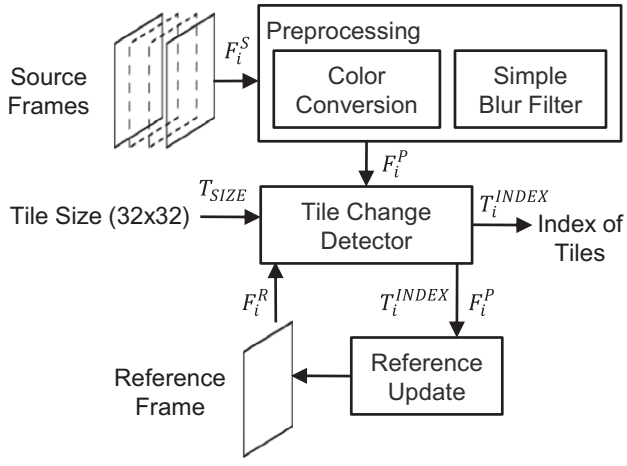


Fig. 7. Detailed MIJ2K motion measurement.

$$Y = 0.2999R + 0.587G + 0.114B \tag{3}$$

Once the source frame has been correctly transformed to a gray-scale image, an optional simple blur filter is used to reduce excessive detail and noise present in many surveillance cameras. We do not go into any more detail about the preprocessing algorithm and motion algorithm improvements, but we will try to determine whether the use of simple blur really does improve compression.

So, the preprocessed frame output by this subsystem, F_i^P, should be a gray-scale image conversion of F_i^S passed through a simple blur filter, as described in (4).

$$F_i^P = Blur\left(GrayScale\left(F_i^S\right)\right) \tag{4}$$

- Tile change detector: this module, illustrated in Fig. 7, should detect the tiles that contain movement for the purpose of selective tile compression and transmission. The method used in this subsystem should be relatively simple in order to meet real-time requirements.

  This subsystem compares the whole preprocessed frame F_i^P against the reference frame F_i^R tile by tile. It detects how much movement there is in each tile to decide whether or not it should be transmitted. Fig. 8 shows how this subsystem works. It is also described in detail in the following:
- Absolute difference $ABS_{i[x]}$: each tile from both preprocessed and reference frames is passed through an absolute difference filter to detect absolute changes between two tiles. The computational complexity of this operation is low, and it is useful for detecting objective differences between tiles. It generates a black-and-white image in which white pixels represent differences, whereas black pixels signify no changes. This is described in (5), where $F_{i[x]}^P$ and $F_{i[x]}^R$ are tile x of frame i in the preprocessed P and reference R frames. On the other hand, $ABS_{i[x]}$ represents the absolute difference between the tiles. In this case, both $F_{i[x]}^P$ and $F_{i[x]}^R$ are two matrices of $u \times v$ containing all the tile's pixel values.

$$ABS_{i[x]} = \left|F_{i[x]}^P - F_{i[x]}^R\right| \tag{5}$$

The tile image $ABS_{i[x]}$ output by the absolute difference process should be analyzed in order to detect how much movement there is, and thus decide whether or not it should be transmitted.

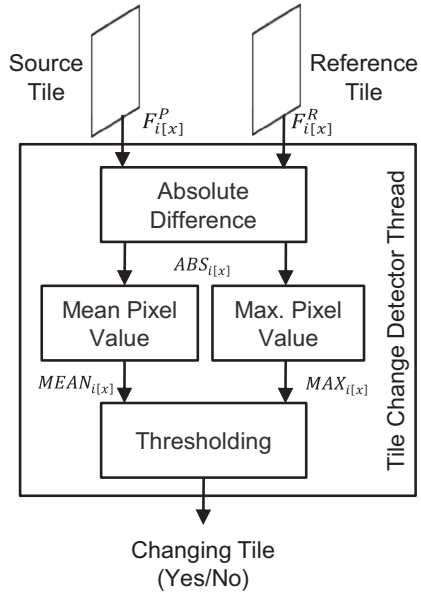


Fig. 8. MIJ2K tile change detector.

Changes should be measured somehow, and here we propose two efficient methods, which should work together.

- Mean value $MEAN_{i[x]}$: this is the first measurement. It is the mean pixel value of the $ABS_{i[x]}$ tile. This process takes all the values of the $ABS_{i[x]}$ tile, calculates the total and divides this by the number of tile pixels, as described in (6).

$$MEAN_{i[x]} = \frac{\sum_{s=0}^{u-1}\sum_{t=0}^{v-1} ABS_{i[x](s,t)}}{u \times v} \tag{6}$$

- Max value $MAX_{i[x]}$: the second measurement is the maximum pixel value of the $ABS_{i[x]}$ tile, as described in (7). This is useful for detecting movement peaks in tiles, since they could be overlooked by the mean value when nearby pixels values are near to zero.

$$\begin{aligned} &\forall s,t / s \geqslant 0 \wedge s < u, \quad t \geqslant 0 \wedge t < v \\ &MAX_{i[x]} = ABS_{i[x](s,t)} \Longleftrightarrow \neg\exists ABS_{i[x](g,h)} > ABS_{i[x](s,t)} \end{aligned} \tag{7}$$

  Operating in conjunction, $MEAN_{i[x]}$ and $MAX_{i[x]}$ can detect all kinds of movements, ranging from small uniform variations (using the mean) to occasional big changes (using max threshold). Both metrics are easy to implement, providing a very low complexity method for detecting movement.
- Thresholding: together $MEAN_{i[x]}$ and $MAX_{i[x]}$ indicators tell us when the tile should be transmitted. In this way, there is said to be a big enough change in a tile to warrant transmission when both values are above some threshold.
- Reference update: this subsystem is used to update the reference frame F_i^R. The first reference frame, F_0^R, is an entirely black image, and it is updated frame by frame with the tiles that are different from the preprocessed frame F_i^P. Logically, the first frame F_0^P will update all the tiles of the reference frame. The reference frame is also updated directly from the blurred image output by the preprocessing subsystem. This will stop the reference frame from having to be preprocessed each time and reduce system complexity.

