



**Politecnico
di Torino**

Politecnico di Torino

Master degree program
in Cinema and Media Engineering

AY 2022-2023

March-April graduation session 2023

Master Thesis

The Dynamic Optimizer Framework

Video encoding, assessment and comparison

Author

Chemin Davide

Supervisor

Masala Enrico

ABSTRACT

Delivery costs, low latency, and streaming at scale are key challenges in driving the technical development of the video industry. Improvements in video coding are always pushing the effort for new video coding solutions and quality monitoring. The recently released coding standards, like AV1 and VVC, are proof of this. In line with the constant aim of ensuring better compression ratios for the same visual quality, we analysed a new approach to digital video compression. The so-called Dynamic Optimizer has been proposed and conceived originally by Netflix, it is compatible with any existing and future video codecs, and it is suitable for non-real time encoding of on-demand video contents in adaptive streaming applications. Therefore, its ideal resources are long sequences of shots, typical of the Netflix catalogue. It works by fine-tuning the available encoding parameters for the single shots in order to find the optimal combination that respects a given bitrate or quality target. We developed a software capable of reproducing the proposed optimizer, which will be released with an open-source licence. It consists of three alternative implementations: a brute force approach that encodes and compares all possible combinations and it is very expensive in terms of number of computations and encodings, and the Lagrangian optimization, based on searching the optimal minimum of the convex hull of all combinations. The third and the most efficient proposed solution instead, which constitutes the innovative element of the work, is based on the RD curves approximation. In this thesis, we analyse the performances of the dynamic optimizer thanks to a comprehensive set of objective assessment tests, using the VMAF quality metric and the AVC, HEVC, VP9, and AV1 encoders. The results highlight the advantages and disadvantages of the different encodings in various bitrate and quality ranges and for diverse types of content.

Keywords. Video encoding, perceptual visual quality, rate-distortion theory, dynamic optimization, lagrangian optimization.



TABLE OF CONTENTS

Introduction	5
1. Video Coding	9
1.1 Analog and digital signals	9
The spread of digital	11
Sampling and quantization	14
2D digital image	15
Chroma subsampling	16
1.2 Compression	18
Compression ratio	18
Compression algorithms	19
1.3 Video coding	21
The JPEG process	24
Interframe coding	25
Motion estimation and compensation	26
1.4 Coding standards	28
H.264, H.265, VP9, and AV1	30
Codecs evolution	32
1.5 Quality	34
Quality metrics	34
Bitrate control	39
1.6 Delivery	41
IP protocol	42
Video streaming	43
Further optimisation	46

2. Dynamic optimization	47
Overview	49
Shot detection	51
Shot encoding	53
Quality assessment	56
Muxing	60
2.1 Brute force method	61
2.2 Lagrangian method	64
2.3 Curve fitting method	69
3. Assessment and results	77
3.1 Objective evaluation	78
Methodology	78
Test video sequences	80
3.2 Optimization results	85
Fixed CRF and Brute force	85
Fixed CRF and Lagrangian methods	87
Number of elemental encodes	92
VMAF and PSNR	94
3.3 Content dependency	96
3.4 Implications	100
Conclusions	102
Bibliography	104

INTRODUCTION

Digital video coding has been increasingly important in the communication field since the introduction of the first video codec in the late 1980s, which led to the progressive shift from traditional analog video processing to digital systems. Since then, this trend has continued to grow, with digital content production and consumption reaching new heights. According to Cisco (2020), by the end of 2023, video content will represent 82% of all internet traffic, up from 75% in 2018. Over-The-Top (OTT) companies significantly contribute to further fuel these figures, mainly because of the rapid expansion of streaming services. OTT platforms deliver video content directly to viewers over the internet, bypassing traditional cable, satellite, or broadcast television transmissions, and enabling users to access video content on-demand. They are heavily investing in the production and releasing of their own content. Likewise, traditional TV and media organisations are also embracing the shift to streaming, and some of them are launching their own on-demand offerings, contributing to the growth of the global demand for video content. With the continuous spread of high-speed internet, the increasing need of high-quality contents, the widespread adoption of mobile devices, and the ongoing advancements in video encoding technologies, the trend towards digital cannot do anything but continue to grow.

From a technical point of view, and specifically in signal theory, the use of a binary representation offers several advantages, including improved resistance to noise and error correction, effective signal reconstruction, less complex circuits and algorithms, as well as stronger encryption techniques. Digital video systems are generally considered cheaper than analog ones because of their simplification in both the hardware and software implementation. Moreover, one of the most significant benefits of the binary representation is reduced memory and transmission costs due to compression, yet at the expense of less accuracy and precision in the data description.

Over the last two decades, video compression and visual quality have gained increasing attention as two of the most researched topics in image and video processing. For this reason, the first chapter of this work is devoted to the theoretical explanation of all steps involved in the digital video processing chain, from digital to analog conversion (*Chapter 1.1*) to streaming delivery (*Chapter 1.6*),

focusing in particular on compression techniques (*Chapter 1.2*) and on the video coding block diagram (*Chapter 1.3*).

After more than 30 years, the fundamentals of video coding described in those sections have not changed in their basic operational principles, but improvements are still requiring huge efforts in order to make video coding technologies more and more performant in terms of data rate and perceptual quality. The well-known video coding standards such as AVC, HEVC, and VP9, together with the most recent ones like VVC and AV1 (*Chapter 1.4*), are adopted in the majority of encoding and decoding applications, and always aim at allowing higher compression ratio and bitrate reduction for the same quality compared to their predecessors. Great advances have already been achieved also in quality monitoring (*Chapter 1.5*), primarily in making objective quality metrics more fitting to human visual system perception, for example with the introduction of Video Multimethod Assessment Fusion (VMAF) quality metric over the traditional Peak Signal-to-Noise Ratio (PSNR).

The proliferation of video streaming services and the surge in demand for video content over the internet arouse new challenges in the fields of video compression and delivery, especially in finding the perfect balance between rate and quality. As streaming becomes a ubiquitous way of consuming multimedia content, the amount of data to transmit also rises, and with it the demand for efficient and high-quality video encoding and decoding. Nonetheless, streaming is subject to strict network requirements and viewing conditions that differ from its different purposes, like video on-demand, video conferencing, online interactive gaming, or mobile fruition. Offering a good experience to final users remains one of the main goals of OTT providers, which translates into keeping quality as high as possible, especially in the context of mobile video applications and poor or fluctuating networks. In this sense, besides the progress in the standardisation of new coding algorithms, and in order to get closer to the goal, the video industry can rely on the viable support of optimization. Nowadays indeed, big companies are struggling to fine-tune the available encoding parameters to try to create the perfect balance for their constantly expanding video contents offer.

Static optimization is the current simplest solution. According to this method, video parameters, and so visual quality, are determined before the streaming starts, by considering the network conditions and the device resolution. From then on, they never change during playback. However, in a real-world environment, conditions may vary significantly, and this means that the video stream should be optimised for each individual user while running, ensuring the best possible viewing experience at any time.

This work proves the efficacy of a new approach to digital video compression, fully described in the second chapter and evaluated in the third one, the so-called Dynamic Optimizer. It was initially proposed by Netflix in 2018 (Katsavounidis, 2018) as part of the larger container of adaptive streaming, and one of its main advantages is being compatible with all existing compression techniques. The word *Optimizer* refers to the search of an optimal solution among different possibilities, the one that minimises or maximises a parameter, given a specific target or constraint. The word *Dynamic* instead, stands for the ability of the system to adapt this search to changing conditions. For an input video sequence, once set the output target, the system must find the optimal combination of shots that does not cross the target. In fact, each shot is encoded with custom parameters at different degrees of compression in order to provide allowance for optimisation.

The Dynamic Optimizer has been designed for on-demand contents, which are pre-processed at the source server before sending, and it is particularly suitable for long sequences of non-homogeneous shots typical of the Netflix catalogue. On the other hand instead, it does not accrue any benefit, for example, to video conferencing and live streaming services.

A good number of studies on the topic can be found in the current research literature, with the main purpose of describing and assessing this framework, but no software implementations are available so far. For this reason, we propose the first open-source software implementation of a video Dynamic Optimizer, which comes with three different optimisation techniques. The first one adopts a brute force approach (*Chapter 2.1*), very inefficient from a computational point of view, but able to find the best optimal solution, so that its output can be used as a comparative reference for the other two methods. They are both based on Lagrangian optimization, an efficient way to find the maximum or minimum value of a function under one or more constraints, in our case the quality or rate targets. It uses the slope of the tangents to the function as a decision condition to reach the convergence point, otherwise known as optimum. The traditional lagrangian optimization technique (*Chapter 2.2*) requires the exhaustive encoding of all shots at different levels of distortion, which has high encoding costs. For this reason, in the third method, in order to reduce the total number of encodings, we designed a lagrangian algorithm based on curve fitting (*Chapter 2.3*), which reconstructs the behaviour of a clip in the form of a curve from the position of a limited number of encoded points. In this way, all the rate-distortion curve values are estimated and approximated based on a few encoded shots.

The process of assessing video resources using Rate-Distortion (RD) curves is the last step of this work and it is described and performed in the third chapter. In this

regard, the evaluation methodology starts with the selection of a set of video sequences covering a wide range of different contents, complexities and features, such as various resolutions, frame rates, levels of detail or motion. According to common practices in video research, when it is not possible to perform subjective tests for measuring the quality of video contents, objective metrics like PSNR or VMAF may be used to compensate for the subjective assessment in a systematic and automated way. Rate-Distortion results are strongly linked to the chosen quality metric, and its choice affects compression performances too. Indeed, the RD theory is a fundamental concept in video coding, because it allows encoders to make informed decisions about the amount of compression to apply to a given video frame or shot, based on the desired visual quality and the available bitrate. In their graphical representation, RD curves visualise the trade-off between bitrate and quality for the analysed video coding algorithms, making their comparison possible. Thus, they are used in the third chapter to evaluate the performances of the three methods, contrasting the optimised sequences to their non-optimized versions (*Chapter 3.2*), but also combining together video resources with different contents and characteristics (*Chapter 3.3*) in order to understand under which conditions the presented framework performs better.

As video streaming services continue to grow and new formats and standards are developed, it is also worth considering new alternative paths and non-standardised techniques. The one discussed in this work, dynamic optimization, is showing promising results in achieving the higher quality at the lowest possible bitrate, in the same way as new developed coding algorithms. Thus, one can imagine to include it into next-generation standards themselves. In this way, we might also improve the overall viewing experience for users, while reducing the bandwidth requirements and storage costs at the same time. In the end, with further research and development, dynamic optimizers can begin to play a significant role in the advancements in the video coding and video compression fields.

1. VIDEO CODING

1.1 ANALOG AND DIGITAL SIGNALS

The signal is the way we represent and convey information. Any emission or reception of information through any telecommunication system is indeed a signal flow in input or output. In the communication and telecommunication field, a signal is often represented in the form of a wave that propagates in space and time. The process of transmitting it is referred to as signal transmission, which involves the transfer of information from one location to another through various channels, with the goal of sending the signal with minimal distortion, noise, or loss of data (Simon & Van Veen, 2003), so that the information can be accurately recovered at the receiving end. We distinguish analog from digital signals.

Analog signals are continuous electrical or electromagnetic signals that vary in amplitude, phase, or frequency. They are transmitted using a continuously changing physical quantity (Keith, 2005), like pressure or voltage, which can be described by a wave shape (*Figure 1.1*). The physical phenomena of sending signals indeed consists of a transmission of continuous electromagnetic waves, like the television or the sound waves.

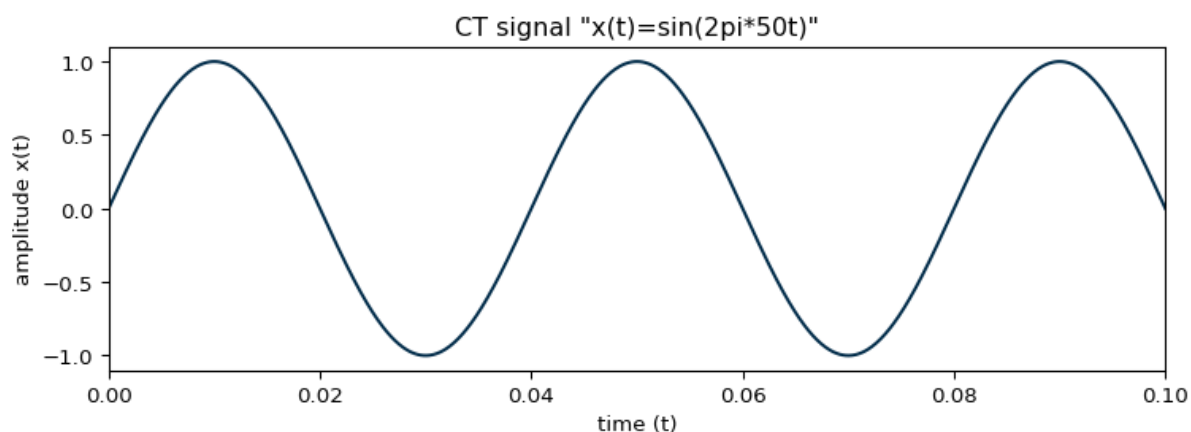


Figure 1.1, real valued CT sinusoidal signal $x(t)=\sin(2\pi 50t)$.

In the early days of communication and signal processing, analog signals were the only form of representation and transmission. The past century however saw the rise of the digital era. Digital signals, in contrast to analog ones, are discrete

representations of the same information. The history of the digital world dates back to the mid-20th century, during the second world war, when digital technology was first applied to the field of communication. A key date in this context is 1948, which coincides with the publication of “A mathematical theory of communication” by Claude Shannon (1948) and “The philosophy of PCM” by Bernard M. Oliver, John R. Pierce and Shannon (1948). PCM, Pulse Code Modulation, is one of the earliest examples of digital communication in telecommunication systems, already invented by Paul M. Rainey in 1926 and remained muted until that time (Nebeker, 1998). PCM is a method for digitally representing analog signals in which an analog waveform is sampled and quantized into a series of binary values. In the 1960s and 1970s, advances in digital technology led to the development of digital computers, which were capable of processing and transmitting digital data. This paved the way for the growth of the internet and other communication technologies that we use today.

Digital signals are discrete in time and frequency, they can take on a finite number of distinct values (Oppenheim and Schaffer, 2010), which represent the amplitude of a waveform at a specific moment in time. The most common form of digital representation is the binary code (*Figure 1.2*). It is only made of two symbols, the values 0 and 1, making it well suited for processing by computers and other digital devices. It allows for a more precise and efficient manipulation and transmission of information.

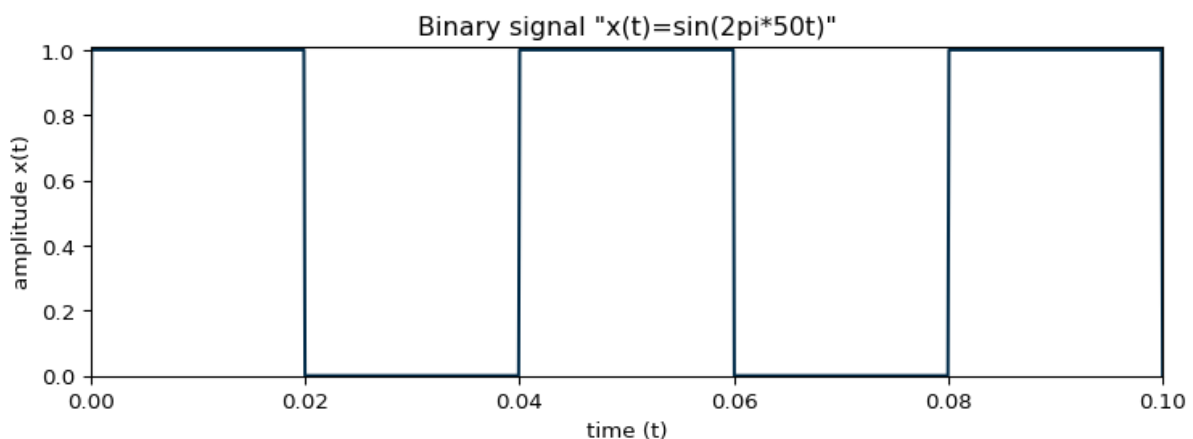


Figure 1.2, a digital signal with two levels: binary representation of $x(t)=\sin(2\pi 50t)$.

The use of digital technology in communication systems continued to grow and evolve throughout the 20th century and into the 21st century, leading to the widespread adoption of digital communication technologies for a variety of applications, including voice and data transmission, mobile telecommunications, and television broadcasting.

The spread of digital

If we take as an example the introduction of digital technology in the broadcasting field, we can see how significant its implications are on the industry (ITU, 2016). Digital broadcasting enables the transmission of high-quality audio and video signals, leading to a clearer and more accurate representation of the original content. Even if it requires more bandwidth than an analogue signal for a comparable quality (Barany, 2007), it allows for the efficient use of the available band, enabling more channels to be transmitted for the same frequency spectrum. Digital broadcasting systems often include two-way communication capabilities, useful to provide personalised services such as Video on Demand (VOD) and interactive television. In terms of cost savings, digital representation can be more convenient, as it eliminates the need for analog-to-digital conversion and reduces the costs associated with signal degradation over long distances.

But the transition to digital broadcasting is just a small part of a bigger revolution, the global shift from traditional to digital media. Considering the past year, on a daily basis, it is estimated that digital users spend an average of 6 hours and 58 minutes online. Although it may seem that the Internet is drawing people away from traditional forms of media, such as television broadcasting, they are still quite popular. The time spent watching television, both broadcast and streaming, is more than the time spent on social media (Datareportal, 2023). In 2021 for example, TV has won the battle for video content, with 3.54 hours of daily consumption on average in Europe (TV key facts, 2022).

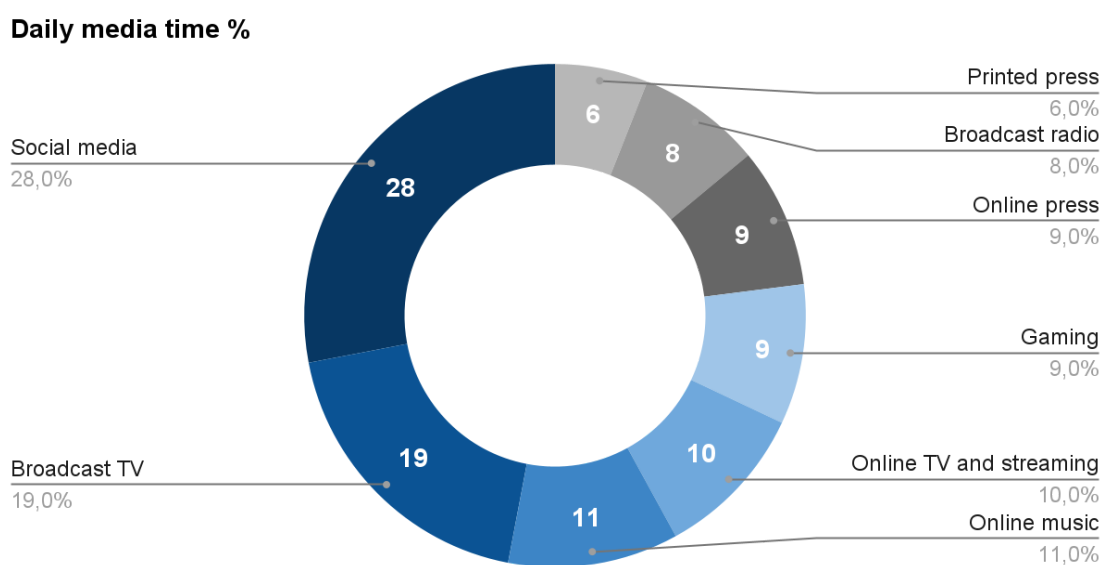


Figure 1.3, percentage of the total time per day typically devoted to media fruition.
(source: GlobalWebIndex, 2019)

While online TV has surpassed the one hour per day mark, linear TV still maintains a steady audience of nearly 2 hours per day (*Figure 1.3*). However, online TV is rapidly gaining ground. The age group of 16-34 year olds are leading the shift towards online consumption, and streaming has become a crucial component of their daily viewing habits. They are spending 80% more time on this service compared to the older consumers. (GlobalWebIndex, 2019)

Online video streaming platforms, including television websites, have managed to attract and retain substantial audiences across different markets. In 2021, the number of worldwide subscribers to online video streaming services reached an impressive 1,060.8 million, with Netflix as the largest player in the market with its over 204 million subscribers (Allam, 2021). As confirmed by Deloitte (2018) in their study on the future of video and television landscape for the next decade, the standard for TV and video distribution will become the Internet Protocol (IP). The transmission will increasingly rely on the IP, which is used to route data over the internet, instead of using specialised equipment to transmit signals over the airwaves, cables or satellite networks as in the traditional broadcast television.

The digitalization and the expansion of online video services have led to an explosion of digital video content, which entails a number of benefits summarised below (Owen, 1982).

- **Better compression ratios.** Digital signals can be compressed more effectively than analog signals, so that it is possible to store more data in a given amount of storage space, reducing the overall costs for data storage. Moreover, they can be efficiently sent over limited bandwidth channels, enabling the transmission of more data in less time. Finally, digital compression algorithms can be designed to be lossless, meaning that the compressed signal can be exactly reconstructed to its original form. This is not possible with analog compression, where some loss of information is inevitable.
- **Management simplicity.** The signal is easy to manipulate and process, as it can take as a value only a binary digit (0 and 1). This makes it possible to perform complex processing tasks with relatively simple circuits and algorithms, reducing the complexity of the overall system.
- **Coding and delivery costs:** Digital signal processing requires less hardware than analog signal processing, and it reduces the cost of production, implementation, but also distribution. Digital signals can be transmitted over longer distances without degradation, eliminating the need for repeated amplification.

- **Resistance to noise.** They offer a high degree of precision and accuracy, as they can be quantized to a very high number of bits. Error detection and correction is easier, as values to reconstruct can be only zeros or ones. This also means greater immunity to noise and interference.
- **Data security.** A wide range of digital cryptographic algorithms have been developed to grant strong encryption techniques and to protect sensitive information from unauthorised access, even in cases where the transmission channel is unreliable. Digital signals can be encrypted and decrypted also using software algorithms, which make it easy to implement those encryption solutions in a variety of applications and devices.
- **Flexibility.** Digital data can be stored, edited, and manipulated with ease, providing a high degree of flexibility and enabling the constant upgrade and development of new and innovative solutions.

Despite the numerous benefits associated with the transition from analog to digital, there are two key implications that are particularly noteworthy. These disadvantages have significant impacts and should be carefully considered when using digital representations (Owen, 1982):

- **Quantization error.** When an analog signal is converted to a digital signal, some information is lost due to quantization error, the difference between the original analog signal and the closest digital approximation.
- **Bandwidth limitations.** Digital signals require more bandwidth than analog signals when transmitting the same amount of information at an equivalent quality. This is because of the Nyquist theory that will be presented in the next section.

The process of converting analog signals into digital signals, commonly called digitization, is achieved through two main steps: sampling and quantization. Sampling refers to the discretization of the analog signal in time, while quantization, on the other hand, refers to the discretization of the intensity of the signal. The combination of these two processes results in a digital representation of the same analog signal, which can be stored, transmitted, and processed more easily than its analog counterpart.

Sampling and quantization

Signal sampling is the process of converting a continuous-time signal into a discrete-time signal (*Figure 1.4*). It is a fundamental step in many digital signal processing systems, including video and audio coding.

Sampling involves measuring the amplitude of the analog signal at regular intervals, resulting in a sequence of discrete values that represent the original information (Hall, 1989). The choice of the sampling frequency, also known as the sampling rate, determines the accuracy of the discrete representation of the continuous-time signal. The sampling rate must be chosen such that it is greater than the highest frequency present in the continuous-time signal. This requirement is known as the Nyquist rate and is given by 2 times the maximum frequency in the signal. An analog signal can be perfectly reconstructed from its samples as long as the Nyquist limit is met. In practice, sampling is often performed using Analog-to-Digital Converters (ADCs), which transform the continuous signal into a sequence of digital values. The quantization process then, which involves mapping the continuous-range of sampled values to a finite set of discrete values, the binary digits, is performed as part of the ADC process.

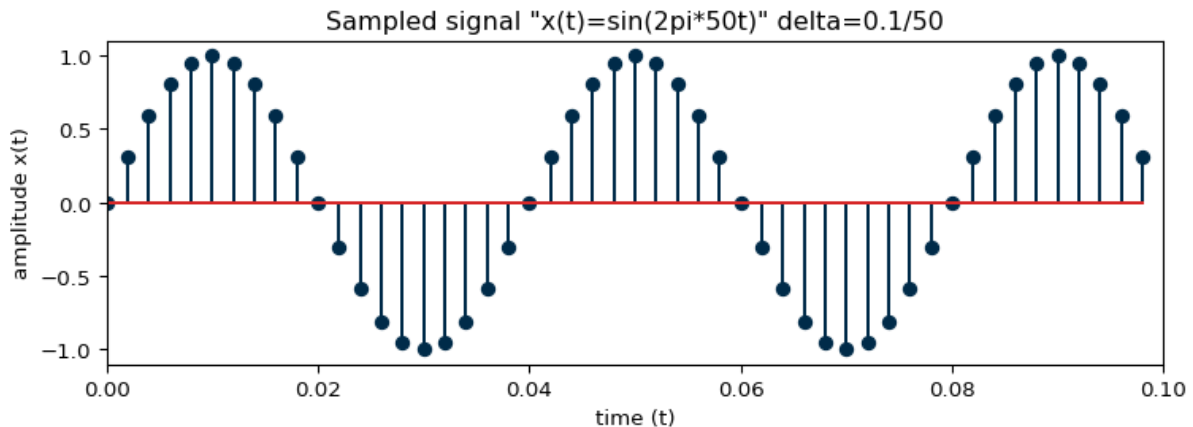


Figure 1.4, discrete-time signal resulting from the sampling of the analog signal $x(t)=\sin(2\pi 50t)$.