  This subsystem introduces two new parameters for optimization, which are $MEAN_{i[x]}$ and $MAX_{i[x]}$. These variables determine motion detection algorithm sensitivity. *A priori* it is not feasible to set values for these parameters manually, and we will have evolutionary algorithms select the correct values.

3. Multiobjective optimization

As discussed in the introduction, one way to solve MO problems is obtain the entire Pareto front. This is a common research field, where the real challenge is to obtain the Pareto front with the minimum iteration length, because some objectives could be very costly to evaluate, and traditional search-based techniques are very time inefficient. In this way new evolutionary algorithms (EAs) (Back, Fogel, & Michalewicz, 1997) have been successfully extrapolated to multiobjective problems. EAs are well suited to MO optimization problems as they are fundamentally based on biological processes, which are inherently MO.

In this case, we will apply Multiobjective Evolutionary Algorithms (MOEAs) (Deb et al., 2001; Coello, Lamont, & Veldhuizen, 2006) to obtain the Pareto front within the video compression field. Compression is done by reducing the redundant information, like spatial and temporal redundancies, present in video scenes to the minimum. This way we can output compressed videos with apparently no loss of information, where the final amount of information needed for representation is reduced.

The problem in this case is that the loss of information (albeit redundant) amounts to a reduction in the quality or similarity of the compressed video compared with the original video. Fewer data necessarily imply lower quality, and, when trying to optimize the relation between conflicting objectives of compression and quality, we are faced with a MO problem.

There is then not a single problem solution that satisfies both objectives, where we obtain the highest compression ratio with the highest quality. The solution in this case is to output the best quality to compression ratio across the full encoder operating range and let the DM select the solution that meets his or her requirements.

The Multicriteria Decision Making (MCDM) (Ehrgott & Gandibleux, 2002) literature describes two general approaches for solving user-preference mechanisms involving MO problems. In the first approach, the DM gives preferences first and the algorithm outputs a set of solutions of preferred regions of the Pareto front according to preferences (*a* priori methods). In the second approach, the algorithm provides almost all the Pareto front solutions, and the DM selects the interesting options (*a posteriori* methods) (Miettinen, 2001).

A priori methods are preferred when there are many objective functions and the search space grows, since, in such cases, the computing resources can be focused on the preferred areas, returning results sooner. In this case, however, we are working with MO problems in the video compression field, where there are only two objective functions. Moreover, a wide encoder operating range must be covered, since DM constraints can easily vary depending on the use to which the encoder is put. So, *a posteriori* methods will be applied in this case to obtain the full Pareto front.

As shown in Fig. 9 we expect to obtain the full Pareto front between the conflicting objectives of quality, measured by PSNR, and CR. We can formally describe this problem as a set of objective functions to be jointly maximized (or minimized). This can be generally defined as a set of functions described in Eq. (8):

$$maximize\ f_n\ (\vec{v})\ where\ n = 1, 2, \ldots, N \tag{8}$$

In this case $f_n(\vec{v})$ is an objective function, and the solution $\vec{v}$ is a vector of M decision variables, which is given by $\vec{v} = (v_1, v_2, \ldots, v_M)$. The values of these decision variables should be between upper and lower bounds as defined by the problem.

Once we have obtained solutions, we can compare them using the notion of dominance, that is, given a decision vector $\vec{u} = (u_1, u_2, \ldots, u_k)$ is said to dominate other $\vec{v}_2 = (v_1, v_2, \ldots, v_k)$, if and only if

$$\forall i \in \{1, \ldots, k\}, \quad u_i \geqslant v_i \wedge \exists i_0 \in \{1, \ldots, k\} | u_{i_0} > v_{i_0} \tag{9}$$

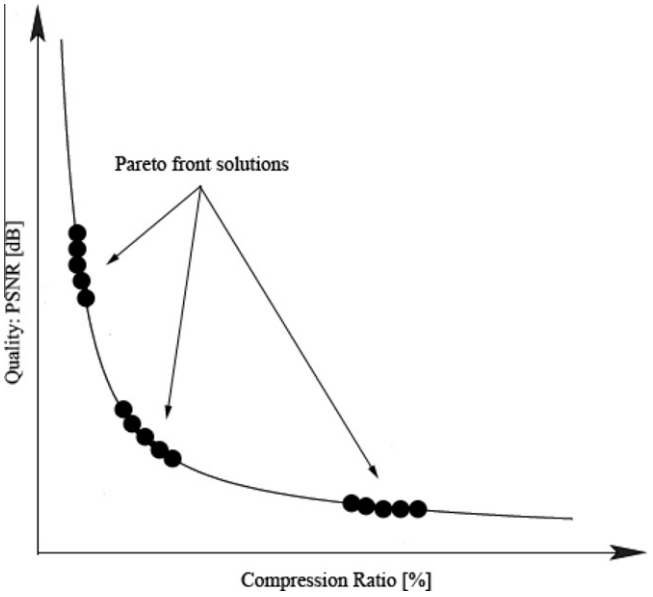


Fig. 9. Pareto front solutions.

In other words, the decision vector $\vec{u}$ is said to dominate $\vec{v}$ if and only if $\vec{u}$ is at least as good as $\vec{v}$ for all objectives and $\vec{u}$ is better than $\vec{v}$ for at least one objective. The Pareto optimal set is given when a decision vector is not dominated by any other vector in the search space.