Signal quantization is the process of converting a continuous range of values into a finite number of discrete digital values (Jain, 1989). Quantization is important because it allows analog signals to be represented and stored in digital form, a necessary step in signal processing and transmission. The quantizer maps a set of values to the same quantization level (*Figure 1.5*), identified by a unique label, for example an integer number. The choice of the quantization step size and the number of levels determines the resolution of the digital representation and the

amount of quantization error introduced. This error comes from the approximation of the continuous signal by its quantized representation, and it is also called quantization noise. In practice, the quantization process is often optimised to balance the trade-off between the resolution of the digital representation and the amount of noise introduced. In the last step of the analog to digital conversion, each quantization level is assigned a label that identifies the level itself. These labels are then translated into a binary representation, which is the form that digital signals take. The goal of analog to digital conversion is to preserve as much information from the original analog signal as possible while still making it usable in a digital format.

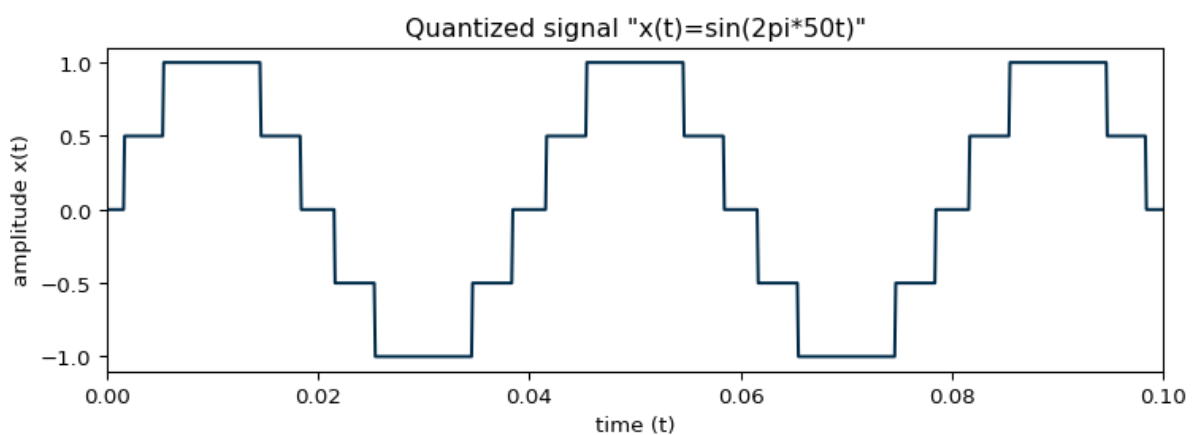


Figure 1.5, sampled signal $x(t)=\sin(2\pi 50t)$ quantized into five levels.

2D digital image

The result of sampling and quantization in image processing is a digital image, which consists of a matrix of pixels, where each pixel is a colour and it is represented by a limited number of bits. The number of rows and columns of this matrix determines the resolution of the image, while the number of bits used for each pixel determines the number of levels that the picture can contain and represent (Young et al., 1998). They both determine the level of detail and quality of the image. In *Figure 1.6*, one can see the results of a downsampling operation, the reduction of the number of samples, or pixels, keeping their size unchanged. *Figure 1.7* instead, shows the consequences of a poor quantization, the so-called banding effect, visible when a low number of bits per level are used. A digital image is easily processed, stored, and transmitted compared to its original analog version, but it may suffer from quantization error, the loss of information during the conversion process. This is the reason why quantization is so important in data compression.

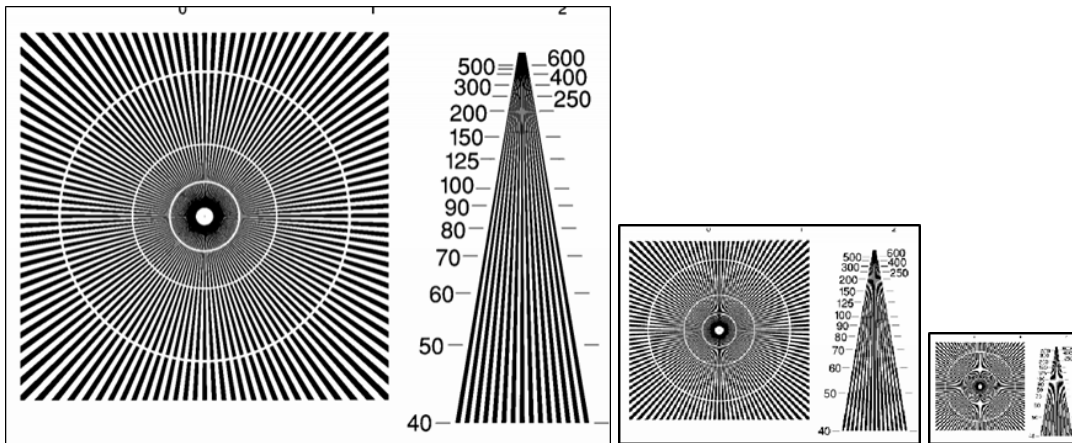


Figure 1.6, the original image on the left has been downsampled by cutting out one pixel out of two. The size of the image is halved. This has been repeated again for the image on the right.



Figure 1.7, the original image on the left has been quantized in its number of grey levels: in order, 128, 16, and 8.

Chroma subsampling

In order to introduce one of the main topics of this chapter, compression, we first present chroma subsampling, a simple type of compression that leverages on the sampling of colour information when reducing the amount of data.

In a grey-level pixel representation, each pixel of the image consists of a single value that represents the intensity or brightness of the cell. This representation is also known as a grayscale or monochrome image, and it is used for example black and white pictures or medical images. In contrast, colour pixel representation uses multiple values for each pixel. The most common colour model in digital images is the RGB (Red, Green, and Blue), in which each pixel is represented by three values that correspond to the intensity of red, green, and blue light. Other colour representations, such as YUV and YCbCr, split the colour in two components, luminance (Y) and chrominance (UV). The Y component represents the brightness

information, while the UV component represents the colour information. YCbCr is a variant of YUV, where the chrominance components are represented by blue chroma (Cb) and red chroma (Cr) instead of UV.

Chroma subsampling is a process used in image and video compression to reduce the amount of data required to represent a digital image or video. It works by reducing the resolution of the chrominance information in the image (Young et al., 1998).

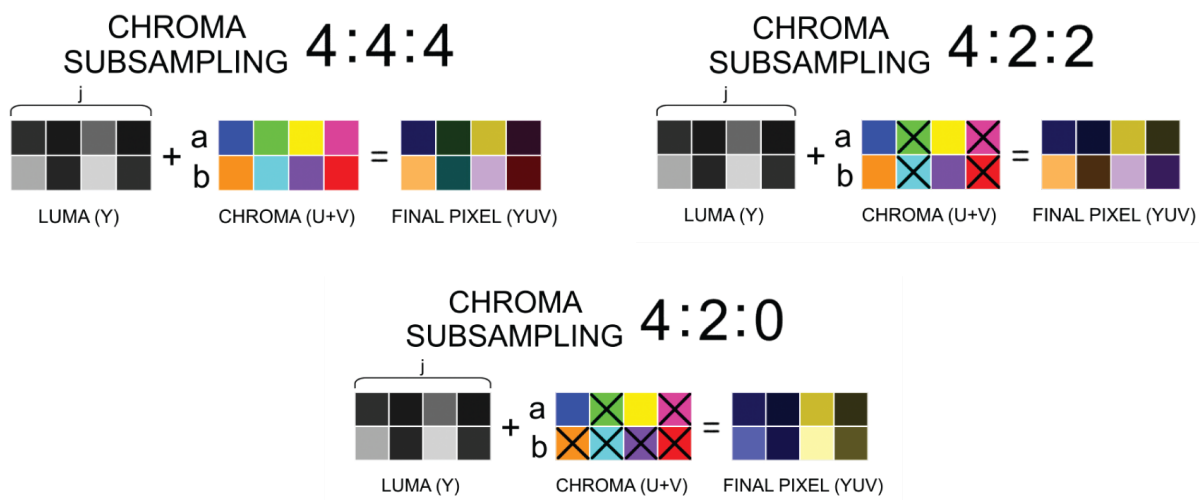


Figure 1.8, examples of colour information removal.

The reason why chroma subsampling works is because the human visual system is less sensitive to colour rather than brightness. It is thus possible to reduce the number of chroma samples required to represent the image without introducing perceptible or visible differences. For example, in the 4:2:2 chroma subsampling two samples are removed from the chrominance information every four samples of the luminance information. In 4:2:0 instead, the chrominance information is sampled at half of the resolution (Figure 1.8). The resulting reduction in the amount of data assures significant savings in storage and transmission costs, without impacting the visual quality of the image or video for our visual system.

1.2 COMPRESSION

The large amount of data in digital video poses significant computational challenges in terms of storage, processing power, and network bandwidth. However, the high degree of redundancy in digital video makes it well-suited for compression, which can significantly reduce these issues. It is crucial to underline a key concept in compression, the trade-off between data size and quality. Higher compression ratios produce smaller size files but also a degradation in the data precision. All this results in a lower visual quality of images and videos. Moreover, the compression process, which is executed during both the encoding and decoding phases, may require heavy computational resources, a fact that should be considered when selecting a compression algorithm. For instance, it would not be efficient for a real-time application with low bandwidth needs to use a computationally intensive algorithm that takes too long to encode and decode the data.

Compression ratio

Uncompressed video files, from now on also referred to as RAW files or RAW videos, contain an enormous amount of data. Here an example:

- Resolution of 720x576 pixels (PAL)
- 25 frames per second
- 8-bit colour depth per pixel
- 4:2:2 YCbCr chroma subsampling scheme

A video with these features generates about 166 Mbps of bitrate (*Equation 1.1*).

$$720 \times 576 \times 25 \times 8 + 2 \times (360 \times 576 \times 25 \times 8) = 165.88 \text{ Mb/s}$$

Equation 1.1, PAL RAW video bitrate requirements.

Whereas, with a native resolution of 1920x1080 typical of the High Definition Television (HDTV) standard, it generates almost one Gigabit of data per second (*Equation 1.2*).

$$1920 \times 1080 \times 25 \times 8 + 2 \times (960 \times 1080 \times 25 \times 8) = 829.44 \text{ Mb/s}$$

Equation 1.2, Full HD RAW video bitrate requirements.

This proves the importance of compression. Compression is performed during the encoding process and it produces a version of the original file that contains fewer bits. The inverse process is called decompression, or simply decoding.

In video processing, compression ratio refers to the amount of reduction in the file size that a video compression algorithm is able to achieve. It is expressed as the ratio of the size of the original file to the size of the compressed one. For example, if the original file is 50 MB and its compressed version is 10 MB, the compression ratio would be $50/10 = 5$. The higher the compression ratio, the smaller the size of the compressed file. It is important to acknowledge nonetheless that the quality of videos may suffer as the compression ratio increases, as compression algorithms will be discarding information in order to achieve the smaller file size. It reminds us again of the quality-rate trade-off already mentioned at the beginning of this section, that will be recalled many times during this paper. Video compression algorithms typically aim at achieving a balance between compression ratio and video quality that is acceptable for a specific application. Indeed, compression ratios may greatly vary from one algorithm to another one.

Compression algorithms

Lossless compression is a technique that decreases the size of the file without discarding any information and allowing in this way a perfect recovery of the original data (Tekalp, 1995). The compressed file will have a smaller size than the original file, but will be an exact copy of it. This type of compression is often used for data files, such as text and spreadsheets, where it is important to maintain the integrity of the data. Here are some common types of lossless compression algorithms used in video encoding.

- **Run-Length encoding (RLE)** is a simple lossless compression algorithm that replaces a series of repeated values with a pair key-value, the single value and a count indicating the number of its repetitions. RLE is best suited for compressing images or video frames with large areas of the same colour.
- **Huffman coding** assigns shorter code words to more frequently occurring symbols and longer code words to less frequent elements. Huffman coding is a type of entropy coding method, which means that it takes into account the probability of occurrence of each symbol and assigns them the binary codes based on their probability.
- **Arithmetic coding** encodes each word as a single fractional number, a value between 0 and 1, instead of a sequence of discrete symbols. This fraction,

like in the previous method, represents the probability of occurrence of each word. The binary coding of fractional numbers in the range $[0,1]$ is proven to be an efficient solution in compression techniques (Howard and Vitter, 1994).

- **Dictionary coding** works by dividing the video data into smaller chunks or blocks, then creating a dictionary of the most frequently occurring patterns. The original data is then replaced with references to the entries in the dictionary, allowing for more efficient storage and transmission of the video (Ignatoski et al., 2020). LZW (Lempel-Ziv-Welch) is the most common type of dictionary coding algorithm.

Lossless compression algorithms are typically less effective than lossy ones in terms of compression ratio and output size, but are recommended when loss of information is not acceptable. On the other hand, lossy compression is a technique that involves discarding some of the data in the original file in order to achieve a smaller file size (Hall, 1995). This type of compression is often used for multimedia files, such as images, videos or sounds, where some of the data can be removed without significantly affecting the overall quality. The compressed file will have a smaller size than the original ones, but will not be identical to them. The level of quality degradation depends on the compression algorithm used and the degree of compression applied. We distinguish two main types of lossy compression techniques used in video encoding:

- **Transform coding** transforms data from the spatial domain into a transformed one, often the frequency domain. One of the most used transforms in video encoding is the Discrete Cosine Transform (DCT). The idea behind transform coding is that transformed data often contains a lot of redundant information that can be more easily removed without affecting the perceived quality of the video. This is the case of the smallest details, hard to encode when looking at close pixels because they rapidly change, but easy to reduce in the transformed domain, where they correspond to high frequencies and so they can be simply eliminated.
- **Motion-compensated prediction** involves predicting future frames based on the motion of pixels from previous frames. The difference between the predicted and actual frames is then quantized and stored as residual data.

Predictive coding and transform-based coding are considered the cornerstone techniques in video compression. Both methods aim at reducing the amount of data required to represent videos, and are widely used in modern video codecs.

1.3 VIDEO CODING

Video encoding refers to the process of representing video data into a digital format so that they can be efficiently transmitted and stored (Bing, 2015). The goal of video encoding, similarly to compression, is to reduce the amount of data required to represent the video while maintaining an acceptable level of quality. The quality of the encoded video depends on the compression algorithm and the parameters used during the encoding process. We distinguish predictive and transform coding.

Predictive coding uses information from previous frames to make predictions about the current frame, and then it encodes only the differences between the estimated values and the real data. Here are some examples.

- **Differential Pulse Code Modulation (DPCM)** converts an analog signal into digital by sampling it at regular intervals and then quantizing the resulting samples. Instead of coding the absolute value of each sample, DPCM works on the difference between consecutive samples (Wiegand and Schwarz, 2011). When this difference is not so wide it leads to a gain in data savings and to a lower requirement in data rate.
- **Adaptive DPCM** is a variant of DPCM that dynamically adjusts the quantization step size based on the signal characteristics, such as signal statistics and noise level. This technique results in a higher compression ratio (Pratheek and Suma, 2013) and reduced quantization noise compared to fixed quantization DPCM due to the intrinsic variable nature of signals.
- **Motion compensation** is a technique used in video compression with the aim of removing temporal redundancies. It involves predicting the future video frames based on past and current frames. The difference between the predicted frame and the actual frame is encoded and transmitted, leading to a much smaller data rate compared to coding entire frames individually (Wien, 2015). This technique is especially useful in video sequences with repetitive patterns, where the difference between frames is relatively small.

Transform-based coding, on the other hand, converts the video data into a different domain, where it is easier to remove redundant information, and then it encodes only the transformed data (Bing, 2015). Here are some examples.

- **Discrete Cosine Transform (DCT)** is a mathematical algorithm that transforms a signal from its time domain into the frequency domain. It is

widely used in image and video compression to reduce the data size while maintaining high visual quality (see next paragraph).

- **Wavelet Transform** is a similar mathematical algorithm to transform a signal from its time domain into the frequency domain. Unlike DCT, wavelet transform provides a multi-resolution representation of the signal.
- **Sub-band decomposition** is a technique used in image and video compression that separates the signal into several frequency bands and processes each band separately, instead of considering the signal and its frequencies in their entirety.
- **Pyramidal decomposition** is a hierarchical sub-band decomposition technique that starts with a high-resolution signal and recursively applies sub-band decomposition until a low-resolution representation of the signal is obtained. This low-resolution representation is then used as a reference for the prediction in lossy compression.

By combining the two techniques, transform and predictive coding, encoders can achieve significant data compression ratios, enabling the efficient storage and transmission of video content. They are two of the main components of video encoders (Sikora, 1997). The video encoding process is typically represented as a block diagram, like in *Figure 1.9*, that provides a visual representation of the various steps involved in the encoding process, and the flow of information through the different stages.

1. In the pre-processing block (In), the video frames in input undergo pre-processing operations such as colour space conversion from RGB to YCbCr and chroma subsampling, to optimise them for the next steps.
2. In the Motion Estimation (ME) and Motion Compensation (MC) blocks, the video encoder predicts the motion of objects among consecutive frames. As further detailed in the next section, the difference between ME and MC is that the former predicts the movement of pixels between frames, while the latter tries to minimise the difference between predicted and original frames.
3. The encoder then applies a mathematical transform to the frame in order to convert it into a frequency-domain representation. One of the traditionally used transforms is the Discrete Cosine Transform (DCT block).
4. The frequency coefficients generated by the transform are then quantized (Q block), resulting in an approximation of their values and thereby in a

reduction of their size. This is the step where most of the information may be lost, and where lossy compression capabilities are maximally exploited.

5. The loss of information in quantization in fact, cannot be recovered during its inverse process (Q^{-1} and I-DCT), which is part of the decoding process but performed also by the encoder side because its resulting coefficients are used in the inverse transform stage to reconstruct the frame for the prediction of the next ones.
6. Finally, the quantized data is compressed using entropy coding techniques (En) such as Huffman coding, variable-length coding, or arithmetic coding, to further size reductions.

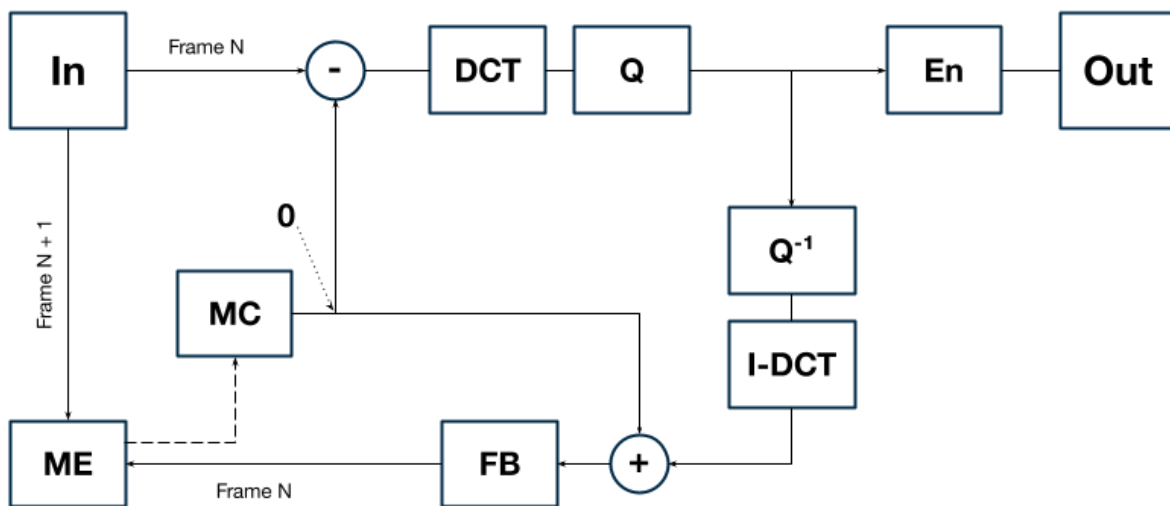


Figure 1.9, basic functional block diagram of a video encoder.

The prediction and the transform operations are always applied to blocks of pixels, called macroblocks, rather than to entire frames. There are more steps involved in modern video codecs in addition to those mentioned. One of them is deblocking filtering, an adaptive low-pass filter, used to remove the sharp edges of macroblocks before they are being used by subsequent frames as predictors. Another improvement introduced over time is the use of variable block partitioning of input frames, in order to capture individual objects in a video scene. All these incremental steps over the basic video encoding structure, introduced more than 20 years ago, has allowed modern video codecs (see *Chapter 1.4*) to be increasingly efficient.

The JPEG process

If we consider single frames, which can be seen as still images, the JPEG (Joint Photographic Experts Group) encoding process for image compression is one of the most widely used methods. The JPEG process (Wallace, 1992) involves a series of specific operations and techniques that are part of the more general video encoding diagram presented above. In this specific context, the block diagram for JPEG still image encoding, becomes a crucial tool for understanding how the compression process works within a single frame. The JPEG block diagram outlined in *Figure 1.10*, is broken down in its fundamental steps in order to understand in depth the function of each component.

1. Colour space transformation from RGB to YCbCr colour space and chroma subsampling to 4:2:0.
2. The raster of pixels is then divided into 8x8 pixels blocks, and each block is transformed using the DCT algorithm. Thus, the image is converted into the frequency domain, where the resulting DCT coefficients represent the different frequency components in the block. The coefficient in the top left corner of the block, the DC coefficient, represents the average brightness of the block so the zero-frequency component of the image, while the other 63 values, the AC coefficients, are the non-zero frequency components. Each coefficient corresponds to the weight of each of the 64 DCT basis functions part of the JPEG standard. They are combined together when decoding to bring back the image to the spatial domain.
3. The quantization of the DCT coefficients reduces the image data's precision and effectively removes the redundant or less noticeable information. The idea behind this is that the human visual system is more sensitive to low frequency information, such as the overall brightness and colour of an image, rather than to high frequencies, such as edges, fine details and textures.
4. Since in DCT most of the information is concentrated in the low frequency components, the DCT coefficients are arranged in a zig-zag order. The reordering starts at the top left corner of the 8x8 DCT coefficient matrix and moves following a zig-zag pattern. This results in a linear sequence of values, where the low frequency coefficients appear first and the high frequency coefficients appear last. This allows the low frequencies to be encoded with more precision, so that it will be easier to entropy encode them.
5. Indeed, for a further compression, the quantized coefficients are entropy coded, using for example the Huffman coding.

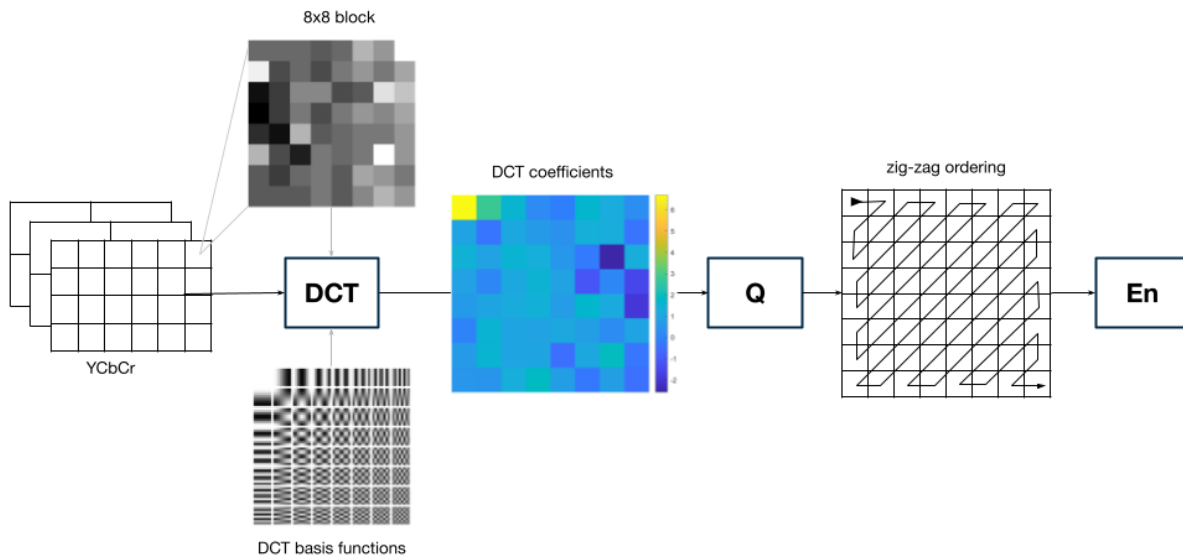


Figure 1.10, the basic JPEG encoding process for still images.

When coding still images, the encoder has the opportunity to optimise the compression for each individual picture. On the other hand, when coding video sequences, the encoder must balance the desire for high compression with the need to maintain high image quality from frame to frame. To address this challenge, video encoding often employs not only intraframe coding but also interframe coding techniques. Intraframe coding regards single frames, treated as still images, whereas interframe coding leverages on temporal redundancy between frames by coding the differences among them.

Interframe coding

Intraframe coding is also known as I-frame coding, and it corresponds to the JPEG process just described. It compresses a single frame independently, without looking at past or future frames. This means that an I-frame can be decoded and displayed alone, without the need of having any information from other frames.

Interframe coding instead, which can be distinguished into P-frame and B-frame coding, takes advantage of the temporal redundancy in video by coding the difference between frames. This means that instead of compressing each frame individually, it considers the difference between frames. P-frames (predictive frames) use information from the previous frame to encode the current frame, while B-frames (bidirectional frames) use information from both previous and future frames to encode the current frame, like in *Figure 1.11*.

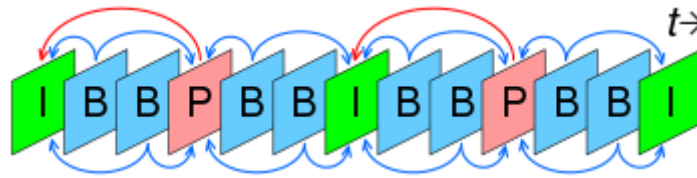


Figure 1.11, dependencies between frames in the IBBBP structure GOP.

The combination of I-frames, P-frames and B-frames represent a Group of Pictures (GOP). The GOP structure defines the number and arrangement of the three types of frame and influences the compression efficiency, the quality of the decoded video and the coding order (ITU, 1994). The coding order is the order in which frames are processed and transformed into a compressed representation, and because of the presence of dependencies among frames, it differs from the display order, the sequence of frames to be played back. In order to be decoded indeed, B-frames need information from a future P-frame, which is displayed after them but has to be coded earlier. For this reason, although they reduce the amount of data needed to be stored because they only contain the differences between frames, B-frames and P-frames lead to an increased complexity and time spent in processing videos.

Before diving into a review of the main currently used video coding standards, it is important to understand how this prediction works and how it is implemented from an algorithmic point of view.

Motion estimation and compensation

Motion estimation and compensation are fundamental techniques used in predictive coding in which the current frame is predicted from its previous ones, but also from future ones.

Motion estimation aims at determining the movement of objects within the image frame-by-frame. This movement can be represented by vectors, which describe the displacement of a block of pixels from one frame to the next one (Wedi, 2003). Based on the direction of the prediction, it is possible to distinguish different types of motion vectors, displayed in *Table 1.1*. Forward vectors try to find the best match between blocks of pixels in the previous and the current frame, so that the differences between them can be encoded. The result of motion estimation is used in motion compensation, which generates a prediction for the current frame foreseeing the future positions of pixels. This reduces the amount of data needed to represent the video stream, as only the changes between frames need to be encoded, rather than all intra-frames in their entirety.









frame	Forward predicted MV of B-frames	Backward predicted MV of B-frames
I	 <p data-bbox="679 573 1007 602">I-frame: no motion vectors</p>	
B		
B		
B		
P	 <p data-bbox="557 1946 1128 1975">P-frame: forward predicted motion vectors of P</p>	

Table 1.1, motion vectors (MV) of 5 consecutive frames with an IBBBP structure.

A key step in motion estimation is to compare the starting block of pixels of the current frame to the corresponding blocks of pixels in the next one, in order to determine the amount and direction of motion. The Sum of Absolute Differences (SAD) is a widely used measure of the similarity between two blocks of pixels, and it is usually performed in a specific search window instead of in the whole frame (Vassiliadis et al., 1998). It evaluates the difference between the two blocks of pixels to help identify the magnitude and direction of the relative motion vector. The search for the best match is not done within the entire frame because this would take too much computational effort, but rather within a rectangular region of the frame of arbitrary size, the search window precisely, centred on the block of pixels in the reference frame. The window defines the range of motion vectors that will be found when searching for the best match. The most similar block is the one that minimises the SAD (*Equation 1.3*), which is calculated by taking the absolute difference between each corresponding pixel of the two blocks, and summing up the values.

$$\min(SAD(p, v)) = \min\left(\sum_{i=1}^N \sum_{j=1}^N \left| f_t(i, j) - f_{t+1}(i, j) \right| \right)$$

Equation 1.3, SAD where p is the block at time t and v the motion vector; N the macroblock size (usually 8) and i, j the horizontal and vertical pixel indexes.

1.4 CODING STANDARDS

Video coding standards are standardised specifications that define the process of converting analog video signals into digital format. They are essential for ensuring compatibility and interoperability of video systems and devices. The purpose of video coding standards, like the one one encoders, is to minimise the amount of data required to represent video content while maintaining its quality, using all the techniques presented above.

Video coding standards typically specify the decoding process, which includes the decompression of video data and the representation of the compressed data in a standardised format (Wien, 2015). Compressed data needs to be decoded by the receiving device in order to be displayed. Nowadays, there are several video coding standards currently in use, with H.264/AVC and H.265/HEVC being two of the most widely used (*Figure 1.12*). The diagram shows the encoders most used by media companies for VOD contents. Despite its gradual decline, which is confirmed by the trend of the previous years, H.264 continues to be the most commonly used codec, also in the case of live streaming (Bitmoving, 2022).

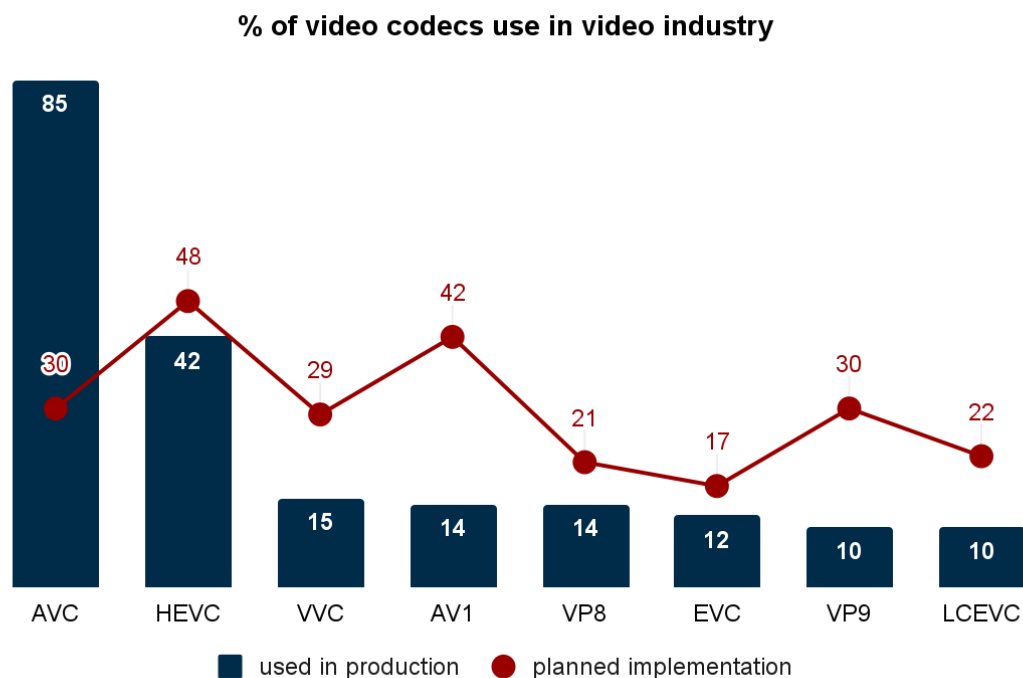


Figure 1.12, the use of video codecs in the VOD streaming industry.

These standards are designed to provide high compression efficiency, which is important for applications such as video transmission over limited bandwidth networks, video storage, and video processing. Practically, a video coding standard is a document that outlines the format and decoding process for compressed video. The scope of the standardisation includes only the decoding process, by imposing restrictions on the output bitstream and its syntax, rather than defining the design of the encoder itself. This allows a great level of flexibility and room for innovation in the implementation specifications, for example to optimise encoders for specific applications. Even though some defined tools may be optional, all decoders that follow the standard must support a minimum set of tools in order to correctly decode the bitstream and produce a compliant output. For this reason, video coding standards are approved and adopted by standardisation organisations such as the International Telecommunications Union (ITU) and the International Standardization Organization (ISO).

H.264, H.265, VP9, and AV1

H.264. The most used standard indeed was developed by ITU-T Video Coding Experts Group (VCEG) and the ISO Moving Picture Experts Group (MPEG) as part of their Joint Collaborative Team on Video Coding (JCT-VC) initiative (Wiegand et al., 2003). The growing need for higher compression of moving pictures for various applications and a wide variety of network conditions led to a call for proposals issued in 1998 by VCEG, and to the later creation of the JCT. The Advanced Video Coding (AVC), also known as H.264, was released in 2003 and quickly became the most widely used video compression standard, in particular for applications such as online video streaming and Blu-ray discs. One of the main achievements of AVC is its high compression efficiency, which allowed for the first time for high-quality video to be transmitted over limited bandwidth networks. Some of the introduced innovations were the variable block-size motion compensation, the use of more than one frame to predict the movement in the incoming frame, the quarter-pixel motion vector precision, the introduction of an adaptive deblocking filtering, and the use of arithmetic entropy coding. Moreover, the decoupling of the Video Coding Layer (VCL) and the newly introduced Network Abstraction Layer (NAL), which allowed greater customization in carrying the video content through different and specific networks. With the main work on the encoder finished, the Joint Video Team was closed after its last meeting in November 2009.

H.265. 2010 saw the new formation of the JVT-VC group under the request of a new coding standard, due to the emergence of UHD video applications and the

increasing deployment of mobile device services at the time. The project call was launched with the aim of addressing two key issues of AVC: higher video resolution such as 2K and 4K, and increased use of parallel processing architectures (Sullivan et al., 2012). HEVC, also known as H.265, was released in 2013 and it is in fact the successor of AVC. It offers improved compression efficiency with a smaller bitrate, in the order of -50%, for the same level of video quality. The video coding layer of HEVC uses the same hybrid approach of AVC and all MPEG video compression standards: interframe and intraframe prediction, and 2-D transform coding.

The core of the coding layer in previous standards was the macroblock, made of a 16×16 block of luma samples and, in the usual case of 4:2:0 colour sampling, two corresponding 8×8 blocks of chroma samples. Whereas, the analogous structure in HEVC is the Coding Tree Unit (CTU), with a variable size, even larger than a traditional macroblock, determined by the encoder itself. In addition, in order to enhance the parallel processing capability and to adapt data structuring for packetization (see *Chapter 1.6*), two new features were introduced. The first one is the option to partition frames into rectangular areas called tiles, which are independently decodable regions encoded with common header information. The second one is wavefront parallel processing (WPP). When enabled, a slice is divided into rows of CTUs, and each row can be processed in parallel.

VP9. In late 2011 also Google undertook a project to develop their own open-source video codec, and decided to start from the existing VP8 (Mukherjee et al., 2013). VP9 was finalised in June 2013 and released the same year. It is an open source video compression standard, widely supported on a variety of platforms and devices, and used by many online video streaming services, such as YouTube. It was developed by Google under their larger WebM project, aimed at bringing the new standard inside of the HTML5 environment. WebM is part of the Alliance for Open Media (AOM), an open source and royalty-free standard community with the main focus on supporting internet-based video consumption. They also fostered the development of another new video standard, AV1, described in the next section.

A big part of the coding efficiency improvements achieved by VP9, in the order of 50% less bitrate than VP8, can be attributed to the incorporation of larger prediction block sizes. VP9 indeed, introduces the concept of super-blocks (SB) of size up to 64×64, but at the same time it enables the blocks to be broken down into smaller sub-blocks down to 4×4 pixels thanks to a recursive decomposition algorithm. The super-block 64×64 is more or less like the CTU in HEVC, with the difference that it is possible to further splitted the super-block into smaller prediction sub-blocks.

AV1. AOM released its first and new video compression standard in 2018 and, like all standards, it is designed to be highly efficient, with improved compression efficiency over the VP9 and HEVC competitors. AV1 is one of the most recent open-source standards, but it is already widely supported by many major companies in the video industry, and it is becoming more and more popular in online video streaming applications (Han et al., 2021). The AV1 format is already integrated in many web platforms and multiple web based video service providers like YouTube or Netflix, which have begun to provide streaming services in AV1 on a large scale (Guo et al., 2021). Since Google is part of AOM, it is considered the successor of VP9, with a compression gain of about 30% compared to it.

AV1 was developed to overcome licensing problems and hardware feasibility limits of existing standards. At this advanced stage in the technological development of video coding solutions, any improvement in the compression efficiency requires an inevitable increase of the computational complexity. Therefore, during the development of AV1, the hardware implementation of the various coding tools was carefully designed so that the resulting standard was likely to be widely adopted also in small lightweight devices such as smartphones.

Codecs evolution

AVC, HEVC, VP9, and AV1 are the current four most used in the industry and the one chosen for the testing and assessment phase of this work, but the development of video coding standards is always an ongoing process. The impressive consumption of multimedia content in different consumer devices, from mobile displays to smart TVs, from gaming consoles to virtual reality headsets, requires coding algorithms to constantly reduce bandwidth and storage footprint while increasing the video quality. Moreover, nowadays the mass market demands contents with higher resolutions, higher dynamic range, and higher frame rates.

To continue facing these challenges MPEG and VCEG have again joined their resources in 2015 (Bross et al., 2021) to start an exploratory phase to design a new video coding standard, the Versatile Video Coding (VVC), targeting yet again a 50% of compression gain over HEVC. The standard was finalised in July 2020. In the meanwhile, parallel projects have been worked out, like Essential Video Coding (EVC) and Low Complexity Enhancement Video Coding (LCEVC). In addition to considering the growing demand for higher video quality and lower bandwidth requirements, they are designed to address two main issues of existing standards: the legal disputes on patents that delay and complicate their release, and the exponential increase of their computational complexity. The first problem is faced in

EVC by using a royalty-free baseline profile (Choi et al., 2020). LCEVC instead deals with limited hardware resources, like mobile devices and other low-powered devices, with a new approach called enhancement layers (Battista et al., 2022). According to it, small pieces of data can be added to the base video stream to increase its quality with no critical additional costs.

The future of video coding is incredibly promising and exciting, as new technologies and techniques are being developed and implemented to improve the quality, efficiency, and accessibility of video contents (Punchihewa and Bailey, 2020). One particular area of interest is the use of artificial intelligence and machine learning to optimise video coding and enhance the user experience. This includes the development of intelligent algorithms that can analyse video content and automatically adjust encoding parameters. Furthermore, MPEG research areas look forward to immersive video, which offers a more immersive and interactive viewing experience by enabling users to explore a 360-degree environment, and video coding for machines, like machine learning and computer vision algorithms, leading frontiers of autonomous vehicles and robotics applications.

1.5 QUALITY

Video quality refers to the overall perception of the visual and audible attributes of a video, including its resolution, sharpness, colour accuracy, contrast, brightness, frame rate, and audio quality. It encompasses both objective measures, such as pixel differences, and subjective measures, for example the observer's opinion on perceived quality and enjoyment of a video. Quality itself is a subjective property, influenced by various factors including the viewer's expectations, the viewing environment, and the intended use of the video. As a result, it is often challenging to accurately quantify and compare the quality of different videos for a visual system that varies so much from person to person. The characteristics of the human visual system are complex and not uniquely defined. This problem is aggravated in video coding because the addition of the temporal domain, in contrast to still-picture coding, further complicates the issue. In practice, highly imperfect distortion models are used in most actual comparisons, such as the Sum of Squared Differences (SSD), the Mean Squared Error (MSE) or the Peak Signal-to-Noise Ratio (PSNR).

Nonetheless, the rate-distortion optimization requires the ability to measure distortion, and the more precise the better its optimization performances. Clearly, since in our case the sources are encoded and transmitted to be ultimately played back or displayed for a human observer, a distortion measure should be consistent with what the subject observes. Thus, distortion measures that correlate well with the perceptual impact of the loss should be favoured, one above all VMAF. But the issue of what distortion measures are more suitable has been the object of continuing study for as long as digital representation of video signals has been considered.

Quality metrics

We distinguish subjective from objective quality metrics. Whilst the former rely on human perception and judgement, the latter use mathematical algorithms to evaluate the quality of the content.

$$MOS = \frac{\sum_{n=1}^N R_n}{N}$$

Equation 1.4, MOS of a single stimulus, where N is the number of testers and R the rating.

Subjective video quality tests are typically performed by recruiting a group of testers who are asked to watch some test sequences and rate their quality based on their own perception. In order to ensure the reliability of the test, the selected sample of viewers usually represents a heterogeneous group of people with different ages, genders, and backgrounds. Instead, in order to ensure consistency, video contents are presented in controlled viewing conditions, like a dimly lit room, and on calibrated displays. According to ITU-T recommendations (2008), testers are asked to rate what they see using one of the standardised testing procedures in *Table 1.2*. The scores are then averaged using the Mean Opinion Score (MOS) metric (*Equation 1.4*), a numerical value ranging from 1 to 5, where 5 represents the highest quality, or its differential version (DMOS) calculated as the difference between the two means in case of double stimulus.

Single-Stimulus (SS)	Double Stimulus (DS)
<ul style="list-style-type: none"> • Absolute Category Rating (ACR) • ACR with Hidden Reference (ACR-HR) • Single Stimulus Continuous Quality Rating (SSCQE) 	<ul style="list-style-type: none"> • Double Stimulus Continuous Quality Scale (DSCQS) • Degradation Category Rating (DCR) • Pair Comparison (PC)

Table 1.2, ITU-T subjective assessment methodologies.

In *Table 1.2*, in ACR, test sequences are presented one at a time and rated independently on a category scale. Common labels on the scale are *bad*, *poor*, *fair*, *good*, and *excellent*, which can be translated into the values 1, 2, 3, 4 and 5 when calculating the MOS. The same happens in the ACR-HR method, but an original reference version of each test sequence is shown together with any other distorted stimulus, without telling the subjects of its presence, hence the acronym Hidden Reference (HR). The results are then calculated as the difference between the reference and the distorted versions scores. But ratings may not be made only of discrete values. In SSCQE for example, the sequence is rated over time using a slider of continuous values. Samples are taken at regular intervals, resulting in a curve rather than a single quality rating. Another is DSCQS, where participants are presented with pairs of sequences, the reference and an impaired version, and are asked to rate both on a continuous scale. Similarly, in DCR, test sequences are displayed in pairs, but participants know in advance that the first one is the reference, so they rate the second one relative to the original. Finally, PC consists in presenting pairs of videos sequentially, impaired at different levels of distortion.

Although subjective assessment is the most natural way to evaluate video content, the organisation of subjective test sessions is expensive and time-consuming, and results are prone to observer variability, making it difficult sometimes to obtain consistent and reliable results. Therefore, objective video quality metrics have become an essential tool in quantifying the quality of a video signal automatically and mathematically, without the need for human intervention. Objective evaluation models can be classified by the level of reference information available when performing the assessment (Chikkerur, 2011). Together with the most used objective quality metrics, they are listed in *Table 1.3*.

Objective methods	Objective metrics
<ul style="list-style-type: none"> ● Full Reference (FR) ● Reduced Reference (RR) ● No-Reference (NR) 	<ul style="list-style-type: none"> ● Sum of Squared Differences (SSD) and Mean Squared Error (MSE) ● Peak Signal-to-Noise Ratio (PSNR) ● Structural Similarity Index (SSIM) and Multi Scale SSIM (MS-SSIM) ● Video Multimethod Assessment Fusion (VMAF)

Table 1.3, objective assessment methods and metrics.

In *Table 1.3*, FR methods compare the quality of the original uncompressed video against its distorted versions so they require the availability of the source reference. They are the most accurate but cannot be used in specific applications, for example the ones on the receiving side, which do not have reference videos. When just a part of the original video is available, for example in transmission over limited bandwidth channels, the RR models are able to use a subset of information from the original video signal for the assessment. NR metrics instead, evaluate video quality without any reference to the reference source, by analysing particular features or artefacts present in the distorted video, such as blockiness or blurriness.

SSD and MSE. The SSD compares the original and distorted video frames by calculating the sum of squared pixel differences among them (*Equation 1.5*), while the MSE computes the average of those squared differences (*Equation 1.6*). They are widely used due to their simplicity, but they do not consider the perceptual characteristics of human vision, and their scores are very far from a subjective analysis.

$$SSD = \sum_{i=1}^N \sum_{j=1}^M \left(f_{ref}(i,j) - f_{dist}(i,j) \right)^2$$

Equation 1.5, the SSD formula, where NxM is the frame dimension, and i,j the pixels indexes.

$$MSE = \frac{1}{NM} SSD$$

Equation 1.6, the MSE formula.

PSNR. Traditionally, PSNR is the most used objective quality metric because of its easy computation and good correlation with subjective quality ratings. It measures the ratio between the maximum possible pixel value and the average squared pixel difference between original and distorted video frames (Equation 1.7). However, it has limitations when it comes to taking into account the perceptual characteristics of human vision, such as sensitivity to spatial and temporal details, contrast, and colour. In these cases, it may not precisely predict the perceived quality. Moreover, it is highly sensitive to compression artefacts, which makes it even more inaccurate in translating perceptual stimuli.

$$PSNR = 20 * \log_{10} \left(\frac{\max(f_{ref})}{\sqrt{MSE}} \right)$$

Equation 1.7, one of the PSNR formulas.

SSIM and MS-SSIM. Both are based on the idea that human visual perception is more sensitive to changes in the structural information within windows of pixels rather than to single pixel values. SSIM compares the structural similarity between the original and distorted video frames, taking into account three components: luminance, contrast, and structure of the image, which represent the spatial dependencies among close pixels. An advanced extension of SSIM is MS-SSIM. It works by decomposing the reference and distorted images into several scales, and by producing a pyramid of images at different resolutions, for each of which the SSIM score is calculated and then averaged. Although computationally more complex, the multi-scale version of SSIM is proven to provide a more accurate assessment of perceptual quality for a wider range of video content (Wang, 2004).

VMAF. In 2016 Netflix announced a new quality metric (Li, 2016) that combines multiple objective quality features using machine learning techniques to better

correlate with human subjective ratings. These features are fused together producing a score in the interval $[0,100]$ where 0 indicates very poor quality and 100 nearly perfect adherence with the reference quality. VMAF takes into account three main perceptual measures:

- Detail Loss Metric (DLM). It measures the loss of fine details when compressing video sources. It can be considered as an extension to MS-SSIM, since it calculates the difference between the high-frequency components of the reference and the distorted videos using a multi-scale decomposition technique. The high-frequency components represent the fine details in the video, which are more sensitive to compression and so to distortions.
- Visual Information Fidelity (VIF). While DLM focuses specifically on the loss of detail, VIF provides a more comprehensive measure of video quality taking into account several key aspects of the human sensitivity to visual information. It is based on the premise that quality is related to information fidelity loss, a measure of how much the distorted video preserves the key visual features and details of the reference video.
- Temporal Indicator (TI). The motion information of the video is extracted by computing the optical flow between consecutive frames (Batsi and Kondi, 2020), a simple mean of the absolute pixel difference in the luminance component between the current frame and the previous one. This is a simple measure of the apparent motion of objects in the video.

VMAF uses machine learning techniques to learn the relationship between these metrics and human perception. It is computed on a per-frame basis and the resulting score for each frame is a weighted average of the individual metrics and feature scores, where the weights are determined by a fusion model trained on a large dataset of reference and distorted videos (Li, 2016). One of its main advantages initially was the possibility to train the machine-learning model on the huge catalogue of sequences owned by Netflix, the developer. Afterwards, Netflix decided to release an open-source implementation, which allows other content providers and researchers to use the metric. This has helped to establish VMAF as a widely accepted standard for video quality assessment. VMAF has been shown to outperform other objective quality metrics, like PSNR and SSIM, in terms of correlation with human subjective ratings and visual perception (Antsiferova et al., 2021). It is also more robust to different types of distortions, including compression artefacts and noise, making it suitable across a wide range of video content.

Bitrate control

Rate control is an essential component of next-generation video coding algorithms because it ensures compressed video streams meet specific bitrate requirements. It makes them compatible with a wide range of network and device configurations and it helps in this way in the stabilisation of the video stream transmission. All coding standards described in *Chapter 1.4* use sophisticated compression techniques to achieve higher compression efficiency and better video quality. However, these compression techniques can also result in highly variable bitrate depending on content complexities, which can make it difficult to deliver high-quality video over networks with limited bandwidth or to devices with limited processing power. Therefore, rate control techniques are used to regulate the bitrate of the compressed video stream and to ensure that it remains within a predefined range. Moreover, they are essential for our purposes, when optimising the quality of compressed videos by controlling the trade-off between compression efficiency and video quality, because they ensure that the compressed video meets a specific rate target or minimise the bitrate given a quality target. Hence, a key problem in high-compression video coding is the operational control of the encoder.