4. MOEA approach to video encoder optimization

In this section we discuss the decision variables that the encoder uses, which we will optimize using NSGA-II, specifying the values of the upper and lower bounds of each one, and how they affect the encoder. Moreover we detail the objective functions used to obtain the Pareto front.

4.1. Objective functions

We will use two standard functions in the scope of video compression to evaluate the performance of each vector of solutions. The first function, f_1, is the average PSNR (12), that represents an objective quality metric, and the second function, f_2, is the compression ratio (13), which describes a ratio of compressed to original video sizes. These functions are described below:

$$MSE_f = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\|I_f(i,j) - K_f(i,j)\|^2 \tag{10}$$

$$PSNR_f = 20\cdot\log_{10}\left(\frac{255}{\sqrt{MSE_f}}\right) \tag{11}$$

$$f_1 = \frac{\sum_{f=1}^{\#Frames} PSNR_f}{\#Frames} \tag{12}$$

$$f_2 = CR = \frac{UncompressedSize}{CompressedSize} \tag{13}$$

The mean square error (MSE_f) is calculated pixel by pixel for each video frame f of $M \times N$ pixels between the original frame, represented by I_f, and the reconstructed compressed frame, K_f. Then, the $PSNR_f$ is calculated for each video frame f over the MSE_f value. Notice that *MSE* is calculated over the luminance component Y' (brightness) of the YUV color space to compute the PSNR.

These two objective functions should be maximized, since higher values of CR will mean lower compressed video sizes, and a greater number of dBs in the PSNR metric is equal to a lower MSE, leading to a greater similarity between original and compressed video frames.

4.2. Decision variables

The encoder we intend to optimize is based on intra-frame encoder techniques (Connor, Brainard, & Limb, 1972), but has been extrapolated to inter-frame coding (Brofferio & Rocca, 1977), basically using a video scene motion detection system (MMS) with the aim of performing simple motion compensation. It is based on JPEG2000 (ISO/IEC, 2000) and its still images compression method. This essentially divides each frame into independent accessible square regions, also called tiles. Our encoder uses tiles for conditional replenishment depending on whether or not motion is detected in the same way as similar encoders do with empty macroblocks.

Each frame or still image in the video sequence can be compressed using JPEG2000 to a specific quality, which will determine the final quality and size of the compressed video in combination with the motion detection system.

All the decision variables or low level encoder parameters that we will try to optimize are summarized below, stating the range of values of each one, divided into functional units.

1. *Tiling setup*
   - $T_{SIZE} = \{16\times16, 24\times24, 32\times32\}$
2. *Motion detection system*
   - *Smoothing* = {*Enabled*, *Disabled*}
   - $MEAN_{i[x]} = [0,255]$
   - $MAX_{i[x]} = [0,255]$
3. *Still image quality control*
   - *Bits per pixel in* 32×32 (bpp_{32}) = [0.4, 7.5]
   - *Bits per pixel in* 24×24 (bpp_{24}) = [0.7, 8.73]
   - *Bits per pixel in* 16×16 (bpp_{16}) = [1.2, 11.0]

Tiling setup represents the size of the squared regions of the still images once compressed. The width and height of each tile must be the same. In theory, lower tile sizes will perform better compression, since they make the motion compensation system more precise. On the other hand, they add an extra overhead, making the compressor less efficient.

Motion detection parameters are the attributes that manage the motion measurement subsystem (MMS) in tile regions. Large values of $MEAN_{i[x]}$ and $MAX_{i[x]}$ will represent a high MMS sensitivity, and lower tile reusability, which will imply better quality but less CR. We think that, in the case of *Smoothing*, which is related to the simple blur filtering, improves the MMS by reducing the noise that is usually present in videos, but we really do not know exactly what effect it has and whether there is any real improvement.

On the other hand, still image quality control only has one parameter. It represents the bits per pixel that the JPEG2000 compressor should use for each still frame. Again, higher values of this parameter will lead to better quality but less CR. Notice that, for each tile size, there is a specific operating range for this value.

Observe how each of these low level encoder parameters actually represents a MO problem. For this reason, we will try to obtain a set of parameter values that achieves a set of solutions near to the optimal Pareto front.

5. Experiments

In this section we present all the tests performed to get the near-optimal Pareto solutions set of the encoder, using the NSGA-II MOEA. We detail both the purposes of the tests and the parameters used in each execution and discuss the results. For these experiments, we will use *a posteriori* methods, that is, output almost the entire Pareto front of the encoder, leaving it to the decision maker to choose the solution that meets his or her requirements *a posteriori*.

The decision variables for optimization are the stated in Section 4.2. Notice that T_{SIZE} and *Smoothing* are discrete variables. Therefore, we will perform additional tests to determine their best values. We will use the concept of Pareto dominance to compare the solutions achieved by the NSGA-II algorithm.

The NSGA-II algorithm will run a maximum of 200 generations with an initial population of 200. The population size has been set in order to ideally get an individual about every 0.15 dB, covering the 25 to 55 dB encoder operating range. This way the DM's solution selection will be more precise in terms of quality. Each test will be performed twice with the Hall Monitor Fig. 10(a) and Akiyo 10(b) test sequences.

5.1. Video test sequences

The test video sequences used to evaluate the encoder are very commonly used to evaluate encoder performance. Selected sequences are 10(a) and (b) from the Xiph.org repository. They are uncompressed CIF-format sequences with a length of 251 frames and a size of 352×288 pixels.

(a) Hall Monitor (b) Akiyo

Fig. 10. Test video sequences.

5.2. Smoothing test

The first test performed is related to the MMS, and particularly to determining how the *Smoothing* parameter affects f_1 and f_2 objective functions when optimizing the $MEAN_{i[x]}$, $MAX_{i[x]}$, and *Bits per pixel* decision variables. *Smoothing* is a low level encoder parameter that can be enabled or disabled only. It should improve the performance of the encoder when enabled, but we have no hard evidence of this. It should be the first test run since this parameter depends on the *tile size* parameter, but not *viceversa*. In this case then we can establish an arbitrary tile size, i.e. 32×32. The *Bits per pixel* range of values is set according to the tile size, as defined in the decision variables section and shown in Table 1.

This test will generate two Pareto fronts that differ only as to whether the encoder uses *Smoothing* prefiltering. So, we could use the concept of Pareto dominance to determine whether *Smoothing* is really useful.

Figs. 11 and 12 both show the Pareto solution set obtained for each test video with the *Smoothing* parameter enabled and disabled. It is clear that the Pareto solution sets with the *Smoothing* parameter enabled dominate the others with *Smoothing* disabled.

This means that the *Smoothing* parameter really does improve the MMS and the final compression performance when enabled. So, for the best results, the *Smoothing* value should be enabled during the compression. In the following tests, *Smoothing* will always be enabled.

5.3. Tile size test

In the second test we will try to determine the influence of the tile size on compression. We can determine whether there is a tile size that outperforms the others. *A priori*, lower tile sizes should perform better compression, improving the MMS, since smaller tile sizes usually fit any object in the image better. Also other video codecs use small macroblock sizes, i.e. H.264 (Wiegand, Sullivan, Bjntegaard, & Luthra, 2003) that works with 16×16, 8×8 and also 4×4.

Table 1
Encoder parameters $Test_1$.

$Test_1$	Run_1	Run_2
T_{SIZE}	32×32	32×32
Smoothing	Enabled	Disabled
$MEAN_{i[x]}$	[0 255]	[0 255]
$MAX_{i[x]}$	[0 255]	[0 255]
bpp_i	[0.4, 7.5]	[0.4, 7.5]

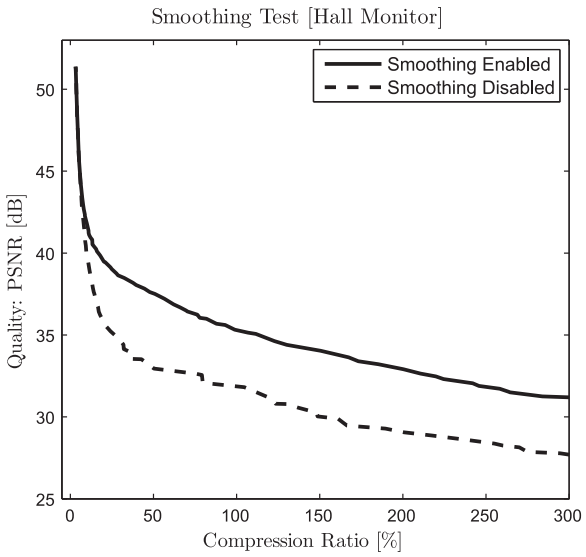


Fig. 11. Smoothing test [Hall Monitor].

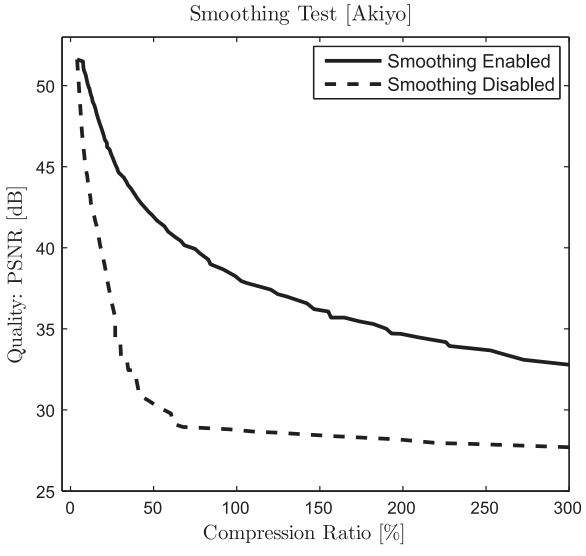


Fig. 12. Smoothing test [Akiyo].

Note, however, that more data are necessary to represent smaller tile sizes in JPEG2000, i.e. observe how the lower bound of the *Bits per Pixel* low level encoder parameter grows as tile size decreases. This indicates that there is an extra overhead on the inclusion of each tile.

In this test we want to compare the Pareto fronts achieved by the $MEAN_{i[x]}$, $MAX_{i[x]}$, and *Bits per pixel* decision variables with the different possible values for the *Tile size* parameter. We could determine the best value for this parameter comparing the Pareto solution sets.

Table 2 describes the parameters used in this case for the three executions of this test, one for each tile size. Notice, in this case, that *Smoothing* is enabled in all tests, as a consequence of the previous result. This test will provide three Pareto fronts, indicating the best tile size to be used with the encoder.

Table 2
Encoder parameters $Test_1$.