The first and most basic technique used in rate control is the constant Quantization Parameter (QP), which sets a fixed quantization step for each video frame of a video (see *Chapter 1.3* and *Figure 1.9*). In constant QP, a fixed QP value is applied to all frames, for example $Q=18$, resulting in a constant level of compression across the entire sequence (Zeng et al., 2022). This can be useful in applications where the video bitrate has to remain relatively constant over time. However, it is considered an obsolete technique because it results in some shots or frames being over, or under, compressed than necessary, depending on the complexity of the content in each frame.

Constant Bitrate (CBR), Variable Bitrate (VBR), and Constant Rate Factor (CRF) are different types of rate control algorithms used in video encoding to regulate the data rate of the encoded video in a more sophisticated way. CBR aims to maintain a constant bitrate throughout the entire video. The encoder is designed to keep the output data rate within a specified range by adjusting the encoding parameters in real-time. VBR, on the other hand, dynamically adjusts the data rate in response to changes in video complexity. VBR can produce a higher quality output than CBR, especially in scenes with high video complexity, by allocating more bits to the complex parts of the video. CRF stands for Constant Rate Factor, and despite its name, it is a variable bitrate encoding mode. Unlike VBR, it is entirely focused on the output quality, and even though it is possible to predict more or less the final output bitrate for a given CRF factor, this method does not allow to control and

adjust bitrate in a predetermined way (Javadtalab, 2011). It works by modifying the encoding bitrate dynamically during the encoding process to achieve a consistent level of visual quality. On the other hand, in Fixed CRF, the rate factor is maintained fixed throughout the encoding process. The target quality level is set in advance and the encoder tries to maintain that quality level for the whole video. Unlike CRF, the encoded bitrate cannot fluctuate dynamically during the encoding process. CRF has been selected as the rate control method for the encoding processes described in *Chapter 2*.

1.6 DELIVERY

Digital video delivery is the process of sending video contents over a channel, more and more often through the internet network, to end-devices such as computers, smartphones, or televisions. Thanks to significant progress in digital data compression technologies, digital delivery is gradually replacing traditional analog TV and radio broadcasting. In the meantime, again due to the widespread adoption of internet-connected devices and the increasing reliance on digital services and online communication, the use of the Internet Protocol (IP) as the underlying technology for internet communications is increasing at a fast pace. Global IP traffic is expected to triple from 2020 to 2024, with an annual growth rate of 27% over the same period (Cisco, 2020). Moreover, especially in the last two decades, these trends confirm another paradigm shift in users' social habits, the progressive phasing out of broadcasting towards multimedia streaming (Hwang, 2009). The possibility to use the secure, peer-to-peer TCP/IP packet-based communication protocols opened tremendous opportunities for the deployment of video streaming services that consumers embraced and continue to adapt all over the world.

There are some fundamental differences of streaming over traditional broadcast TV. Instead of dealing with the problem of missing bits due to transmission errors, streaming applications need to devise strategies to maintain the flow active also in case of reduced internet connectivity and packet loss or delay. As long as receiving display devices are connected and receive data, contents can be accessible to watch anytime and anywhere, without the need for users to store or archive them. Sometimes however, streaming providers may decide to include the possibility to download video contents as an additional service. During streaming playback, applications have the ability to dynamically switch bitrate and quality in real-time, allowing for a nearly-optimal experience and performance of the service. Decisions on bitrate changes can be taken from the client-side of applications, without impacting other people watching the same or different contents. Moreover, in case of on-demand video, streaming guarantees a long processing time in the preparation of encoded bitstreams, resulting in a significant opportunity to produce higher efficient streams than those offered by live streaming and traditional TV broadcasters. Among the others, it offers multiple representations of the same video content for custom service differentiation, the use of different codecs to different clients, based on their decoding capabilities, different resolutions and even different bitrates depending on many parameters such as network bandwidth. These and more are the reasons why the entire digital video world is moving towards an all-IP telecommunications infrastructure.

IP protocol

The Internet Protocol (IP) is a fundamental protocol in the TCP/IP Internet protocol suite (Figure 1.13), which provides the basis for the communication over the Internet (Tenenbaum and Wetherall, 2013). It operates at the network layer and is responsible for packet forwarding and routing. The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) are two transport-layer protocols that operate on top of IP. TCP provides a connection-oriented service, which means that it establishes a dedicated communication channel between two hosts before transmitting data. This enables a reliable and ordered transmission of data, with error detection and correction mechanisms to ensure that information is delivered correctly. TCP also provides flow control and congestion control mechanisms to manage data transfer rates and prevent network congestion. UDP, on the other hand, is a connectionless protocol that provides a best-effort service. It does not establish a dedicated communication channel and it does not provide any error correction or flow control mechanism. Instead, it simply transmits data as quickly as possible, with no guarantees on delivery or order. This makes UDP ideal for real-time applications such as live video streaming and online gaming, where speed is more important than precision. IP, for its part, is a connectionless and packet-switched protocol, meaning that data is divided, or packetized, into small units called packets and transmitted over the network in a non-continuous manner. Each packet contains a header and a payload, with information such as the source and destination IP addresses, the protocol used, and other routing information. Once arrived at the destination host, the same will be responsible for reassembling data in a correct way. Thus, like UDP, also IP provides a best-effort service, without guaranteeing an ordered delivery or the delivery of every packet, and delegating the role to higher level protocols.

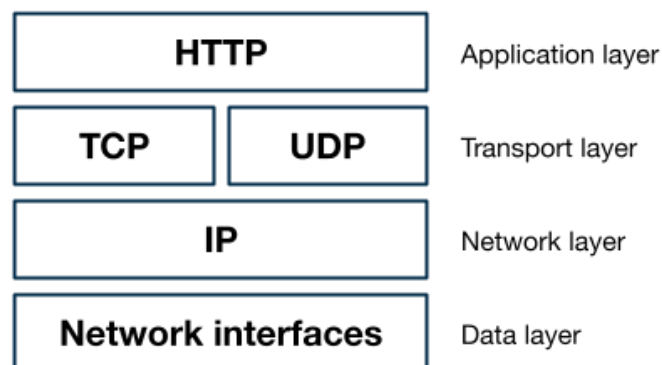


Figure 1.13, part of the protocols of the TCP/IP stack.

During transmission indeed, packets may encounter delays or errors, especially due to network congestion or failures, leading to packet loss or to the need for retransmission. To deal with this, and to compensate for IP behaviour, the TCP/IP envisages various mechanisms and protocols, and the Hypertext Transfer Protocol (HTTP) is one of them. HTTP is a protocol used for transmitting data over the internet, designed in particular to transmit web pages and other resources from web servers to clients (*Figure 1.14*). When a client, such as a web browser, wants to retrieve a resource, typically a web page, from a web server, it sends an HTTP request to it. This request is typically sent over TCP/IP, which breaks the request down into packets and sends them over the internet. Each packet contains the necessary addressing and routing information to ensure that it reaches its destination. When the server receives the request, it processes it and generates an HTTP response, which is sent back to the client again in the form of packets containing the requested information.



Figure 1.14, HTTP communication process.

Throughout this process, TCP/IP structure ensures that packets are transmitted reliably and in the correct order, even if they are sent over multiple routes and through multiple networks. But in addition to packet loss and delays, errors might also occur in IP due to factors such as incorrect routing or address resolution. These errors lead to network performance issues or even security vulnerabilities. Modern IP implementations however have evolved to include robust error detection and correction mechanisms, as well as strong encryption techniques to protect against unauthorised access.

Video streaming

Multimedia networking has become increasingly important in recent years, as more and more people consume multimedia content over the Internet. IP provides the foundational connectivity also for multimedia networking, and also for streaming,

the most popular method for delivering multimedia content over the Internet. We distinguish live from on-demand streaming.

Live streaming refers to the delivery of multimedia content in real-time, when it happens, with little or no delay. In on-demand streaming, on the other hand, resources are pre-recorded and stored, allowing viewers to access them at any time. This type of streaming is commonly used for movies and TV shows, for which viewers can choose what they want to watch and when to watch it. On-demand contents can be delivered using a variety of formats and protocols, including HTTP-based protocols like MPEG-DASH and HLS, which allow the delivery of a large amount of data, as video information, in an efficient manner, mainly thanks to adaptive bitrate streaming techniques.

Adaptive streaming is a technology that consists in automatically adjusting in real-time the quality of sent video contents based on the viewer's network conditions, in order to match the available bandwidth (Akhshabi et al., 2011). This helps in reaching a seamless viewing experience even when network conditions change. In case of on-demand delivery moreover, video content has the opportunity to be pre-processed. Thus, it is typically broken down into small segments, each of which is encoded at multiple bitrates and resolutions, forming the so-called bitrate ladder (Katsenou et al., 2021). The resulting encoded segments are then stored on the server, so that all their different representations are ready to be delivered according to custom needs. When a user starts watching a video, the adaptive streaming client initially requests the lowest bitrate stream, which ensures that the video starts playing quickly without buffering. As the video plays, the client continuously monitors the network and device conditions, and based on this information, it decides whether to request a higher or lower bitrate stream to the server. This process is commonly known as bitstream switching.

Chunked encoding is the just mentioned process of breaking down video content into smaller segments, or chunks precisely, delivered to the client device in a specific order. This method allows for the transfer of large amounts of data in a more efficient and reliable way, because it reduces the risk of network congestion and makes it possible for the client to start playing the video content immediately, even before the entire file has been downloaded. Each chunk is a portion of the video of fixed duration, usually of about 30 seconds and up to a maximum of three minutes. However, this is not the best way of splitting a sequence into segments, because it has not any significant link to its content. It is proven that modifying bitrate and quality during scene changes is less noticeable than changing them in the middle of a scene (Manohara et al, 2018). This is where shot-based encoding comes into play. According to this method, the division into segments should be in

time with noticeable changes in the visual content of the video, so primarily when shots change. Shots are a continuous sequence of frames captured by a single camera in a particular time period (Abdulhussain, 2018), ranging from a shot boundary to the next one, or from the transition between two consecutive shots to the next one.

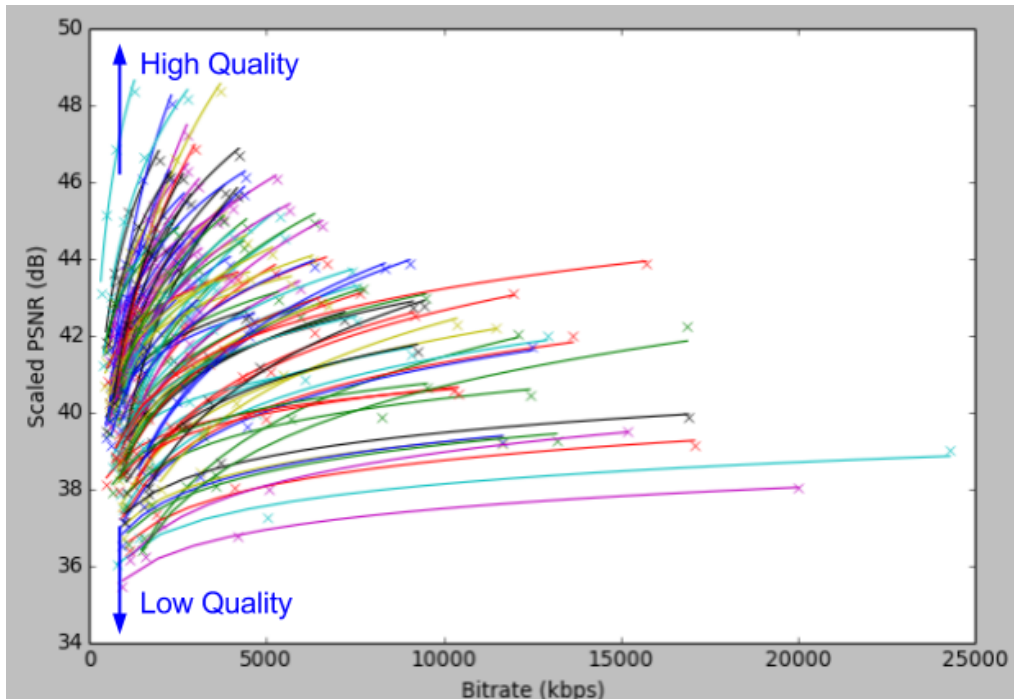


Figure 1.15, quality and rate diversity according to titles complexity (source: Netflix).

The segmentation of video content into packets for HTTP delivery is the idea behind HTTP Adaptive Streaming (HAS), which has emerged as the dominant standard for delivering video over the Internet continuously adapting quality to end-users' network conditions and device capabilities. Also in HAS the bitrate ladder is made of all shots encoded at various bitrate and resolution combinations. Traditionally, however, a fixed bitrate ladder is used for all video contents (Menon et al., 2022), despite their diverse characteristics or network conditions. Indeed, some resources may achieve high quality at lower bitrates, while others may not achieve high quality even at higher bitrates (Figure 1.15). Therefore, it makes no sense to use the same encoding ladder for all of them, as it is not practical and can lead to poor and inefficient coding. To address this issue, Netflix proposed another extension of adaptive streaming, the per-title encoding (De Cock et al., 2016), to further improve the video quality and reduce the amount of bandwidth required for delivery. It analyses and takes into account the specific complexity of each video title and adjusts the bitrate ladder and the encoding parameters accordingly.

Further optimisation

As we have seen so far, delivering high-quality video content over the Internet is not without challenges, which includes the necessity to consider all costs related to the provision of streaming services. Streaming requires significant computing power to encode the video content in so many formats and versions, as well as a global distribution infrastructure to ensure the best user experience. Content Delivery Networks (CDNs) play a critical role in moving video resources over the Internet by replicating them across multiple servers distributed globally. Indeed, when customers stream video content, the CDN servers nearest to their location will be responsible for delivering them the content, ensuring faster and more reliable content delivery. The costs for this kind of service are based on the amount of content delivered, stored and processed, the geographic distribution of the end-users, or the amount of traffic handled by the CDN servers. Moreover, the energy costs associated with delivering video content over the Internet are significant (Gådin et al., 2022), with a large portion of energy consumption caused by data centres and fixed access networking, the physical connection infrastructure.

This is why the optimization of encoding parameters is a crucial step in the digital video delivery process. There are several methods for optimising encoding parameters, including using predetermined quality metrics, automating the process using machine learning algorithms, or a combination of both.

If we come back to the principles of adaptive streaming discussed above, it becomes apparent that the challenge for service providers is not limited to encoding each title at a single quality or bitrate target. Instead, they must create a collection of encodings not just for individual shots but rather for the entire sequences, that encompasses the whole range of qualities and bitrates that users may require depending on their network conditions. This, together with a number of key observations summarised in the following chapter, has led to the development of a new and innovative video encoding optimization framework called Dynamic Optimizer.

2. DYNAMIC OPTIMIZATION

Nowadays, common contents available on the video market are typically composed of sequences of short shots, usually from 2 to 10 seconds long. Often they have no relation with those joined to them, so that they can be treated independently by video encoders as fundamental coding units. From now on, we will refer to shots interchangeably also as coding units. From a perceptual point of view, our visual system is highly sensitive to visual disruptions that often occur at the boundaries between shots, like changes in camera perspective, scenic elements, or colours, contrast or brightness variations. This means that any visual change that happens inside of the shot itself, in this case on the compression level and the perceptual quality, is likely to be noticed more. Therefore, the shot-change is the perfect moment in which to modify encoding settings and quantization parameters.

Since the visual content at the shot level is relatively uniform and consistent, each shot can have a unique coding mode, with almost the same visual quality for all frames. Under these conditions, it is worth considering the use of specific encoding options with a low level of flexibility but strongly consistent, like Fixed QP or Constant Rate Factor (CRF). The former ensures constant bitrate, the latter instead constant quality. CRF requires a desired quality level to be set before encoding the video stream, so that the encoder strives to maintain that level by decreasing the bitrate during high motion scenes and increasing it when the action slows down. This is because our visual system perception is less sensitive to loss of detail in a moving scene compared to a static one. When encoding with CRF the final bitrate of the encoded file cannot be predetermined and it may increase uncontrollably for very complex scenes, producing variable levels of perceptual quality along the scene and within single shots (Dror, 2017). Nevertheless, in shot-based encoding this can be seen as a strength, since content complexity is generally uniform within the same shot. Moreover, the introduction of new perceptual video quality metrics like SSIM and VMAF over the traditional PSNR, provides an opportunity to enhance encoder decisions, in particular when the quality parameters have a great impact on the overall sequence, such as in per-title encoding.

Rate-Quality (RD) curves illustrate the trade-off between a stronger compression, hence reduction of bitrate, and a loss in perceptual quality. The choice of the appropriate quality metric is essential in determining the set of operating points reachable by the encoder, and since quality is a subjective property the quality of

the output can significantly vary depending on the metric used to evaluate it, even for the same video source. The cloud of all RD points combined from all RD curves comprises all the operating points reachable by the encoder. The convex hull of this set of points (*Figure 2.1*) is a fundamental tool in video encoding because it represents the upper limit of achievable compression performance for the given video sequence. Thus, the closer points lie to the convex hull, the more the encoder is able to be efficient in terms of compression. Next generation video encoders could have the opportunity to exploit and analyse RD data to optimise their video compression processes. Indeed, shot-based encoding, quality assessment, and RD curves are key concepts in dynamic optimization.

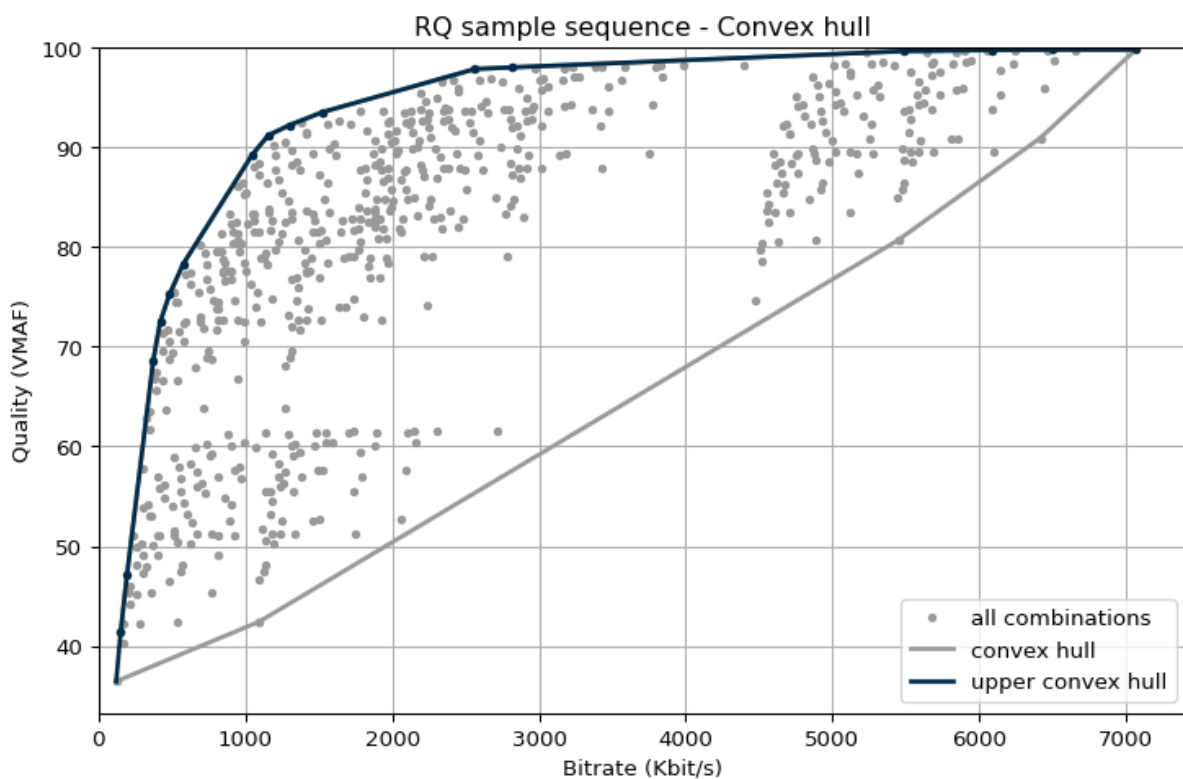


Figure 2.1, convex hull and upper convex hull of an arbitrary set of RD points.

In the presented study, FFmpeg is used for processing and encoding test video sequences, the VMAF library for objectively evaluating their visual quality, and the Python programming language for the arithmetic, vector, and matrix operations on the RD points and on their convex hull. FFmpeg is an open-source software suite intended mainly for audio and video encoding and decoding. It supports a wide range of coding options and filters, like the ones used in the testing phase in *Chapter 3*. VMAF is a perceptual video quality assessment algorithm developed by Netflix (see *Chapter 1.5*) and from 2020 it is part of the FFmpeg filters library. Lastly,

we chose Python because of its high level of abstraction as a programming language and because it guarantees a perfect synergy and integration with the system underneath, where for example FFmpeg commands are executed. In particular, within Python, the Numpy library turned out to be useful for faster manipulation of arrays. A detailed description on the experimental procedure is given in the following sections and in the next chapter.

Overview

According to Netflix (Katsavounidis and Guo, 2018), the Dynamic Optimizer framework follows the steps listed below. Before execution, the Optimizer is configured to work towards one or a list of output rate or quality targets. The output optimised videos need to comply with those targets.

1. First of all, the system is fed with a long video sequence, a RAW uncompressed video file to be split into single shots, also called coding units. Also high quality compressed videos can be inputted.
2. Multiple QP or CRF values are used to encode each shot. For instance, the H.264 encoder in FFmpeg allows for CRF values ranging from 0 to 51. The encoded shots are also referred to as elemental encodes. Their amount corresponds to the product of the number of coding units and the number of CRF values for each coding unit. It is therefore evident that the number of possible alternatives, or combinations, to be considered by the system are the number of CRF points at the power of the number of shots.
3. The chosen video quality metric is computed for each individual elemental encode, generating a set of RQ pairs. This collection can be stored in a multi-dimensional matrix where the first dimension is the coding unit indexes and the second one the number of CRF values. Instead, the third dimension is two, and it comprises a pair of values: rate and quality. For each shot, the two-dimensional array of RQ pairs can be plotted visualising the corresponding RQ curve.
4. Then, a joint convex hull is constructed for the entire sequence, which involves only a subset of points from each individual shot.
5. Finally, the convex hull is used to generate the optimal coding path that maximises quality for a specific target average bitrate or, equivalently, minimises bitrate for a given target average distortion.

The methodology described above were sourced from the work presented by Netflix in their reference paper (Katsavounidis and Guo, 2018). However, it should be noted that there are some differences with the present implementation, since some components were not included. First of all, shot detection does not impose any limit to the duration of each coding unit, unlike it was provided for in the reference paper, for example with a 10 seconds maximum time. The limit was suggested in the reference system in order to allow for frequent seek points and compression adaptation during streaming playback. In this case instead if a shot lasts more than 10 seconds it is not unconditionally splitted. Secondly, according to the reference paper, video sequences in input are resized to a ladder of lower resolutions in order to enable streaming client applications to fastly switch among them and to adapt to fluctuating network conditions. However, the downsampling of each shot to a number of target encoding resolutions is not part of the implemented pre-processing phase. Trimmed shots and downsampling are typical steps of most of the current on-demand streaming services. Since our work is more focused on optimization and optimization methods rather than adaptive streaming techniques we decided not to include them in the developed code, referring to an established literature and to already implemented algorithms for a more comprehensive treatment of the topic.

Based on the claims made so far, it can be inferred that the framework presented is well-suited in particular for on-demand streaming applications. Their content in fact can be exhaustively pre-processed before delivery, because there is no need to send it live as it happens. This kind of service permits providers to process the content before delivery, ensuring that the end-users receive the highest possible quality, but also it reduces their delivery costs by decreasing the bitrate and the amount of data to send as much as possible. The same happens in adaptive bitrate streaming, another similar context in which the Dynamic Optimizer capabilities can be exploited. Adaptive streaming is a common technique used in on-demand streaming in which the same content segments are encoded with different parameters, such as bitrate and resolution, to enable delivery based on users' network conditions. Optimization is becoming more and more crucial in adaptive streaming because it allows OTT providers to find the best combination of encoding parameters for each segment of the video. The same happens in shot-based encoding, in which the segments to encode into multiple versions correspond to the shots. Finally, the ideal content for this type of optimization would be a long sequence of non-homogeneous shots with alternating static and dynamic scenes. These scenes would have varying detail, motion, and complexity levels, and would be encoded with a wide range of compressions. A movie, a TV series episode, or a TV show reflects these characteristics.

The developed code is designed to algorithmically implement the concepts presented in the reference paper providing three alternative types of optimization. Despite their differences, all three share the main common steps, starting from shot detection and encoding, to quality assessment and the representation through RD curves. The following sections provide detailed descriptions of each of the common components, to then arrive at the specifications of each of the three optimization methods (*Chapters 2.1, 2.2, and 2.3*) and their related algorithms.

Shot detection

In video production a video sequence is typically structured into a hierarchy of two elements, scenes and shots. While the sequence is the largest narrative unit, the scene is the smallest unit of narrative within a sequence, and it usually consists of a single shot or a series of shots that are edited together to create a seamless whole. A shot is a single, continuous take of a scene, from the moment the camera starts rolling to when it stops recording. In video processing, it can be seen also as a sequence of video frames depicting the same visual contents. Shot detection algorithms are commonly used in video processing to automatically detect scene changes in a video stream by analysing differences in the visual content of consecutive video frames. This allows for efficient video indexing, retrieval, and editing.