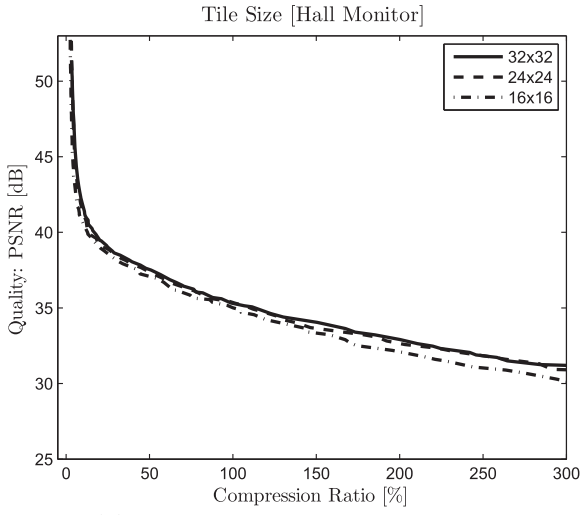
$Test_2$	Run_1	Run_2	Run_3
T_{SIZE}	16×16	24×24	32×32
Smoothing	Enabled	Enabled	Enabled
$MEAN_{i[x]}$	[0 255]	[0 255]	[0 255]
$MAX_{i[x]}$	[0 255]	[0 255]	[0 255]
bpp_i	[1.2, 11.0]	[0.7, 8.73]	[0.4, 7.5]

Fig. 13(a) and Fig. 14(a) show how all the Pareto fronts obtained are very similar for each tile size. But there is a slight performance improvement when using 32×32 tiles. Fig. 13(b) and Fig. 14(b) show a detailed area of the Pareto front, located at the Pareto front inflection point, clearly indicating how the 32×32 Pareto front dominates the other Pareto solution sets.

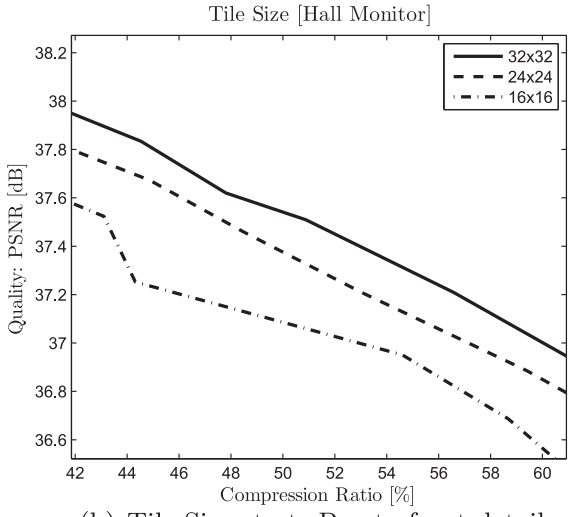
From the comparison of these Pareto fronts generated with NSGA-II we can select the best value for the *Tile Size* low level encoder parameter, which will be set at 32×32. We also have to prove how the overhead present with small tile sizes is not offset by the theoretical improvement in the MMS.

5.4. Results of optimization

In the previous tests we have determined the best values for the discrete encoder parameters (T_{SIZE} and *Smoothing*), and we now know that $T_{SIZE} = 32 \times 32$ and *Smoothing* = *enabled* improves the compression. The remaining parameters ($MEAN_{i[x]}$, $MAX_{i[x]}$ and bpp_i) are continuous, and their values are determined for each vector of solutions by the NSGA-II algorithm.

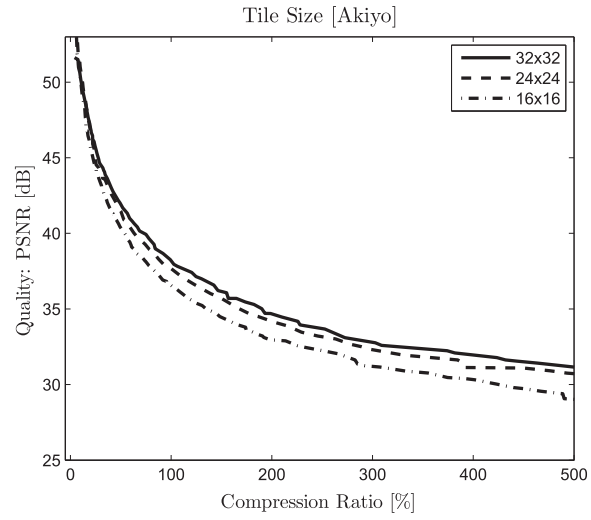


(a) Tile Sizes test. Full pareto front

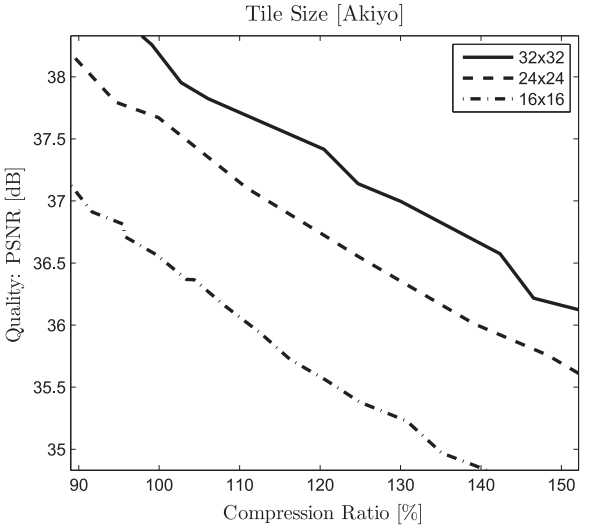


(b) Tile Sizes test. Pareto front detail

Fig. 13. Tile sizes test [Hall Monitor].



(a) Tile Sizes test. Full pareto front



(b) Tile Sizes test. Pareto front detail

Fig. 14. Tile sizes test [Akiyo].

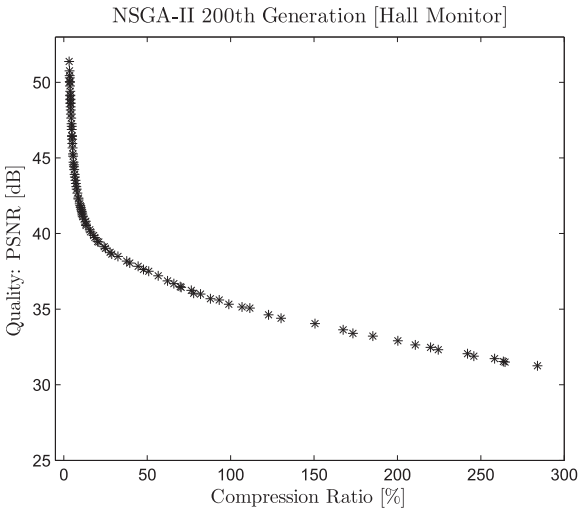


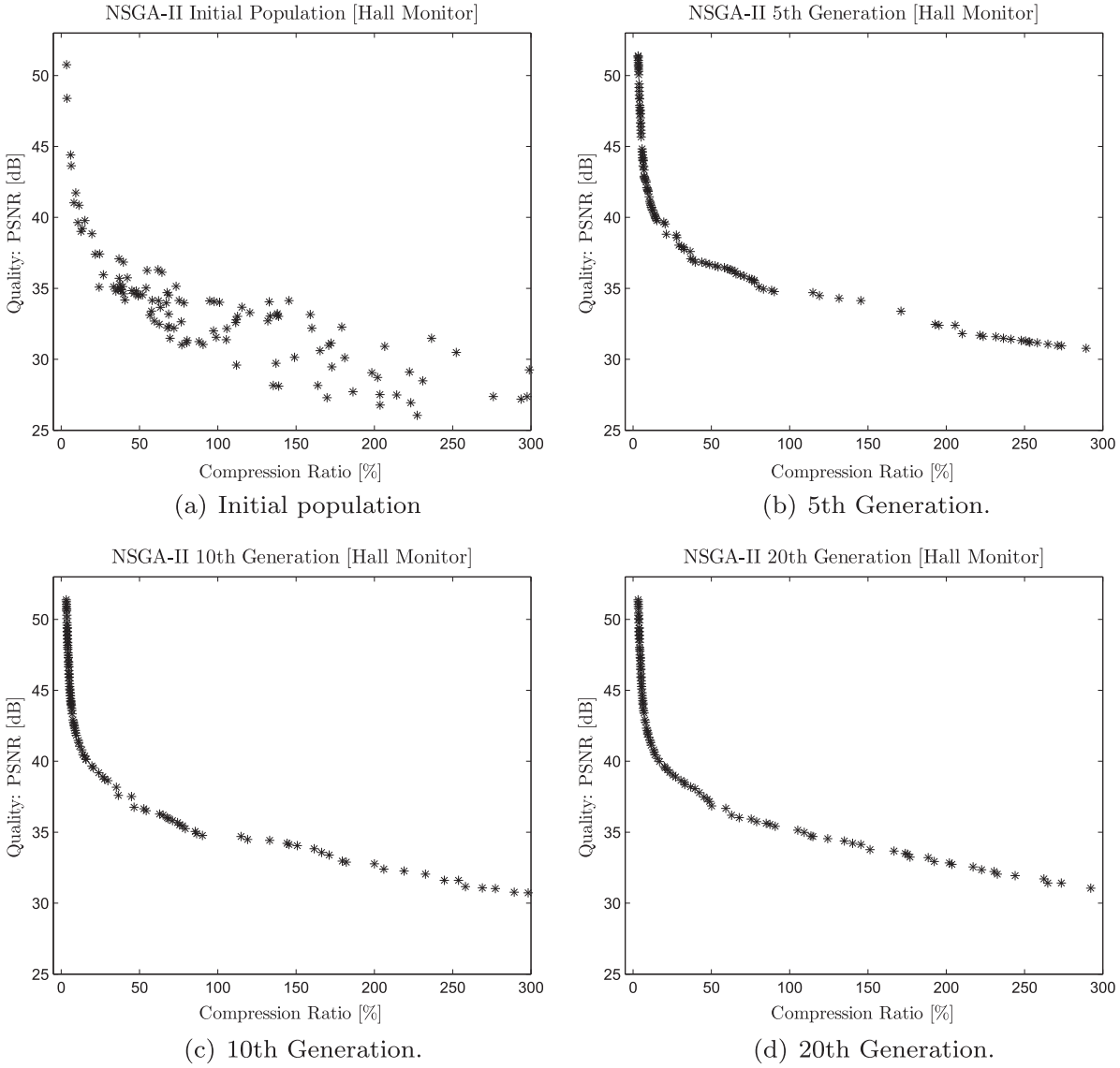
Fig. 15. Final Pareto solution set [Hall Monitor].

The DM using the results of this optimization could select any solution from the Pareto front that meets his or her requirements (in terms of quality or compression ratio). The Pareto front contains a set of values of $MEAN_{i[x]}$, $MAX_{i[x]}$ and bpp_i. The DM can also use the Pareto front to evaluate the trade-offs between all solutions. In this section we will report the Pareto solution sets for each video sequence.

Figs. 15 and 17 represent the best Pareto solution sets for each compressed video. They illustrate how these solution sets cover a wide operating range in terms of quality and compression ratio.

Also we want to show the convergence speed of NSGA-II for such problems. Fig. 16(a)–(d) show the evolution of generations for the Hall Monitor test video, and how the solution approximated in the 20th generation is almost equal to the Pareto front solution set obtained with 200 generations, illustrated in Fig. 15.

With the Akiyo test sequence, on the other hand, convergence speeds up, and, by the 5th generation (Fig. 18), we get a good Pareto solution set, compared with the solution obtained in 200th generation, shown in Fig. 17.



(a) Initial population (b) 5th Generation.

(c) 10th Generation. (d) 20th Generation.