On the operational side, shot change algorithms work can be challenging. We distinguish two types of changes, cuts and transitions (Gushchin et al., 2021). The passage from the last frame of a scene to the first frame of the next one is defined as a cut (*Figure 2.2*) when the change is abrupt and it involves the immediately followed first frame of the new scene frame. On the other hand, the transition is a progressive change which involves a certain number of frames, usually from 10 to 50. It can be a crossfade (*Figure 2.3*), the gradual dissolve of the new scene on top of the previous with overlapped and averaged pixel values in a weighted fashion, or a fade-in or fade-out, the gradual transition from or to black.



Figure 2.2, shot change: cut.



Figure 2.3, shot change: crossfade.

While existing algorithms have shown high accuracy in detecting scene transitions in general cases, they still struggle with identifying complex transitions. The built-in shot detection feature in FFmpeg, which was used in this work, is quite basic and imprecise. It solely considers the differences in pixel values between two consecutive frames, the current frame and the next one, and it is therefore incapable of accurately detecting gradual changes between shots. For instance, the *snow_mnt* resource is made up of three shots with dissolving transitions, but the algorithm considers the entire sequence as made of just one shot because it is not able to detect crossfades, rendering any shot-based optimization vain. Likewise, in the five-shot test sequence *rush_field_cuts* (Figure 2.2) the algorithm detects only four shots when the threshold is set to 0.25, which means a difference of at least 10% from the previous frame, but six shots when the threshold is set to 20%. In the first case, this leads to the algorithm missing one transition because the content at the boundary of the two scenes was too similar, while in the second one, to a false positive, the wrong detection of a change within the shot itself.

```
ffmpeg -i test.y4m
  -filter:v "select='gt(scene,0.2) '",
  metadata=print:file=test.log"
  -f null -
```

Listing 2.1, FFmpeg code snippet. Shot detection command.

Below is a description of the FFmpeg filters used in *Listing 2.1* to detect shot changes (FFmpeg filters, n.d.). The needed components are as follows:

- **scene()**. Assign a value between 0 and 1 to indicate a new scene. A low value stands for a low probability for the current frame to introduce a new scene, while a higher value means the frame is more likely to be the first of a new scene.

- **gt()**. Return 1 if the new-scene indicator is greater than the threshold, 0 otherwise (FFmpeg utilities, n.d.).
- **select()**. Extract the first frames of new shots to return to output. For each input frame, if the inner expression is evaluated to zero, the frame is discarded, otherwise it is sent to the output.
- **metadata()**. Manage frame metadata and print key-value pairs if metadata was found. When in print mode, the output is written to the specified output file.

The code above produces a log file with the same layout as to the one shown in *Listing 2.2*. It includes the new-shot indicator scores for each detected shot, which measure the difference between the two scenes, as well as the timestamp, both in time and frame index, of the first frame of new scenes. Results are then used to split the video sequence into individual shots and encode them separately.

```

frame:0    pts:197      pts_time:6.57323
lavfi.scene_score=0.242753
frame:1    pts:318      pts_time:10.6106
lavfi.scene_score=0.356787
frame:2    pts:378      pts_time:12.6126
lavfi.scene_score=0.368828
frame:3    pts:480      pts_time:16.016
lavfi.scene_score=0.463275

```

Listing 2.2, test.log snippet. Shot-detection results.

Shot encoding

Each shot gets encoded using a number of quantization parameters or CRF values, according to the specifications of the chosen encoder and the user-defined inputs. For example, the H.264 FFmpeg encoder offers 52 possible CRF points within a range of 0-51. However, the user may opt to only encode within a specific range, such as [10,40], and at custom intervals like one point out of two, resulting in this latter case in 16 versions of the same shot, each one at different compression levels. These multiple versions of the coding units are called elemental encodes.

```
ffmpeg -ss 0.0
      -to 6.57323
      -i test.y4m
      -c:v libx264
      -crf 28
      -pix_fmt yuv420p
      -an
      test.mp4
```

Listing 2.3, FFmpeg code snippet. Video encoding using libx264 encoder for the AVC standard.

In FFmpeg, the encoding process (FFmpeg documentation, n.d.) of a single shot is depicted in *Listing 2.3*, from which the following parameters emerge.

- **ss and to.** They refer to the starting and ending points of the video sequence and they are expressed in terms of their respective timestamps in time.
- **i.** This denotes the input file to be encoded. Instead, *test.mp4* refers to the output file that will be created. They may also include the file path if necessary.
- **c:v.** It specifies what encoder to be used for the input video stream. In this case, *libx264* is the default AVC encoder in FFmpeg.
- **crf.** The CRF used in the encoding process determines the output quality.
- **pix_fmt.** The output chroma subsampling scheme. It is useful when implicitly using the default presets (see *Table 2.1a* and the corresponding paragraph), to ensure that no other subsampling scheme is copied from the input file, like the 4:2:2 in a HDR video.
- **an.** It tells the encoder to avoid considering audio during the encoding process. As a result, the output file will not have any audio track.

RAW YUV files inside of a *.yuv* container have no header because pixel data are simply arranged in a contiguous sequence with no additional information. When inputting these kinds of files, the parameters in *Listing 2.4* have to be added before the input file keyword to specify their format, resolution, framerate, and subsampling scheme. All parameters may slightly vary based on the selected encoder.

```
-f rawvideo
-video_size 1920x1080
-r 25
-pixel_format yuv420p
```

Listing 2.4, FFmpeg code snippet: additional parameters when encoding from YUV input files.

The Constant Rate Factor has been chosen as the core parameter to optimise. It encompasses a set of compression variables that determine a constant level of quality for the output source. It is a little more sophisticated than the basic Quantization Parameter because it compresses different frames by different amounts, varying the QP to maintain a certain level of perceived quality, especially taking motion into account. For example, a CRF value of 18 will increase the QP in case of high motion frames and decrease it for low motion frames, essentially changing the bitrate allocation over time. Lower CRF values will result in higher quality, while higher values will increase the degree of compression. It is recommended to avoid using a CRF of 0 because it is lossless. Moreover it is generally better to stay within a range of values such as [10,40] or [15,35] and to avoid extreme points. Recommended CRF values for each encoder are listed in *Table 2.1b*. CRF, unlike QP, can compensate for the rough shot-detection provided by FFmpeg thanks to a frame-by-frame compression that guarantees an efficient content-dependent encoding at the frame level.

During video coding standardisation procedures, researchers have to use the only-available software implementation of the coding scheme under testing, which is called reference encoder. The JM, HM, Libvpx, and Libaom reference encoders are different from the FFmpeg native ones respectively for AVC (“H.264 Video Encoding Guide”, n.d.), HEVC (“H.265/HEVC Video Encoding Guide”, n.d.), VP9 (“H.265/HEVC Video Encoding Guide”, n.d.), and AV1 (“AV1 Video Encoding Guide”, n.d.) coding standards. In parallel to these reference encoders in fact, the open-source community, but also private companies, have been developing alternative software solutions in compliance with standardised specifications. These encoders are then used by most companies that deal with video processing. The default preset and profile configurations for the selected encoders presented in *Table 2.1a*. It is worthwhile to underline that a preset is a set of predefined encoding parameters for specific use cases, while a profile defines a set of constraints on the encoding parameters to ensure compatibility with specific playback devices or systems. The *medium* preset for example, tries to keep the balance between coding speed and computational complexity, while a faster preset minimises the encoding

time at the cost of higher bitrate and redundancy. On the other hand, common video profiles for instance in H.264 are *Baseline*, *Main*, and *High* profiles (Bing, 2015). The *Baseline* profile includes a limited set of tools and features, making it suitable for lower-end devices with limited processing power, while the *High* profile includes more advanced options to make it suitable for high-end devices and applications that require high-quality video.

Encoder	Standard	Parameters
libx264	H.264	-preset medium -profile high
libx265	H.265	-preset medium -profile main
libvpx-vp9	VP9	-profile 0 -b:v 0
libsvtav1	AV1	-preset 1 -profile main

Encoder	CRF range	Recomm. CRF range	Default CRF
libx264	[0, 51]	[17, 28]	23
libx265	[0, 51]	-	28
libvpx-vp9	[0, 63]	[15, 35]	31
libsvtav1	[0, 63]	-	50

Tables 2.1a and 2.1b. Above, default encoding parameters. Below, CRF range, recommended CRF range, and default CRF value.

Quality assessment

In order to evaluate the quality of each elemental encode, the VMAF quality metric is used. Instead, the number of bits per second, is gathered thanks to FFprobe, a tool related to FFmpeg that extracts and reads metadata information from multimedia streams. By combining these two values in a bitrate and quality pair, the set of RQ points can be determined for each shot. These RQ points form the shot's RQ curve. One can convert RQ values into their corresponding RD points, transforming quality into distortion like in *Equation 2.1*, the Inverted VMAF score. Practically, a single curve is described by a two-dimensional array of points that are associated with a specific shot that has been encoded at a particular CRF value. The rate-distortion behaviour of a coding unit indeed is described by the set of RD points from its elemental encodes, resulting in a distinct RD curve per shot.

$$IVMAF = 100 - VMAF$$

Equation 2.1, Inverted VMAF distortion metric.

The VMAF scores are calculated using the new VMAF built-in library in FFmpeg (“Using VMAF with FFmpeg”, n.d.), which includes the *libvmaf* filter. This filter provides a convenient and fast way to calculate VMAF scores within FFmpeg. To execute the filter, refer to the code in *Listing 2.5*, which requires three primary parameters.

- **i.** In this case, the VMAF filter envisages two input files, the uncompressed reference video and one of its compressed and distorted versions.
- **lavfi.** This command defines a complex filtergraph that allows for an arbitrary number of inputs or outputs. In this case, the two inputs are passed to the *libvmaf* library to produce VMAF scores. It also includes the possibility to compute the PSNR. The output is then printed to a log file in JSON format.
- **f.** The output file format is forced to null, causing FFmpeg to not print anything, because it is the VMAF library to be devoted to deal with the computation and saving of VMAF scores.

```
ffmpeg -i {ref_file}
       -i {dist_file}
       -lavfi "[0:v]setpts=PTS-STARTPTS[ref];
              [1:v]setpts=PTS-STARTPTS[dist];
              [dist][ref]libvmaf=feature=name=psnr:
              log_path=log.json:
              log_fmt=json"
       -f null -
```

Listing 2.5, FFmpeg code snippet: VMAF assessment.

For each elemental encode, the provided code produces a log file with a layout similar to the one shown in *Listing 2.6*. This log file contains frame-wise scores, including the VMAF metric, the PSNR, the VIF for information fidelity loss, and an *integer_motion* value for temporal information. The *pooled_metrics* key instead contains aggregated values of the VMAF components: their minimum and maximum value, and their arithmetic and harmonic mean for the whole clip.

```

{
  ...
  "frames": [
    {
      "frameNum": 0,
      "metrics": {
        "integer_adm2": 0.983573,
        "integer_adm_scale0": 0.969885,
        "integer_adm_scale1": 0.977173,
        "integer_adm_scale2": 0.986255,
        "integer_adm_scale3": 0.990574,
        "integer_motion2": 0.000000,
        "integer_motion": 0.000000,
        "integer_vif_scale0": 0.631788,
        "integer_vif_scale1": 0.962852,
        "integer_vif_scale2": 0.982089,
        "integer_vif_scale3": 0.989710,
        "psnr_y": 37.626702,
        "psnr_cb": 40.921762,
        "psnr_cr": 42.338333,
        "vmaf": 92.243645
      }
    },
    ...
  ],
  "pooled_metrics": {
    ...
    "vmaf": {
      "min": 68.920346,
      "max": 94.360660,
      "mean": 93.179319,
      "harmonic_mean": 92.962130
    }
  }
}

```

Listing 2.6, log.json snippet: VMAF results printed on a log file in JSON format.

Since PSNR is computed separately on the YCbCr components, its three values are merged together using *Equation 2.2* to obtain a single PSNR score for each frame. Quality assessment results are then combined on a shot-by-shot basis to get the total sequence ratings. At the shot level, both PSNR and VMAF scores are aggregated by taking the arithmetic mean of the per-frame values. These values are already calculated by the library and printed in the *pooled_metrics* section. At the sequence level instead, pooled results are again averaged, in this case using a weighted mean based on the shot duration, in order to obtain a single quality measure for the entire sequence.

$$PSNR = (6 * PSNR_Y + PSNR_{Cb} + PSNR_{Cr}) / 8$$

Equation 2.2, combined PSNR.

The aggregation of VMAF scores from individual shots produces different results from the assessment of the VMAF metric on the entire sequence. The former evaluation and aggregation is used during optimization steps to check for the compliance with quality targets, whilst the latter should be performed on the final optimised sequence in order to evaluate the overall optimization performances. Their ratings differ because the VMAF metric includes motion information, and per-frame scores also consider information from the previous shot. When evaluating isolated shots though, since they are separate files, it is not possible to refer to the last frame of the previous shot. As proven in *Figure 2.4*, the first frame of each shot has a lower VMAF value when evaluated alone because, and as shown in *Figure 2.5*, the *integer_motion* value of the first frames is much higher when calculated for the entire sequence. This leads to a deviation between the VMAF scores computed shot-by-shot and then aggregated, and the one resulting from the entire sequence assessment. Furthermore, this error increases proportionally with the number of shots, making the Dynamic Optimizer less effective on long sequences. To avoid this issue, we preferred to store and plot only aggregated VMAF scores, without computing the VMAF algorithm on the final muxed sequence, with the aim of producing comparison plots and average results with the same quality scale.

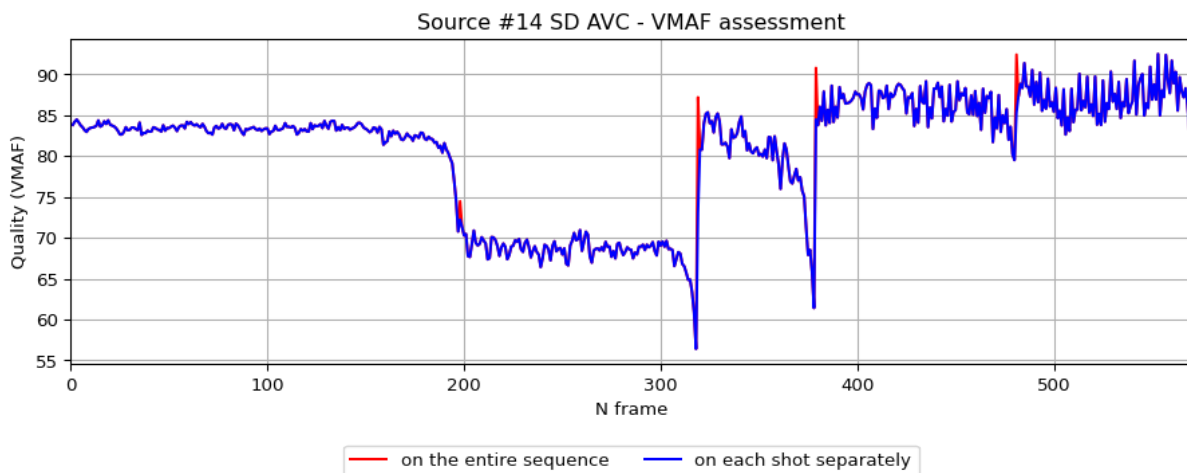


Figure 2.4, differences between per-frame VMAF scores.

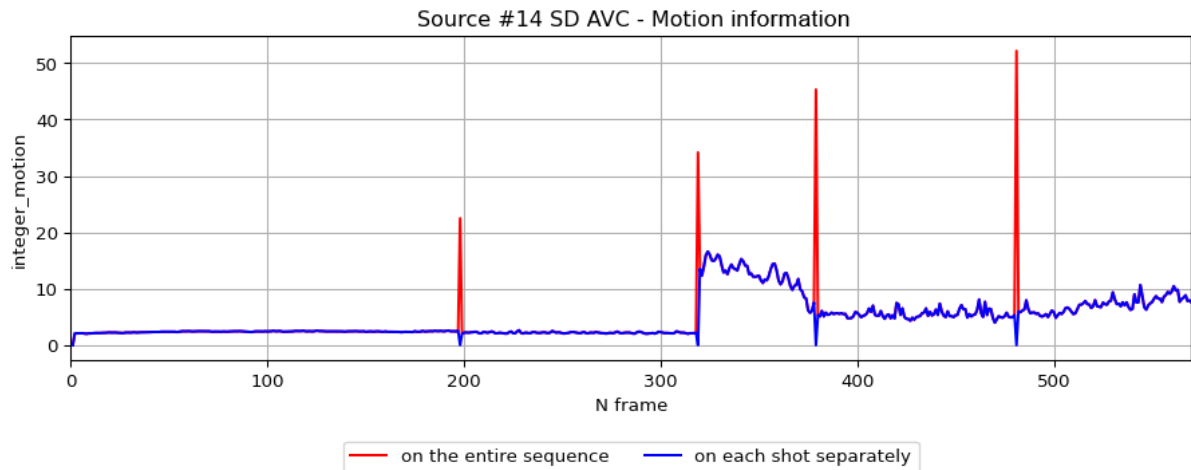


Figure 2.5, difference between per-frame *integer_motion* scores.

After considering all the individual compression ratios, bitrate values, and quality ratings for each shot in a sequence, we can assemble the set of possible compression combinations for the sequence and identify its convex hull. The points close to the convex hull contain the optimal combinations of elemental encodes that achieve the desired outcome either by minimising the distortion for a given target rate or by minimising the bitrate for a specific target quality. This goal can be achieved only by relying on the information gathered from the rate-distortion analysis.

Muxing

Lastly, video muxing or multiplexing is the process of sequentially combining multiple video inputs into a single video stream to be transmitted or saved as a container file. FFmpeg offers a specific command (*Listing 2.7*) that merges together the list of optimal elemental encodes coming from the optimization operations.

```
ffmpeg -f concat
-i shot_list.txt
-c copy text.mp4
```

Listing 2.7, FFmpeg code snippet. Concat command for video muxing where *-f* is the output format, *-i* the input files and *-c* the combined file in output.

2.1 BRUTE FORCE METHOD

The first technique introduced was also the earliest developed, as it is the most straightforward and intuitive. It has been labelled as a brute force method because it directly aims at the optimal solution without any approximations, but in a very expensive way in terms of time, number of computations, and encoding processes.

After having gathered all RD values from each elemental encode, the brute force method proceeds by generating every possible combination of compressed shots for the entire sequence. For instance, a 4-shot scene that is encoded at only 6 CRF points for each shot produces 24 elemental encodes and 1296 potential combinations to evaluate (*Equation 2.3*). Considering the huge amount of shots that typically compose a movie or an episode of a television serie and, in the AVC encoder for example, the availability of 51 CRF points, the number of combinations to assess becomes prohibitively high. To illustrate further, a sequence comprising 126 shots encoded at 31 CRFs would yield over 8×10^{187} possible alternatives.

$$C = P^U$$

Equation 2.3, the total number of combinations C , where U is the number of coding units and P is the number of CRF points per shot.

The responsibility of the creation of all combinations is entrusted to the *Itertools* library in Python. This module implements several functions that create efficient iterators for fast looping. One of them, *product()*, is depicted in *Listing 2.8* and it accepts a matrix of RD points for each elemental encode as input, flattened using the *** operator that packs the arguments into a single tuple, and returns all possible combinations, to be later processed in a for loop. The algorithm indeed proceeds to iterate through all available options to look for the optimal solution, the one that minimises distortions without exceeding the bitrate target, or minimises the rate without falling below the quality target (*Listing 2.9*).

```
for current_comb in itertools.product(*elem_encodes):  
    #do something
```

Listing 2.8, Python code snippet. For-loop of *elem_encodes*, the matrix of all RD points.

```

if current_x < target_value and current_y < y_min:
    x_min = current_x
    y_min = current_y
    output = list(current_comb)

```

Listing 2.9, Python code snippet, where x and y are the rate and distortion values or vice versa. If the current point is below the target and the current minimum, it becomes the new current optimum.

The optimization of the four identified shots in *rush_fields_cuts* at 10, 18, 25, 32, and 40 CRF values require 20 encodes and results in 625 options, ranging from the highest compressed sequence corresponding to the CRF combination [40, 40, 40, 40] to the one closest to the original version, where all shots have a CRF value of 10. Each combination is plotted with its corresponding bitrate and distortion pair (Figure 2.6). The optimal solution for a 1000 Kbps bitrate target is [25, 32, 25, 32], while for a 98.5 VMAF quality target, or its reciprocal 1.5 I-VMAF distortion target, it is [10, 25, 18, 18]. The shots belonging to the optimal combination are then multiplexed together in order to produce the final optimised sequence.

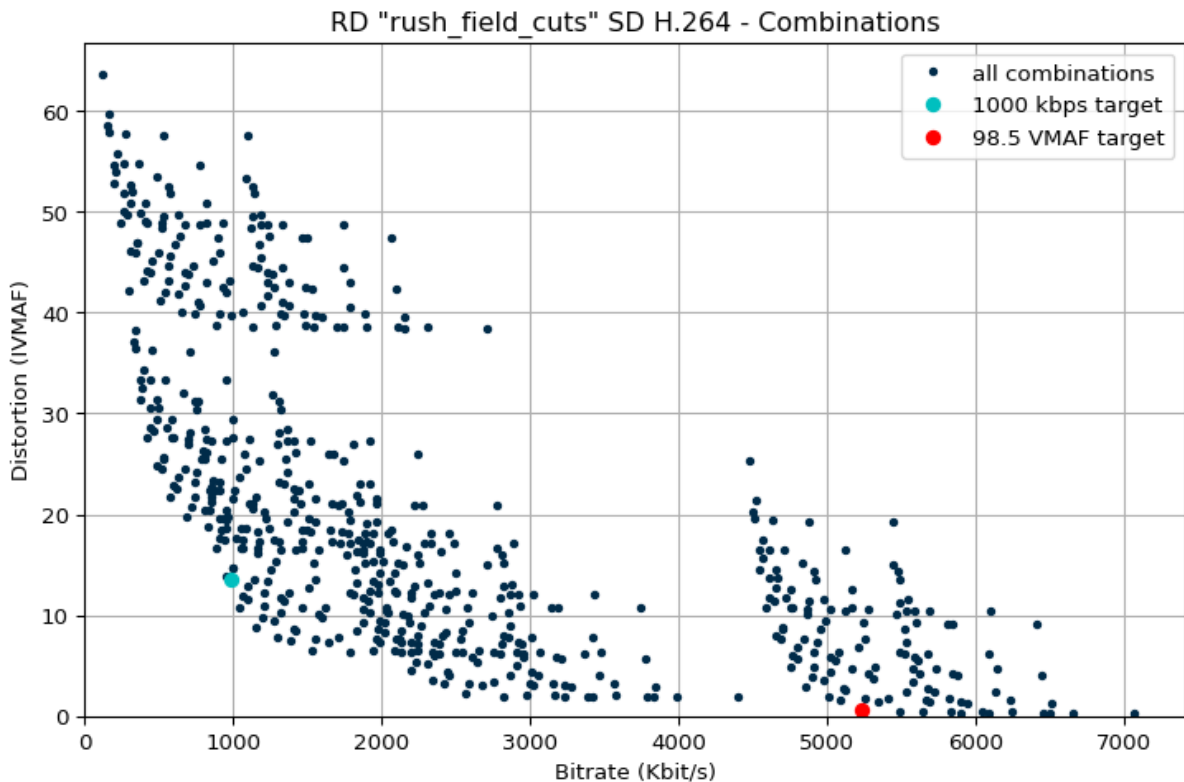


Figure 2.6, RD combinations for the *rush_field_cuts* source. The two highlighted points are the optimal combinations or solutions for the specified rate and quality targets.

The brute force approach suffers from exponential complexity, as the number of combinations to consider increases exponentially with the number of shots and CRF points (*Equation 2.3*), and it is therefore unsuitable for long sequences with many shots and number of elemental encodes. For this reason, this approach is primarily used for benchmarking purposes as it considers all existing solutions, not just those lying on the convex hull, and so it returns the best possible solution. For instance, in *Figure 2.6*, the brute force approach returned [25, 32, 25, 32] for the bitrate target. However, for the same source the Lagrangian method fully detailed in the next chapter produced the combination [32, 32, 25, 25], which is part of the convex hull but not the best solution. Due to its complexity the brute force approach is only suitable to evaluate the performance of the other two methods that however have a certain degree of approximation.

2.2 LAGRANGIAN METHOD

According to Lagrange theory, encoding parameters, in our case the CRF value to which encoding each shot, can be optimised independently of each other. This can be done shot-by-shot, without considering the sequence as a whole and the dependencies among shots, thus simplifying computations significantly. However, compared to the previous brute force approach, this method may return suboptimal solutions because it cannot reach points that do not lie on the convex hull. Nonetheless, studies have demonstrated that the Lagrangian solution is a reasonable approximation when the convex hull is dense enough (Ortega and Ramchandran, 1998), that is when enough elemental encodes get encoded, thus keeping the costs for encoding still high. In this section we present the practical implementation of this method within the Dynamic Optimizer framework.