Fig. 16. NSGA-II convergence [Hall Monitor].

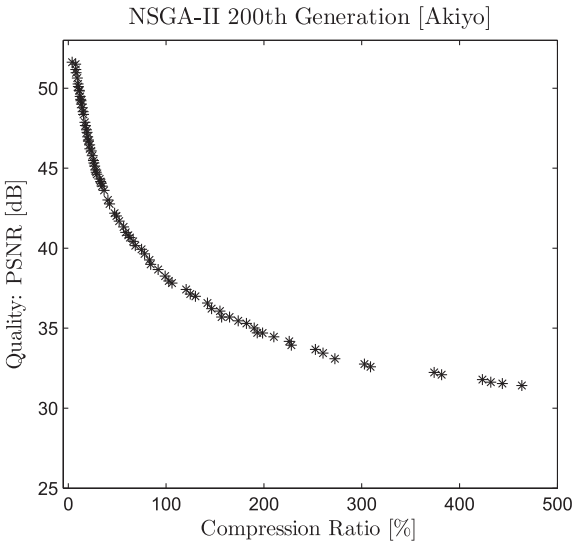


Fig. 17. Final Pareto solution set [Akiyo].

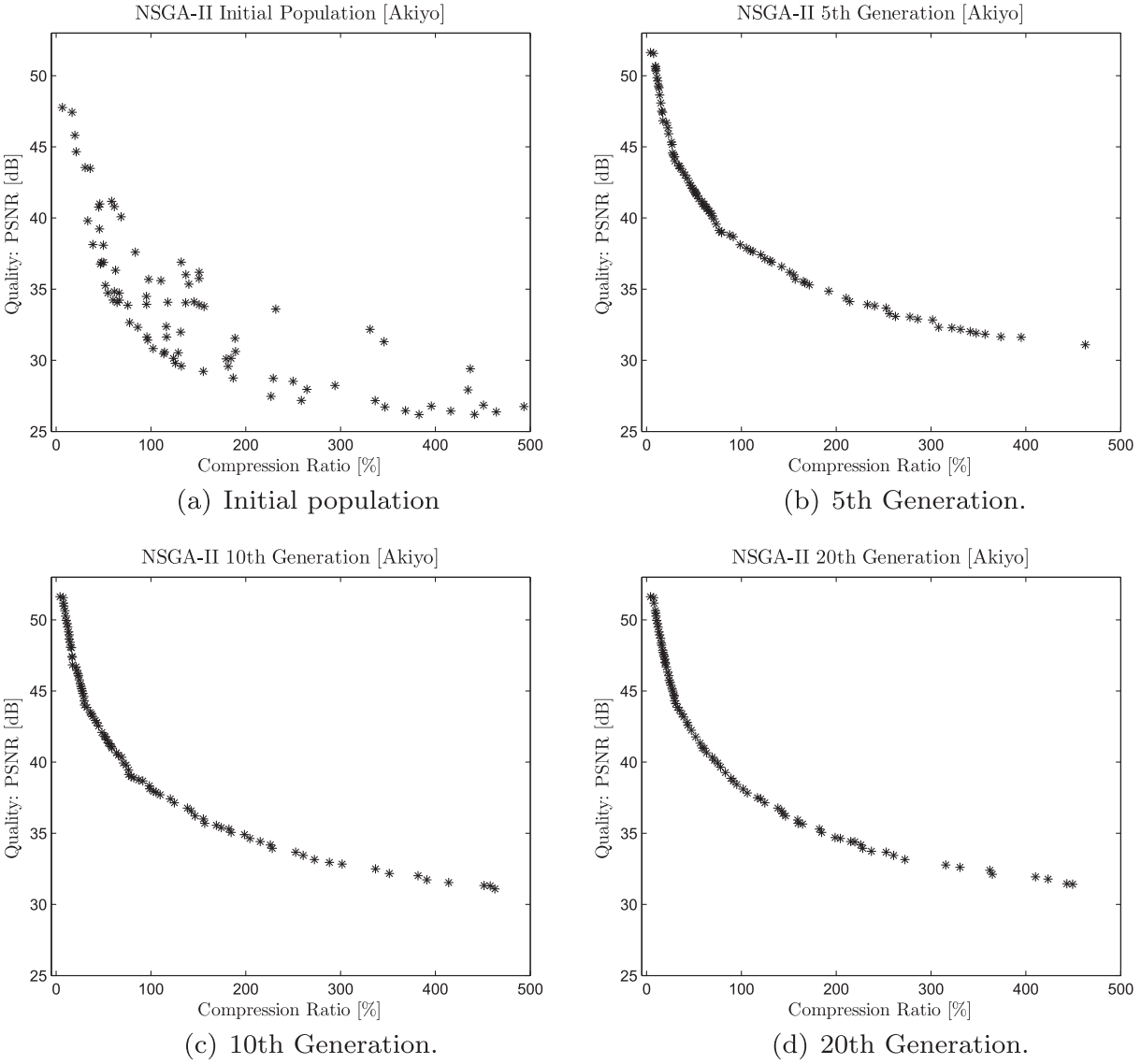
This demonstrates the suitability of using MOEAs for such problems that have a large search space and are not suited for approximation using classical search-based techniques. In actual fact, they take about two minutes to evaluate f_1 and f_2 objective functions for each individual in this problem.

6. Conclusions

In this paper we have worked with the multiobjective definition in the field of video compression. We have used NSGA-II to optimize the internal parameters of a new video codec called MIJ2K, but this procedure could be extrapolated to any video compressor with good results. Using MOEAs we can output an entire Pareto solution set covering the whole operating range of the codec for both quality and compression ratio.