Once the individual RD values for each elemental encode have been gathered and grouped by shot index, the leftmost and rightmost RD points for the entire sequence are extracted. This involves taking all shots encoded with the lowest CRF value, for example 15, and aggregating them to obtain the rightmost RD point for the sequence. Likewise, in order to identify the leftmost point, the most compressed shots are taken, the ones encoded at the maximum CRF value, for example 45. Assuming to place these points in the Cartesian plane, we can compute the slope between the two points, from now on the total or sequence slope (*Figure 2.7*). We can see this as the slope or gradient of a straight line passing through two points using *Equation 2.4* where it is referred to as ‘*m*’.

$$m = - \frac{(\max_{dist} - \min_{dist})}{(\min_{rate} - \max_{rate})}$$

Equation 2.4, the slope between two RD points.

In order to find the initial optimal elemental encode for each single shot, a straight line with the same slope as the total slope is slid toward the convex hull, until it intersects the first point (*Figure 2.8*). Since in this method, for the sake of complexity, there is no way to know the position of all combinations in advance, the same line can be slid toward the RD curve of each single shot, like in *Figure 2.9*. For this purpose, *Listing 2.10* illustrates the Python code involved in the calculation of the slope between consecutive RD points in the curve, which finds the couple of points whose slope minimises the difference with the total slope.

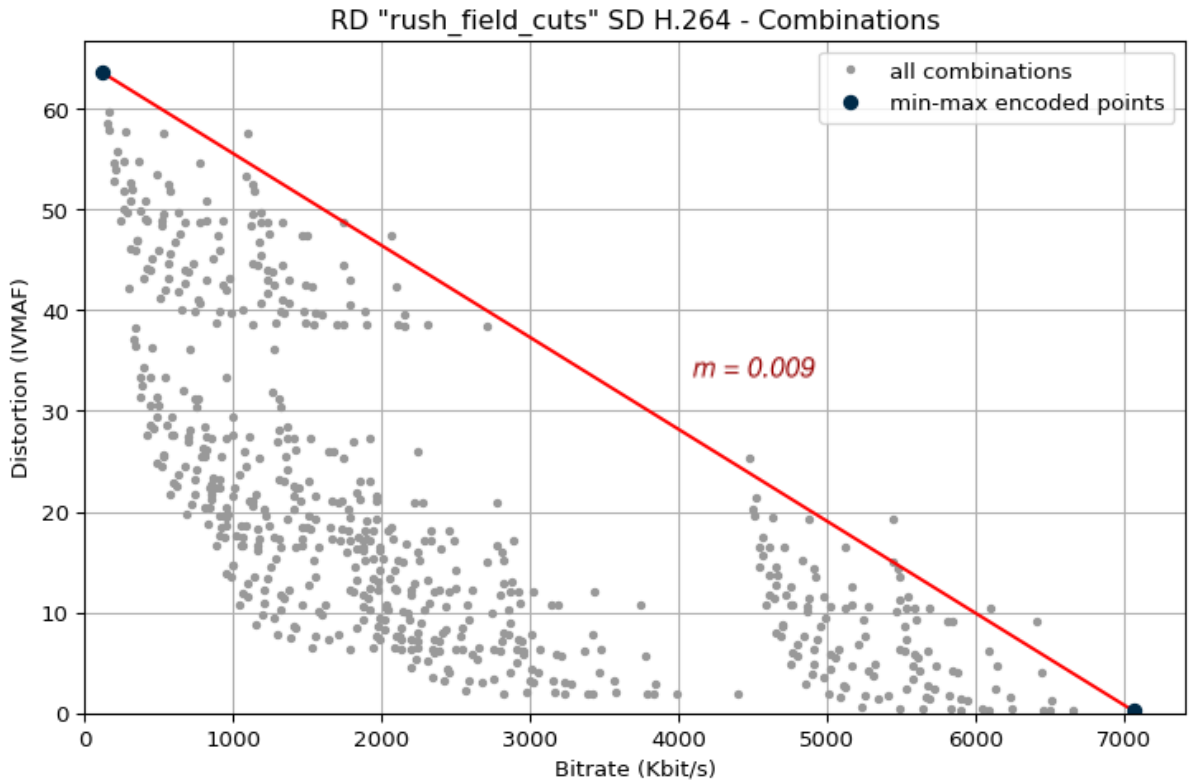


Figure 2.7, the line between min and max sequence RD points.

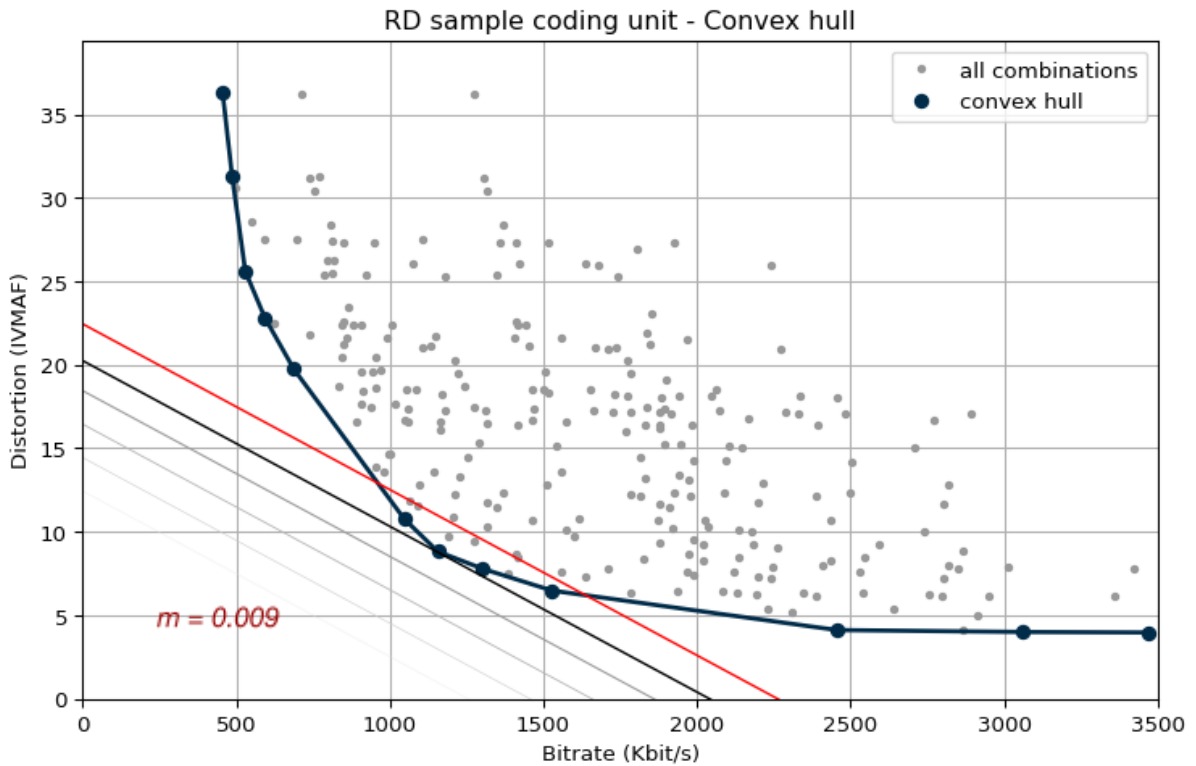


Figure 2.8, a straight line with the same slope as the total one intersecting the first point encountered on the convex hull.

```

diffs = abs(current_slopes - sequence_slope)
min_slopes = np.argmin(diffs, axis=1)

```

Listing 2.10, Python code snippet. Use the *argmin()* method to find the minima, in the first axis, of the differences between the slopes of two consecutive points in an RD curve and the *sequence_slope*.

An additional check is required in order to find the optimal point, one of the two from *Listing 2.10*, the resulting slope. The check is performed by drawing a line with the same slope as the total slope, starting and ending at the intersection with the x and y axis, and passing through one of the two involved points. The line's direction is lower-right to upper-left corners in the plane, and it is used to check if any of the other points of the RD curve lie on its left side. Whenever the cross product between the line and the other points turns out to be greater than zero (*Listing 2.11*), it means that there is a point on the left side of the line, and the one we considered is not the first encountered. The check is repeated until none of the remaining points are on the left, as illustrated in *Figure 2.9*, so that the current point is for sure the one with the closest tangent slope to *sequence_slope*.



Figure 2.9, the RD discrete curve of the first shot of *rush_field_cuts*, with the five encoded points and the tangent to the curve whose slope is closest to *sequence_slope*.

```

cross_product = (LX[0] - LY[0]) * (P[1] - LY[1])
                - (P[0] - LX[0]) * (LY[1] - LY[1])
if cross_product > 0.1:
    current_pint = new_point

```

Listing 2.11, Python code snippet. P() are the coordinates of the points, while LX() and LY() are the (x,y) coordinates of the points intersecting respectively the x and y axis.

The code demonstrated so far returns a temporary optimal compression level for each coding unit. Their RD values are then combined using a weighted mean on shot duration to create the rate-distortion pair of the entire sequence. The RD score of the current optimal combination is then checked to determine whether it exceeds or not the target set at the beginning, as shown in *Listing 2.12*. If it does, the leftmost RD sequence values are updated, otherwise the rightmost ones are updated, producing a new *sequence_slope* value. It is worth mentioning that the Numpy *einsum()* method in *Listing 2.12* provides an efficient and concise way to do element-wise multiplications and sums inside of multi-dimensional arrays.

```

new_rate = np.einsum('i->', current_opt["rate"])
new_dist = np.einsum('i->', current_opt["dist"])
if np.einsum('i->', current_opt["rate"]) > target_value:
    current_opt["right"] = [new_rate, new_dist]
else:
    current_opt["left"] = [new_rate, new_dist]

```

Listing 2.12, Python code snippet. Current optimal combination comparison with the given target.

The process is iterated, and for each iteration, a new total slope and optimal CRF are computed, to progressively refine the results. It stops when the algorithm reaches convergence, for instance when the new combination is the same as the last stored one. Since the code always updates two values, the leftmost and rightmost RD points, it takes as final output the one on the left, because it meets the target requirement and has a lower bitrate. *Figure 2.10*, depicts the update of the total slope at each new cycle, along with the temporary left and right RD points. We can see how it only reaches points lying on the convex hull and for a sequence of 4 shots encoded at 5 CRF values each, it achieves convergence after five iterations.

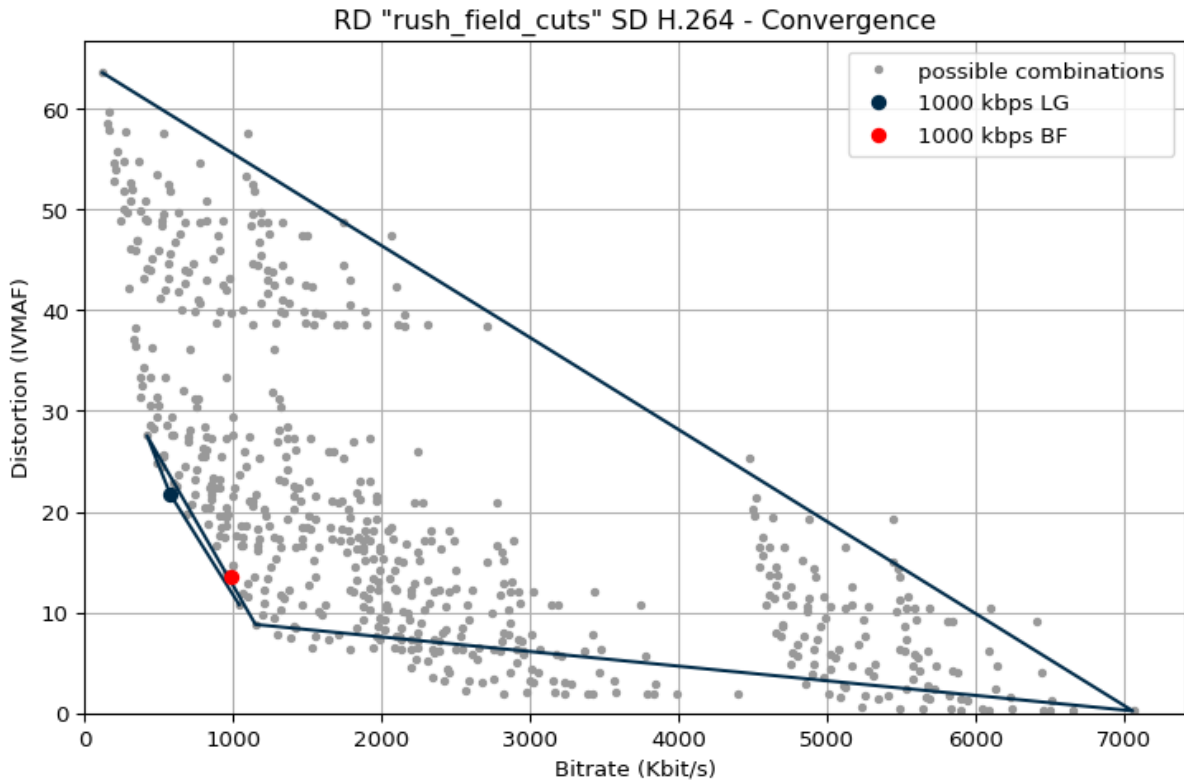


Figure 2.10, RD combinations for the *rush_field_cuts* source. The segments connect the different leftmost and rightmost points at each new iteration.

The main advantage of the optimization based on Lagrange theory, especially compared to the brute force approach in *Chapter 2.1*, is its low computational complexity. It does not require a significant amount of computational resources to achieve the final optimum, because it avoids searching for the optimum on all possible combinations. This method is therefore suitable when time and computational costs are not relevant factors and can be used in scenarios where high-quality results instead are important, because even though it does not guarantee optimality, its results are very close to the absolute optimal. However, it employs the same amount of encodings of the brute force method, as all elemental encodes for each coding unit have to pass through the encoder. To improve this aspect, in the next section we introduce a new approach based on Lagrange theory but with the addition of curve fitting.

2.3 CURVE FITTING METHOD

The optimization method explained below resembles the Lagrangian theory with exhaustive coding explained in *Chapter 2.2*, except for the fact that it allows for the encoding of each shot using a limited number of CRF values. This means that instead of encoding all of the points in the [10,45] range the optimal solution can be achieved for example using just 5 or 6 points within that range. By doing so, the amount of time and computing resources required to encode shots can be drastically reduced while also minimising the number of algorithmic operations needed, because it is based on Lagrangian optimization. The main drawback of this approach is the degree of approximation and suboptimality of the final solution, which is measured in *Chapter 3* and makes it the least precise method.

When using curve fitting, the choice of the CRF values to use when encoding shots depends on the selected interval. For wider ranges like [5,45] it is better to use a higher number of points, whereas for narrower ranges, it is recommended the opposite. Moreover, it is generally advised to encode from 4 to 7 CRF values because the use of fewer points may result in the algorithm being unable to accomplish the curve fitting, while using too many points would unnecessarily increase the encoding time and costs without providing any performance benefits. The initial CRF range is divided into the number of points per shot using the method outlined in *Listing 2.13*. For instance, if the range is [10, 40] and the encoding is to be done using 5 points, the resulting CRF values would be [10, 13, 25, 38, 40].

```
def interval(bounds,n):
    w = (bounds[1] - bounds[0]) / (n - 1)
    return [round(bounds[0] + i * w) for i in range(n)]
```

Listing 2.13, Python code snippet where *bounds* is the interval edge, i.e. [15,45], and *n* the number of subintervals to split.

The starting basis for curve fitting operations and their approximation of the RD interpolating curve for each shot, are the RD values of the few encoded elemental encodes. Rather than encoding all points of the curve, only a small number of them are processed. The others instead are estimated during the curve fitting process. Curve fitting is the construction of the curve or any mathematical function that best fits a series of data points. *Equation 2.5* is the function of an equilateral hyperbola,

and it has been identified as the best model function able to describe the trend of an RD curve. The approximation of this function is performed using the `curve_fit()` method provided by the SciPy library in Python, as described in *Listing 2.14*, where it is translated in its implicit form. By adjusting the a , b , c parameters, the hyperbola function is custom shaped by the fitting algorithm to better suit the set of points and to provide the best possible description of the RD curve behaviour. The construction of the curve needs to be constrained by lower and upper bounds on the parameters to avoid negative values, as rate and distortion values never go below zero. The method returns the value of the a , b , c parameters of the equation that describes the curve, as well as the estimated covariance of the parameters.

$$y = \frac{1}{x}$$

Equation 2.5, RD curve fitting explicit model function.

```
def f(x, a, b, c):
    return a / (x + b) + c

par, cov = curve_fit(f, x, y, bounds=((0, -np.inf, 0), np.inf))
```

Listing 2.14, Python code snippet where f is the implicit model function, a, b, c its parameters, x and y the encoded rate and distortion set of points.

In *Figure 2.11*, the actual RD curve formed by the real encoded points is compared to its estimated version obtained through curve fitting using only five CRF points as a reference. The close similarity between the estimated and encoded curves proves the effectiveness of this approach. However, the result shown is not solely made of the outcome from the curve fitting process mentioned above, but it comes also from additional adjustments explained hereinafter, which aims at a more accurate fit. Specifically, it is not just a matter of finding the curve equation, instead there are two additional issues that must be considered. The first one arises when bounding the function to prevent negative values, which leads to less accurate fitting especially at low bitrates. There, the function tends to shift upward, moving further away from the encoded points, so it has to be shifted back. The second concern lies in the fact that the resulting equation describes the curve as a continuous function, even though it should consist of a limited and discrete number of RD points, the available CRF values in the selected encoder.

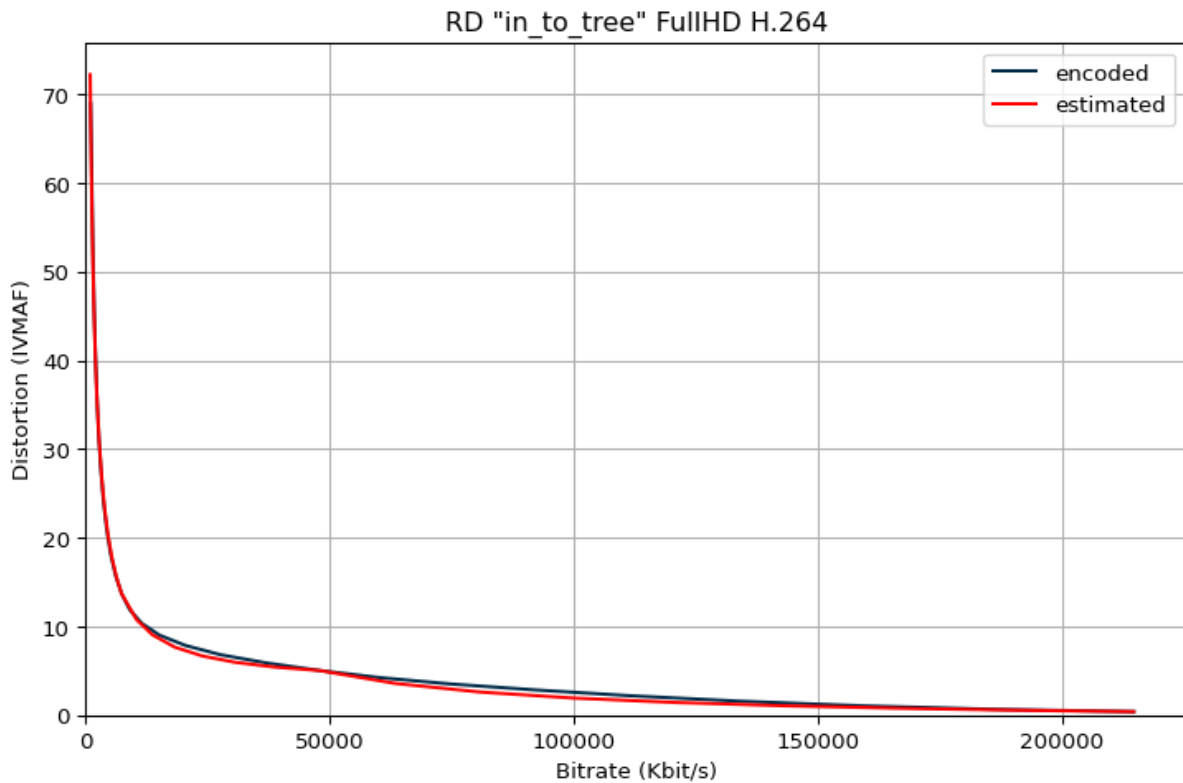


Figure 2.11, curve fitting performances in the CRF range [10,45].

To address the second problem, the algorithm must estimate the position of all the missing points within a certain range, for example the 31 points in the [10,40] interval, starting from the few encoded ones. However, a uniform distribution on the bitrate or distortion dimensions would result in an incorrect placement of points, as shown in *Figure 2.12*. In this case, it results in sparse CRF points at lower bitrates where the curve rapidly descends and in a dense concentration on them in the final part of the curve. To prevent this, the proper spacing and arrangement of the points along the curve takes advantage of its probability distribution function. The method in *Listing 2.15* solved the problem as illustrated in *Figure 2.13*.

```
def pts_disp():
    tx = np.linspace(x[0], x[-1], num=51)
    ty = f(tx,*par)
    cdf = normalise(cumtrapz(ty, tx, initial=0))
    f_interp = interp1d(cdf, tx)
    return f_interp(np.linspace(0, 1, 51))
```

Listing 2.15, Python code snippet. Custom points distribution along the curve.

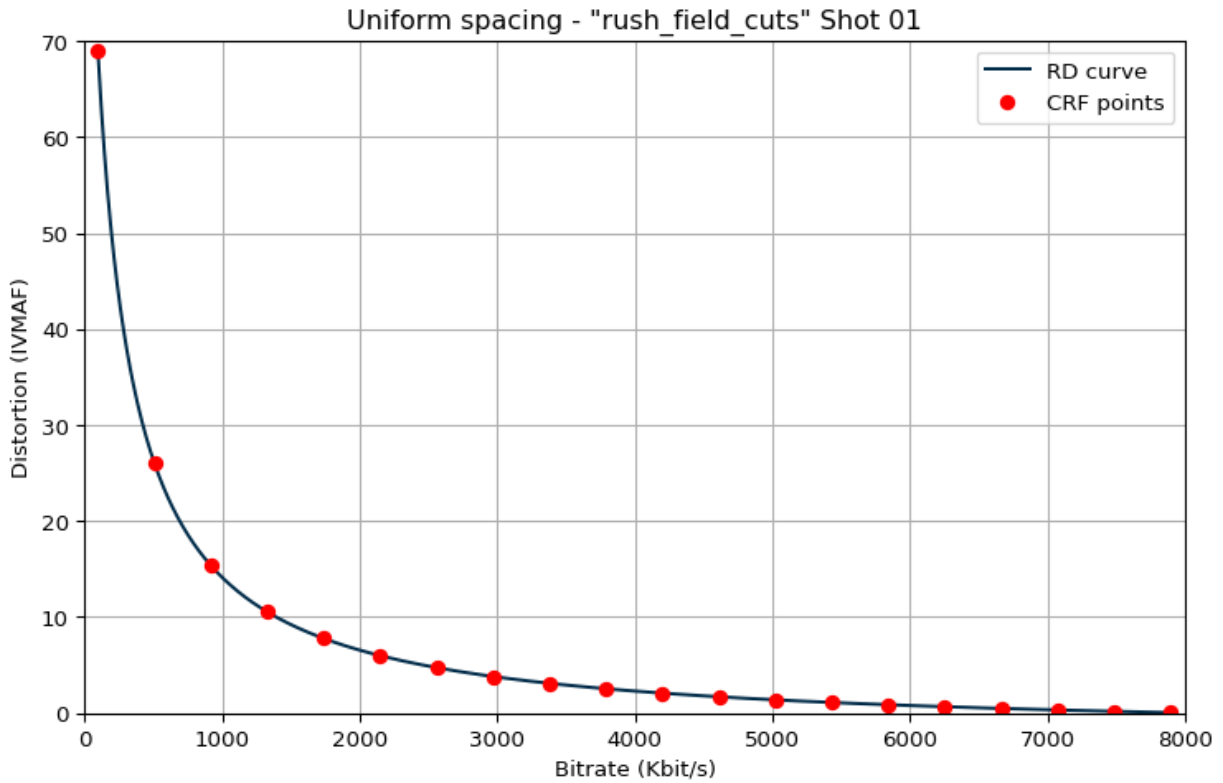


Figure 2.12, Evenly spaced CRF points on the x-axis and along the estimated RD curve.