Also we show the suitability of using MOEAs in the field of video compression, since they achieve a fast convergence speed, as required by such problems due to the cost of evaluating each objective function.

Even so, we intend to further research this problem. In this paper, we have optimized the low level encoder parameters in order



(a) Initial population

(b) 5th Generation.

(c) 10th Generation.

(d) 20th Generation.

Fig. 18. NSGA-II convergence [Akiyo].

to create a Pareto solution set for a specific test video. The solutions obtained can be used with different videos that share similar profiles to the videos used here with the aim of reusing the optimization.

In future work, we will research real-time optimization depending on given constraints independently of the video that is being compressed. This is potentially very useful, for instance, when the video compressor is used for real-time streaming purposes and the network constraints are variable depending on load. The algorithm should then search new optimal solutions for the new given constraints.

Acknowledgments

This work was supported in part by Projects CICYT TIN2008-06742-C02-02/TSI, CICYT TEC2008-06732-C02-02/TEC, SINPROB, CAM MADRINET S-0505/TIC/0255 and DPS2008-07029-C02-02.

References

Adams, M. 2001. The JPEG-2000 still image compression standard. ISO/IEC JTC 1/SC 29/WG 1, 2412.

Adams, M., Ward, R. 2001. Wavelet transforms in the JPEG-2000 standard. In Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, vol. 1, pp. 160–163.

Back, T., Fogel, D. B., & Michalewicz, Z. (Eds.). (1997). *Handbook of Evolutionary Computation*. Bristol, UK, UK: IOP Publishing Ltd..

Brofferio, S., & Rocca, F. (1977). Interframe redundancy reduction of video signals generated by translating objects. *IEEE Transactions on Communications [legacy, pre-1988], 25*(4), 448–455.

Christopoulos, C., Skodras, A., & Ebrahimi, T. (2000). The JPEG 2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics, 46*(4), 1103–1127.

Coello, C. A. C., Lamont, G. B., & Veldhuizen, D. A. V. (2006). *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc..

Connor, D., Brainard, R., & Limb, J. (1972). Intraframe coding for picture transmission. *Proceedings of the IEEE, 60*(7), 779–791.

Czuni, L., Csaszar, G., & Licsar, A. (2006). Estimating the Optimal Quantization Parameter in h. 264. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition* (pp. 330–333). Washington, DC, USA: IEEE Computer Society.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester: Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation, 6*, 182–197.

Ehrgott, M., & Gandibleux, X. (2002). *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. Kluwer Academic Publishers..

Hang, H.-M., Chou, Y.-M., & Cheng, S.-C. (1997). Motion estimation for video coding standards. *Journal of VLSI Signal Processings and Systems, 17*(2–3), 113–136.

ISO/IEC. 15444-1:2000 information technology jpeg2000 image coding system-part 1: core coding system. Technical report.

Jiang, M. 2006. Adaptive rate control for advanced video coding. PhD thesis, Santa Clara, CA, USA. Adviser-Nam Ling.

Koski, J. (1988). Multicriteria truss optimization. *Engineering and in the Sciences*.

kuang Chen, Y., Vetro, A., Sun, H. and Kung, S.Y. 1997. Optimizing intra/inter coding mode decisions. In Proceedings of International Symposium on Multimedia Information Processing, pp. 561–568.

Lora, M. 1994-2008. Xiph.org:: Test media. World Wide Web electronic publication.

Luis, A., Patricio, M. Scalable Streaming of JPEG 2000 Live Video Using RTP over UDP.

Miettinen, K. (2001). Some methods for nonlinear multi-objective optimization. *Lecture Notes in Computer Science*, 1–20.

Ng, J. K.-Y., Leung, K. R., & Hui, C. K.-C. (2005). A qos-enabled transmission scheme for mpeg video streaming. *Real-Time Systems, 30*(3), 217–256.

Pennebaker, W., & Mitchell, J. (1993). *JPEG Still Image Data Compression Standard*. Kluwer Academic Publishers..

Rabbani, M., & Joshi, R. (2002). An overview of the JPEG 2000 still image compression standard. *Signal Processing: Image Communication, 17*(1), 3–48.

Sawaragi, Y., Nakayama, H., & Tanino, T. (1985). *Theory of Multiobjective Optimization*. Orlando: Academic Press.

Shi, Y., Sun, H. 2000. Image and video compression for multimedia engineering: fundamentals, algorithms, and standards, CRC Pr I Llc.

Steuer, R. E. (1986). *Multiple Criteria Optimization: Theory Computation and Application*. New York: John Wiley. pp. 546.

Thurston, D. L. (2006). Multi-Attribute Utility Analysis of Conflicting Preferences. In Kemper E. Lewis et al. (Eds.), *Decision Making in Engineering Design*. New York, New York: ASME.

Tu, Y.-K., Yang, J.-F., Shen, Y.-N., & Sun, M.-T. (2003). Fast Variable-Size Block Motion Estimation using Merging Procedure with an Adaptive Threshold. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo* (pp. 789–792). Washington, DC, USA: IEEE Computer Society.

Wallace, G. 1991. The JPEG still picture compression standard.

Wang, Y.-C., & Leou, J.-J. (2003). A Rate Control Scheme for h. 26l Video Transmission. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo – Volume 3 (ICME '03)* (pp. 349–352). Washington, DC, USA: IEEE Computer Society.

Wiegand, T., Sullivan, G., Bjntegaard, G., & Luthra, A. (2003). Overview of the H. 264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology, 13*(7), 560–576.

Zhang, Q., Zhu, W., & Ya-W, Z. (2005). End-to-end qos for video delivery over wireless internet. *Proceedings of the IEEE, 93*(1), 123–134.