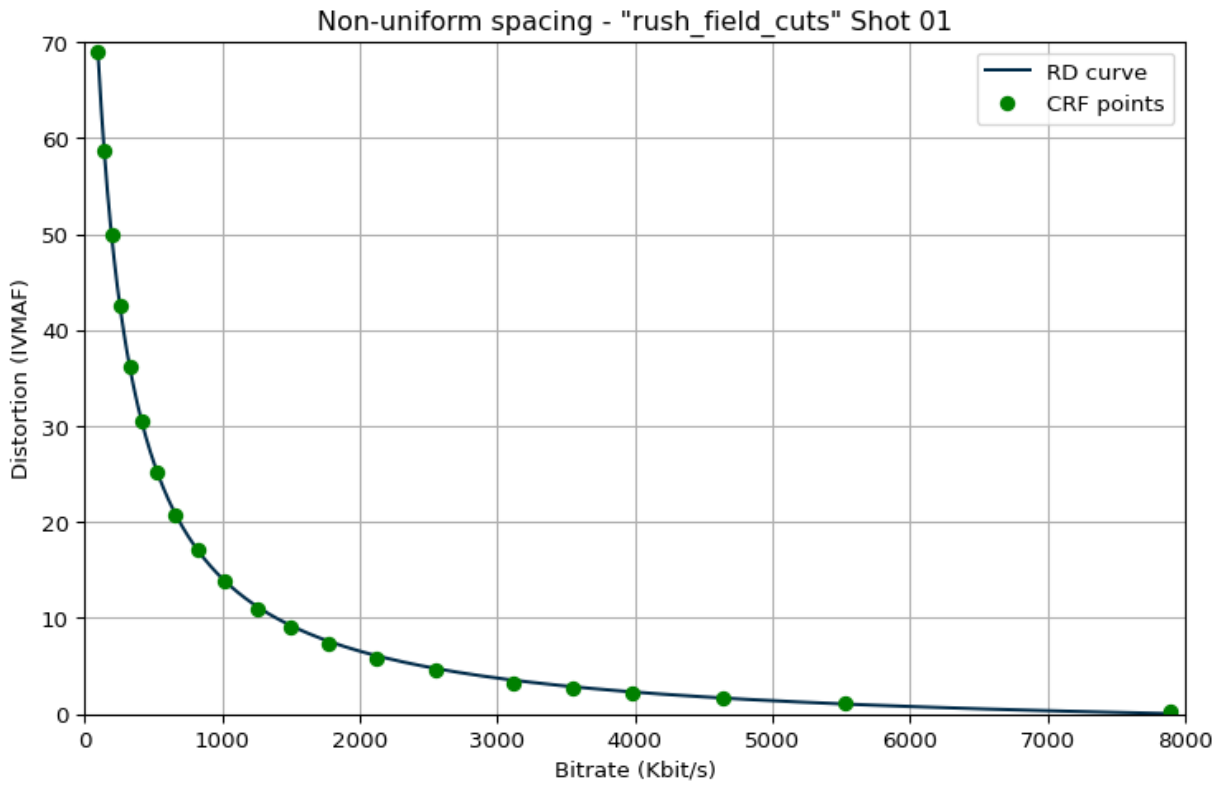


Figure 2.13, CRF points spaced according to the distribution function of the estimated RD curve.

The code outlined in *Listing 2.15* generates a set of 51 linearly-spaced bitrates from the minimum to the maximum encoded rate values, as well as the corresponding distortion values, obtained thanks to the equation $f()$ derived from the curve fitting process (*Equation 2.5*). This step allows for the creation of the continuous RD equation. The Cumulative Distribution Function (CDF) of the RD curve, defined as the integral of its probability density function (Tanyer, 2012), is computed by approximating the integral of the distortion values along the rate coordinate using the trapezoidal rule. The resulting CDF probabilities for the rate values (*Figure 2.14*) are then normalised in the range $[0,1]$. The algorithm finally creates an interpolation function based on the CDF values and applies it to the linearly-spaced array of 51 CRF points in order to properly arrange rate values on the x-axis.

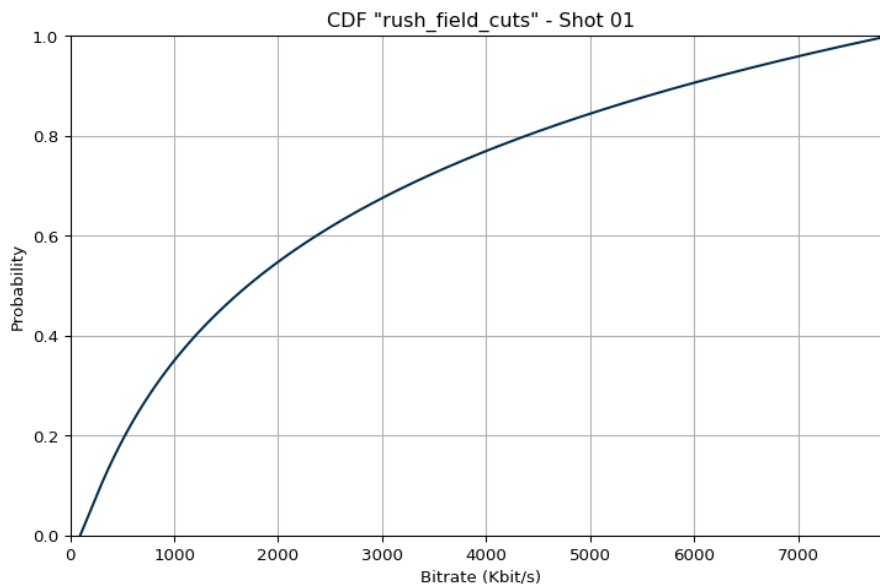


Figure 2.14, rate CDF of the first shot of *rush_field_cuts*.

In the end, with the aim of addressing the upward shift problem and improving the fitting to real values, the resulting RD points are again adjusted, in this case along the vertical axis. The sample code in *Listing 2.16* shows how interpolation with the few encoded points as a reference is used to move up and down points. For each non-encoded range, it linearly interpolates the estimated points either compressing or expanding them using the two encoded extremes as a reference. The interpolation is performed by the Numpy *interp()* function, suitable for monotonically increasing samples like the points along an RD curve.

```

distortion = np.concatenate([np.interp(y[v:v+1],
                                     (y[v],y[v+1]),
                                     (y0[i],y0[i+1]))
                             for i,v in enumerate(crfs)])

```

Listing 2.16, Python code snippet, where y are the estimated distortion values, y_0 the actual encoded ones, and $crfs$ the set of CRF values used in the encoding process.

The results can be observed in *Figure 2.15*, which includes a test sequence with only one shot, but also in *Figure 2.16* and *Figure 2.17*, with the first two shots of the *rush_fields_cuts* resource. Despite the distinct behaviour and trend of the different curves, the improved curve fitting algorithm based on CDF and interpolation with actual values, offers a significant improvement over the curve fitting algorithm alone.

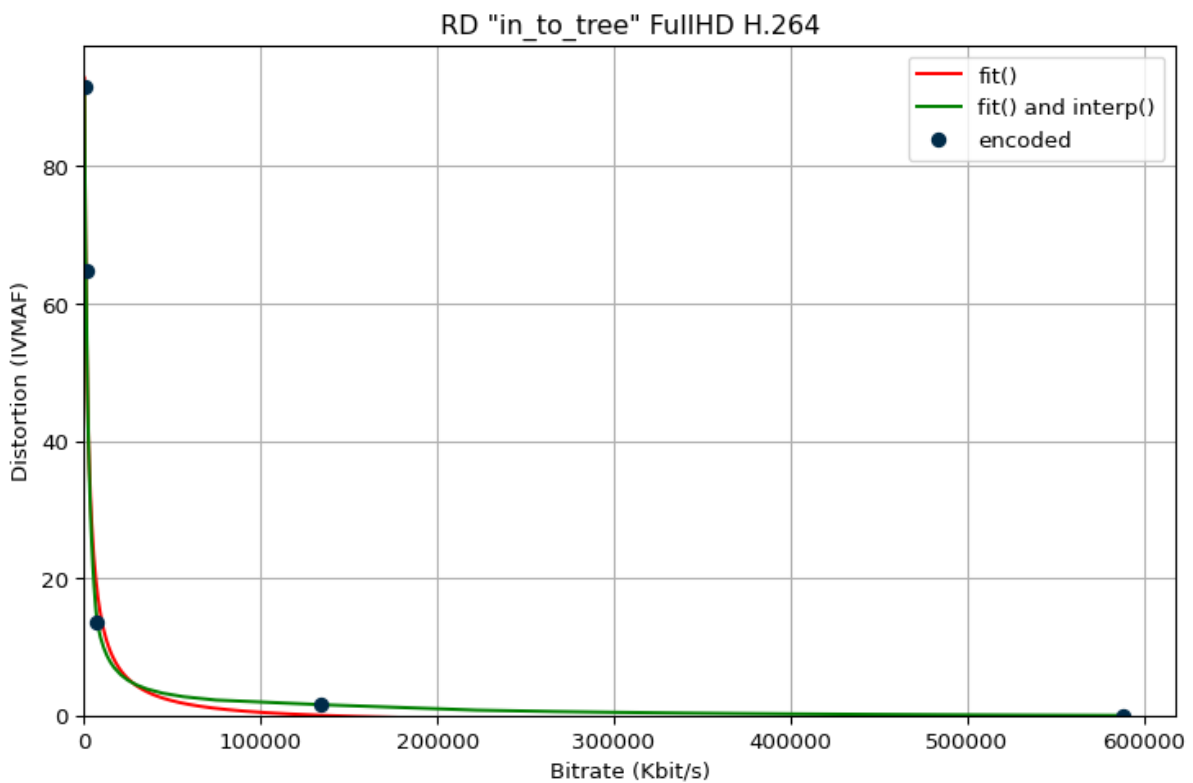


Figure 2.15, curve fitting performances for the RD curve of a single-shot test video.

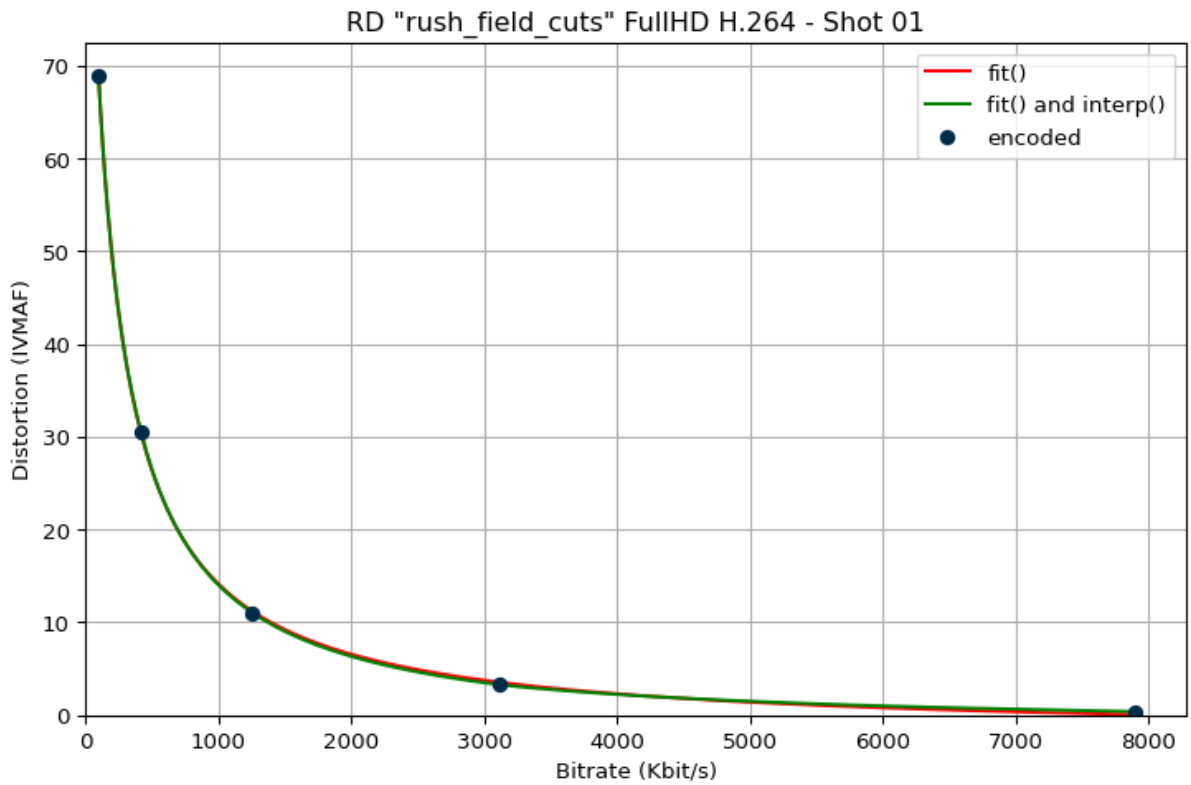


Figure 2.16, curve fitting performances for the first shot of the test sequence.

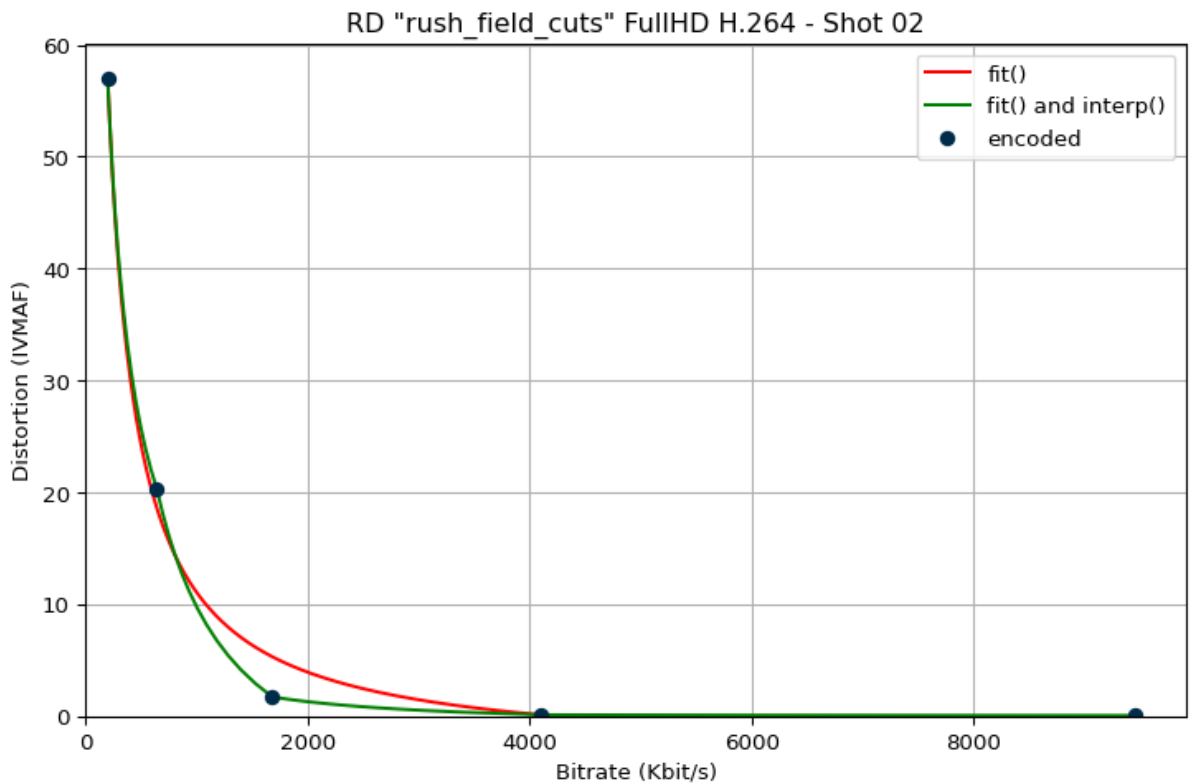


Figure 2.17, curve fitting performances for the second shot of the test sequence.

The presented curve fitting technique focuses only on estimating the RD curves behaviour and the missing RD values for the non-encoded shots. By integrating it into Lagrange theory, the previous and following steps become identical to those outlined in *Chapter 2.2* for Lagrangian optimization. Even though this technique only provides solutions on the convex hull, unlike the pure Lagrangian it encompasses all encoded points, although estimated. Thus, when comparing the two techniques on a small number of CRF points per coding unit, for example 5 or 6, Lagrangian optimization works only on the few encoded points while Curve fitting optimization on the entire range, performing much better. Indeed, the latter has low computational complexity and coding costs, as only some of the elemental encodes are encoded, and the rest are estimated. However, since it is based on an approximation, the estimated values may differ from the actual ones, and the results may not be as precise as those obtained from the other techniques. A detailed performance comparison of the three methods is presented in the next chapter.

3. ASSESSMENT AND RESULTS

In order to evaluate the performances of the dynamic optimization framework we have conducted a comprehensive set of objective assessment tests. These tests consist of a comparison between the optimised versions from the presented software and the same video sequences encoded without any optimization technique, so the one returned just by the encoding processes. In addition, we provide a comparative assessment of the effectiveness of three implemented methods by feeding them with different types of contents and by fine-tuning the available encoding parameters. We present in this section the implementation process of such tests, from the selection of relevant video test sequences to final results. In particular, according to the consolidated research methods in current digital video studies (Mansri, 2020), when analysing the characteristics of any video source, we focus on its rate and quality performances.

The roots of today's video compression schemes are based on the classical Rate-Distortion (RD) theory, originally developed by Shannon in 1959 (Blau and Michaeli, 2019). The theory explains and describes the fundamental tradeoff between the bitrate used to encode a video signal, and the distortion introduced when decoding the information from its compressed representation (Cover & Thomas, 2012). This tradeoff can be represented by a function, the RD curve illustrated in *Figure 3.1b*, in which the bitrate increase corresponds to a reduction of the distortion. One can easily convert it into a RQ curve, where Q stands for quality, by reversing the distortion values (*Figure 3.1a*).

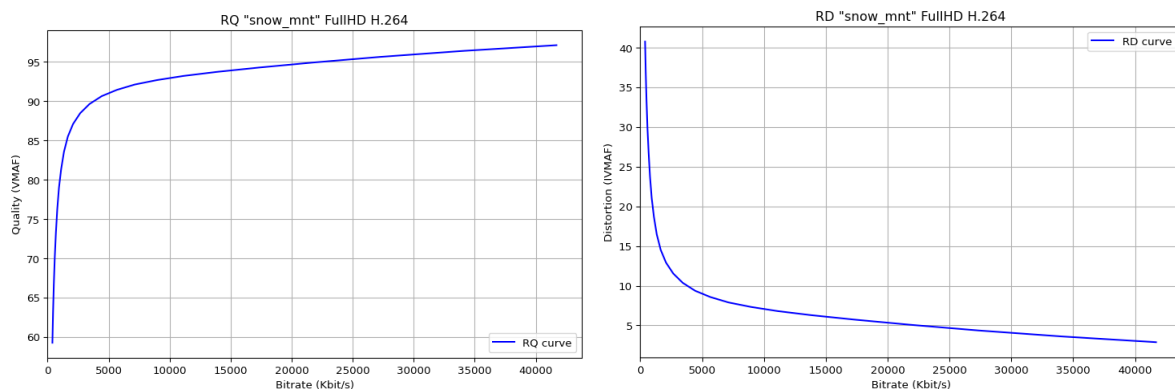


Figure 3.1a and 3.1b. On the left, a RQ curve. On the right, its inverse, the RD curve

The widely varying content and motion one can find in typical video sequences requires a complex and continuous interplay among various motion analysis and coding options. This is where rate-distortion capabilities come into place. Once an arbitrary output quality is set, we have to attempt to use the fewest number of bits possible to represent the source. Alternatively, one can set a limit to the output bitrate and attempt to reach the highest possible visual quality from the same source. This is the typical tradeoff of RD theory on how much details and valuable information we are willing to cut out in order to reduce the occupied storage space or the number of bits to transmit.

In this chapter, we will present the rate-distortion analysis of the Dynamic Optimizer in different conditions, starting from the inputted test video sequences and the choice of the initialization parameters. Within the selected encoding scheme indeed, which is fixed and standardised, there are some encoding parameters that we can control. They are chosen on an input-by-input basis and they are the core of the optimization process because they are those to be optimised.

3.1 OBJECTIVE EVALUATION

Methodology

The assessment process described in this chapter is closely related to the optimization steps already detailed in *Chapter 2*. In this introductory paragraph therefore, some of its broad outlines may be recalled. The system needs to be set up for analysis purposes before execution, with a focus on custom parameters, variables, and inputs. The input-by-input components configured at the beginning are outlined below and explained in depth immediately after.

1. The objective function to minimise, typically distortion, and one or more constraints to meet, often a bitrate limit.
2. The coding units precision, that determines the shot detection threshold, and the elemental encodes' compression levels, controlled by the CRF parameter.
3. The distortion metric used for evaluation, which affects the RD features interpretation and the later performance analysis.
4. Test video sequences. The Dynamic Optimizer can receive a variety of different inputs.

The first item of the list is the target constraint. Ideal input sources for the presented framework should be high quality video sequences files, both raw and compressed, with an initial quality or bitrate greater than the target limit fixed beforehand. They can be used both for the encoding processes but also as a reference for the quality evaluation and the comparison with the generated optimised files. In order to have a comprehensive analysis, the wider spectrum of bitrate and quality target values should be considered, so that also the behaviour at their extremes is taken into account. On one hand, it is necessary to consider these far ends because they may generate unexpected results and be not consistent with the general trend. Most of the time however, for the sake of clarity and in order to identify consistent average trends, we will mainly focus on common ranges. Target values must be selected within a certain range, not only according to the source properties but also to meet storage or transmission channel requirements. For each input sequence indeed, based on features like its frame rate, resolution and other spatiotemporal information, we have to define a reasonable set of quality and rate output targets, but also a number of encodings and degrees of compressions.

The second entry of the list regards shot detection threshold and CRF. In *Table 3.1* and *Table 3.2* we can see the bitrate and VMAF values of two different clips made of a single shot, encoded at 8 CRF samples in the whole range of the H.264 encoding CRF values. We assume for example, with the aim of comparing the two videos, to consider bitrate ranging from 1000 Kbps to 500 Mbps and quality values ranging from 60 to 90 VMAF score. To avoid encoding at all possible CRF values, which would result in 52 encodes, some of which may be unnecessary, we opt to reduce this range. However, we would need to select different CRF ranges for the two sources in order to meet the same requirements. Considering bitrate, we would look at the [0,36] range for the first clip (*Table 3.1*), but [7,44] for the second one (*Table 3.2*), and the same happens for quality. It is therefore important to appropriately choose the right initialization parameters for each sequence, especially the number of CRF points and the extent of the considered interval. It might be precisely a waste of computational resources to encode in the entire range, but on the other hand the system may not reach the best solution if the interval is too narrow.

CRF	0	7	15	22	29	36	44	51
Bitrate (kbps)	588466	309944	92030	15116	3630	1359	579	174
Quality (VMAF)	99.99	99.93	97.15	90.98	75.89	44.38	18.00	8.36

Table 3.1, RQ values for each CRF point of the *in_to_tree* clip, characterised by a low coding complexity.

CRF	0	7	15	22	29	36	44	51
Bitrate (kbps)	633897	303257	92718	28110	10685	4108	1427	650
Quality (VMAF)	99.99	99.99	99.98	96.18	79.43	50.76	17.67	4.26

Table 3.2, RQ values for each CRF point of *crowd_run*, a high coding complexity sequence.

The third custom parameter in the list above is the distortion metric. For each target, the software is expected to produce an optimised output video file, encoded using the resulting combination of parameters and in compliance with the given target. This file is evaluated in order to quantify its quality and amount of distortion, using VMAF as a quality metric. The software also supports the PSNR metric especially for testing reasons, to compare its performances with the main quality metric. In both cases, the assessment produces a rate-quality pair for each final video, so for each specified target value. We must specify multiple target values when optimising the test sequence because just one or two RD points are not enough to ensure a comprehensive analysis and to construct an RD curve. A curve instead allows to describe the behaviour of the video at a multitude of quality and bitrate points, and so to characterise it. Finally, the last entry in the list concerns input files.

Test video sequences

The selection of test scenes is an important issue. When dealing with testing of digital video sequences it is worth considering and covering all possible cases, and generalising results for the wider spectrum of video contents without exceeding in the number of tests. All sources are taken from the three video sets listed in *Table 3.3*, largely used in current video testing studies. We have selected all the test sequences for this study (*Table 3.4*) based on the ITU recommendation (2008), which suggests considering spatiotemporal information.

Spatiotemporal information is a metric used to describe the amount of spatial and temporal complexity in a video scene (Robitza, 2021). It is distinguished into two distinct components, Spatial Information (SI) and Temporal Information (TI). The former measures the spatial details in frames, which include features such as edges, textures, and patterns that generally indicate more complex scenes. The latter on the other hand, measures the level of temporal changes present in a video sequence as a simple difference between adjacent frames, and is typically higher for sequences with more motion.

#	Set name
p01	SVT High Definition Multi Format Test Set - 2006
p02	SVT Open Content Video Test Suite 2022 – Natural Complexity
p03	Netflix Open Source Content

Table 3.3, test sets.

#	Res	Sequence name	Frame count	Frame rate	Input	Description
01	SD	elephants dream	15.691	24	raw	Synthetic content sequence
02	HD	big_buck_bunny	14.315	24	raw	Synthetic content sequence
03	FullHD	rush_field_cuts	570	29,97	raw	5-shots natural sequence
04		snow_mnt	570	29,97	raw	3-shots content with dissolve transitions
05		in_to_tree	500	50	raw	Single-shot natural content from <i>p01</i> . Coding complexity: easy
06		crowd_run	500	50	raw	Single-shot natural content from <i>p01</i> . Coding complexity: difficult
07		edit_svt_10sec	1.250 and 2.500	25 and 50	raw	Edited sequence; 10-secs long shots from the 5 clips of <i>p01</i> (both low and high complexity)
08		edit_svt_mix	1.250 and 2.500	25 and 50	raw	Edited sequence; 1-to-5-secs long shots from the 5 clips of <i>p01</i> (low to high complexity)
09		edit_svt_fullmix	12.000 and 24.000	25 and 50	raw	Edited scene; a random sequence of <i>edit_mix</i> and <i>edit_10sec</i> clips
10	UHDTV	meridian	43.093	59,94	raw	Short film made of challenging shots, representative of the existing titles on Netflix
12	DCI 4K	bar_dinner	1.200	30	raw	Edited 7-shots sequence; <i>bar_scene</i> and <i>dinner_scene</i> from <i>p03</i> concatenated
13		edit_nature	1.199	60	raw	Edited sequence; randomly mixed shots from <i>p02</i> (low to high complexity)

Table 3.4, test sequences.

These two parameters play an important role in determining the amount of possible video compression and in our case, the subsequent level of impairment that affects multimedia information when it is delivered over a network. Compression difficulty is directly related to the SI and TI values of a sequence (Robitza et al., 2021). Indeed, to provide a reliable and effective assessment, it is important to choose test sequences with SI and TI consistent with the Dynamic Optimizer use cases. Considering the ideal contents and use conditions described in the previous chapter, the set of test samples should cover the widest possible range of SI and TI ratings. For each scene, these values can be calculated and plotted on a spatiotemporal information plane. Hence, *Figure 3.2* shows the amounts of SI and TI for some representative test scenes. Still scenes or sequences with very limited motion, like *meridian*, are located at the bottom of the plot where TI is close to 0. On the right-hand side of the plot instead, where SI is higher, we can find scenes with maximal spatial details, like *snow_mnt*. Finally, in the top-right corner are found scenes with both the greatest detail and motion, like *crown_run*.

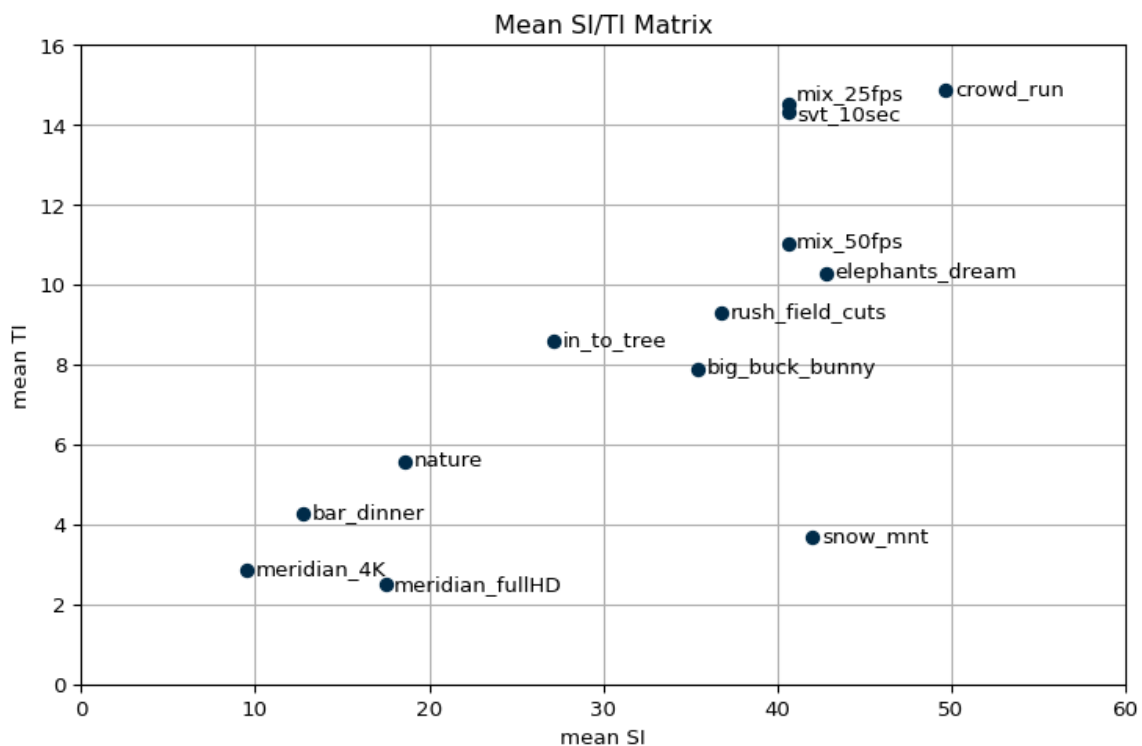


Figure 3.2, SI/TI Matrix.

The number of selected test sequences is also crucial, and again according to ITU-T (2008), it should not be excessive. Since just a limited number of samples are to be selected, and in order to ensure coding difficulty heterogeneity, it is ideal to choose sequences from each of the four quadrants of the plane.

As already mentioned, input files are high definition and high quality test sequences, both raw and compressed files. Quality is a broad concept that encompasses various factors with an impact on the overall perception of a video sequence. In computer networks and telecommunication, quality is seen as a fidelity measure. Fidelity measurements quantify the degree to which a representation accurately represents the original signal or data. In this context, it refers to how closely the compressed or transmitted video sequence matches the original uncompressed sequence in terms of visual quality. In terms of fruition and enjoyment instead, it measures the richness of the viewer experience and the degree to which a video sequence meets the expectations of the viewer or the application.

It differs from high definition, that is rather a type of video quality that refers to any higher resolution than standard definition video. While high resolution is an important aspect of video quality, it is not the only determining factor. Frame rate, colour depth, dynamic range, and compression efficiency also play a critical role in enhancing video quality. For example, a higher frame rate results in smoother and less jerky motion, while a wider colour depth or dynamic range is able to include a greater part of the spectrum of colours and brightness. Compression also impacts the quality of a video, as it usually leads to visual artefacts and reduced detail.

Format	Resolution (px)
DCI 4K	4096 × 2160
UHDTV	3840 × 2160
FullHD	1920 × 1080
HD	1280 × 720
SD 16:9	640 × 360

Table 3.5, video resolutions of the tested sequences.

#	Sequence name	In resolution	Out resolution
14	#03 rush_field_cuts	FullHD	SD
15	#10 meridian	UHDTV	SD
16			HD
17			FullHD

Table 3.6, scaled sources.

Besides spatiotemporal complexity and duration, the test video sequences are characterised and were selected based on the following technical features.

- **Resolution.** The set of reference videos includes resolutions from SD to 4K according to the information listed in *Table 3.5*. Some of them, for analysis purposes, were scaled as indicated in *Table 3.6*.
- **Frame rate.** It is the number of frames per second (fps) and the system supports any fps value. It has to be explicitly specified only for *.yuv* files as they do not contain header information.
- **Bit depth.** It is the precision of the pixel representation and is always 8-bit. When a 10-bit video is provided, since encoders are configured to work in standard profiles, the converted elemental encodes are always in 8-bit format.
- **Chroma subsampling.** The colour scheme of the videos is always 4:2:0, although any other video with a different supported scheme can be imported.
- **Input standard and container.** The common input format is raw video in a *.yuv* or *.y4m* file. The difference between the two is that YUV requires additional parameters to be explicitly specified when importing. In addition, compressed video sources are also supported, but the available standards and container formats depend on the FFmpeg installation.

3.2 OPTIMIZATION RESULTS

When comparing different sequences optimised using CRF, it is necessary to have a reference point, or a baseline. This is where the concept of Fixed CRF comes into play. In contrast to the dynamic optimization that produces different CRF values for each coding unit, the Fixed CRF encoding method maintains a uniform level of quality throughout the entire video by adapting the compression rate according to the complexity of the scene. But in this case, each shot is encoded using the same CRF value, which means encoding the whole sequence uniformly at a single CRF point, without dividing it into separate shots. It provides a benchmark for video quality assessment because it allows the comparison of optimised outputs with a fixed baseline. In this sense, we can evaluate the effectiveness of different encoding methods and compare their performance under different conditions.

Henceforward, the symbol FX on the charts will stand for Fixed CRF, while the following acronyms, BF, LG, and CF will be employed to refer to the corresponding optimization methods: Brute Force, Lagrangian, and Curve Fitting. The charts represent RQ curves, as part of the rate-distortion theory explained at the beginning of this chapter. All encodings were performed using virtual instances running Ubuntu Linux OS on servers with an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz.

Fixed CRF and Brute force

To compare the performance of Fixed CRF and Brute force encoding methods, short video clips consisting of a few shots were used, and a limited number of CRF values were tested to avoid an exponential increase of the complexity. The computational time spent to test each combination depends on the number of combinations but also on the hardware processing power. Due to the limited number of elemental encodes tested however, the results may not be fully representative. Besides this limitation we can still identify a general trend for BF, for example in *Figure 3.3* with the AV1 encoder. There, BF has a slight advantage over FX, especially at low and intermediate bitrates. Also in *Figure 3.4* and *Figure 3.5* AVC encoding allows a quality gain over FX for the same bitrate, both on a FullHD and a 4K resource. Indeed, from these curves we start distinguishing differences on the optimization performances among encoders and between low and high bitrates. From now on, we will refer to the relative Low-Bitrate, Medium-Bitrate and High-Bitrate parts of the curves as LB, MB and HB.

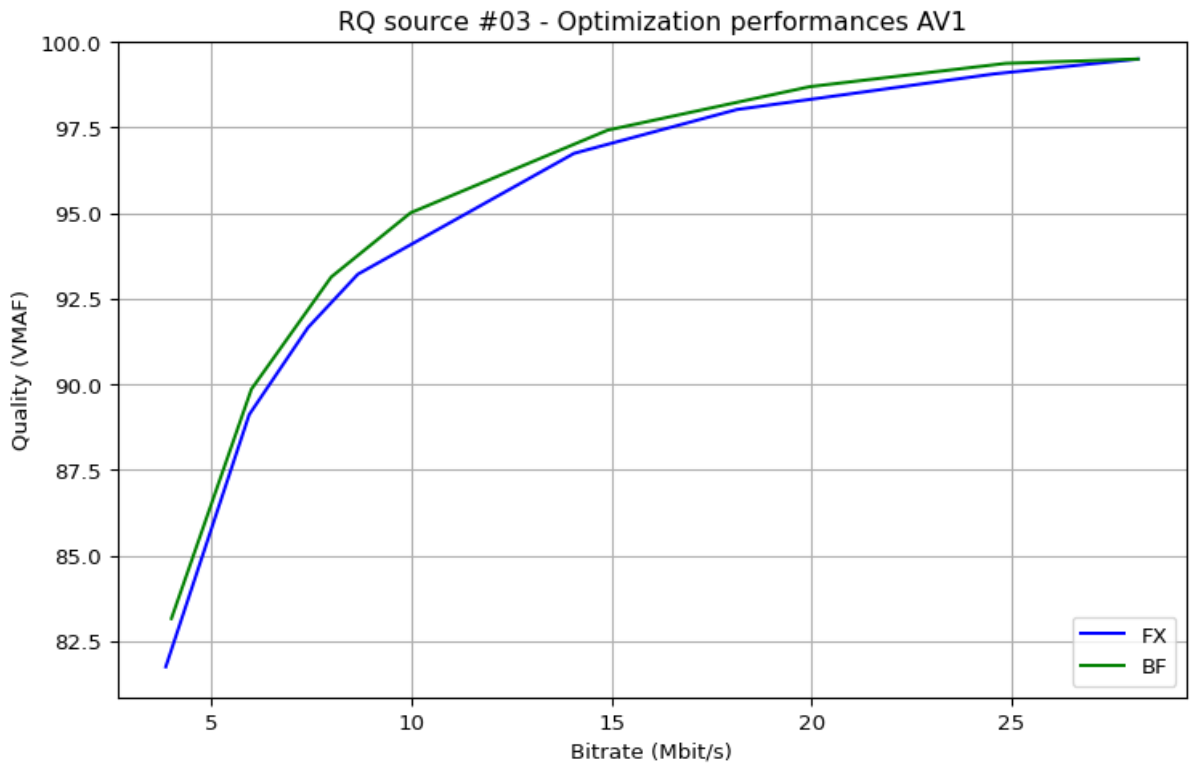


Figure 3.3, FullHD sequence AV1 encoded at 6 CRF points in between 25 and 55.

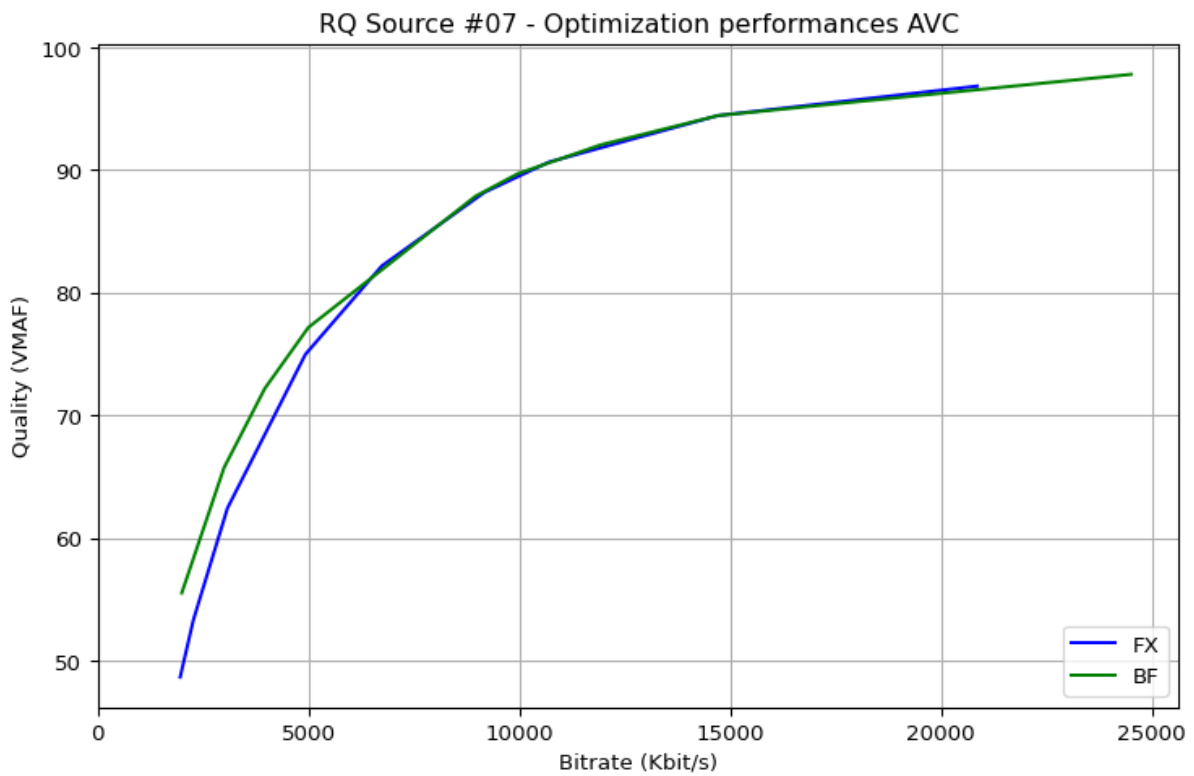


Figure 3.4, FullHD sequence AVC encoded in the CRF range [15,40] and at 16 CRF points.

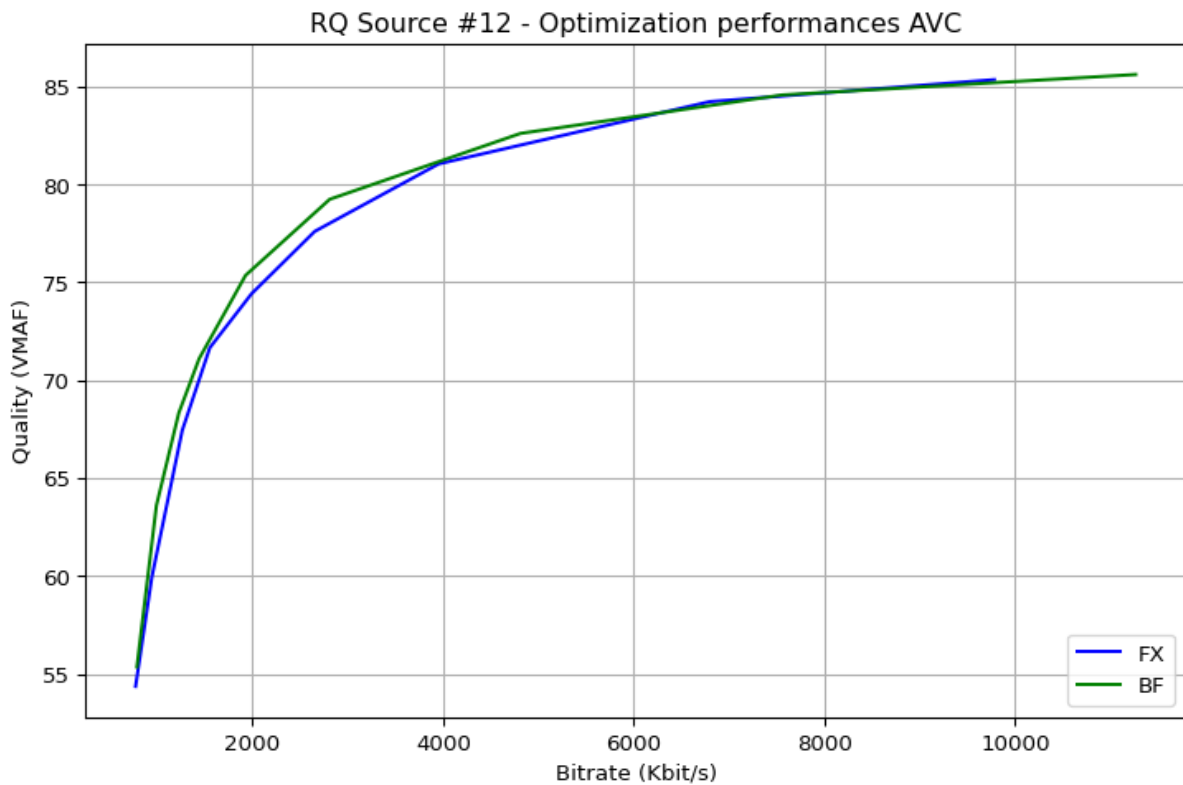


Figure 3.5, 4K AVC encoding. 12 CRF points in the range [20,40].

Fixed CRF and Lagrangian methods

In this section, we compare the optimization performances of two Lagrangian optimization techniques, the pure Lagrange method with exhaustive coding, and the one that uses curve fitting to avoid encoding redundancies. FX is used as a reference instead of BF, particularly in cases where it is not feasible to run the brute force algorithm. As shown in *Figure 3.6*, the RD curves for the four techniques are plotted on a small area of the RD plane for better clarity. It is challenging to plot the RD curves for the four techniques together due to their distinct characteristics, varied use cases, scenarios, and requirements. This is the reason why in all the other figures we decided to distinguish them, to provide a better understanding of their individual performances. Among the four methods, the baseline FX method exhibits the poorest performance, followed by the CF method, which is probably limited by its approximation degree. The LG method performs better than CF, while the BF method is the most accurate and serves as the reference for the best optimal solution. By looking at the distance between the curves we can estimate a gain in quality for LG against FX up to 8% and 6% for CF over FX (*Figure 3.9*), and the same applies to bitrate gains.

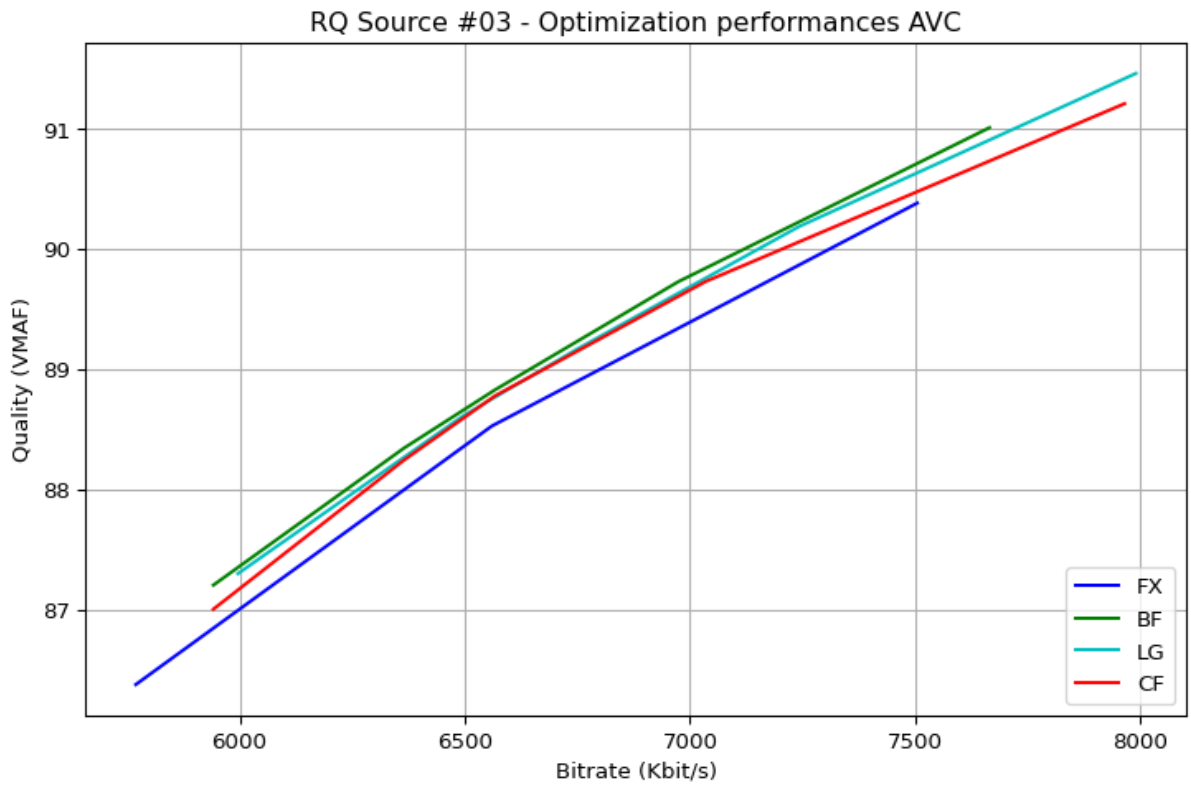


Figure 3.6, RQ curves of the four methods. 12 elemental encodes per shot were encoded using AVC in the range [19,31].

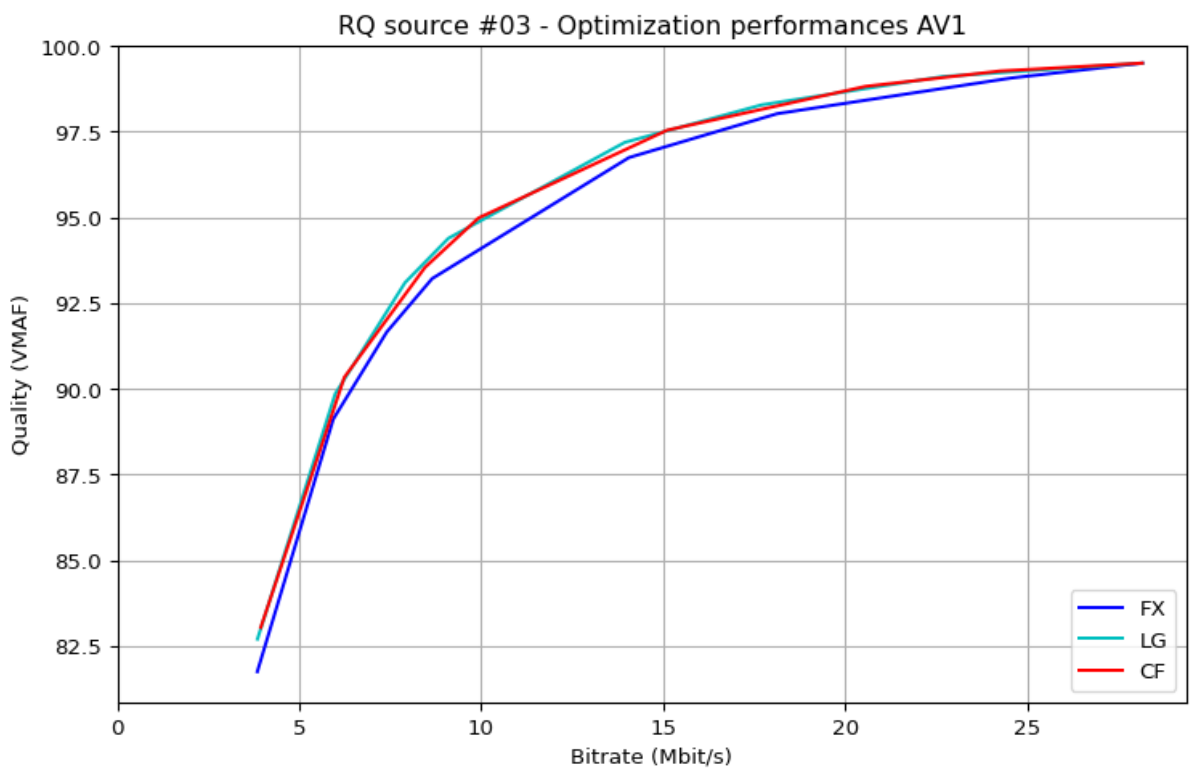


Figure 3.7, AV1 full-points encoding in the range [15,40].

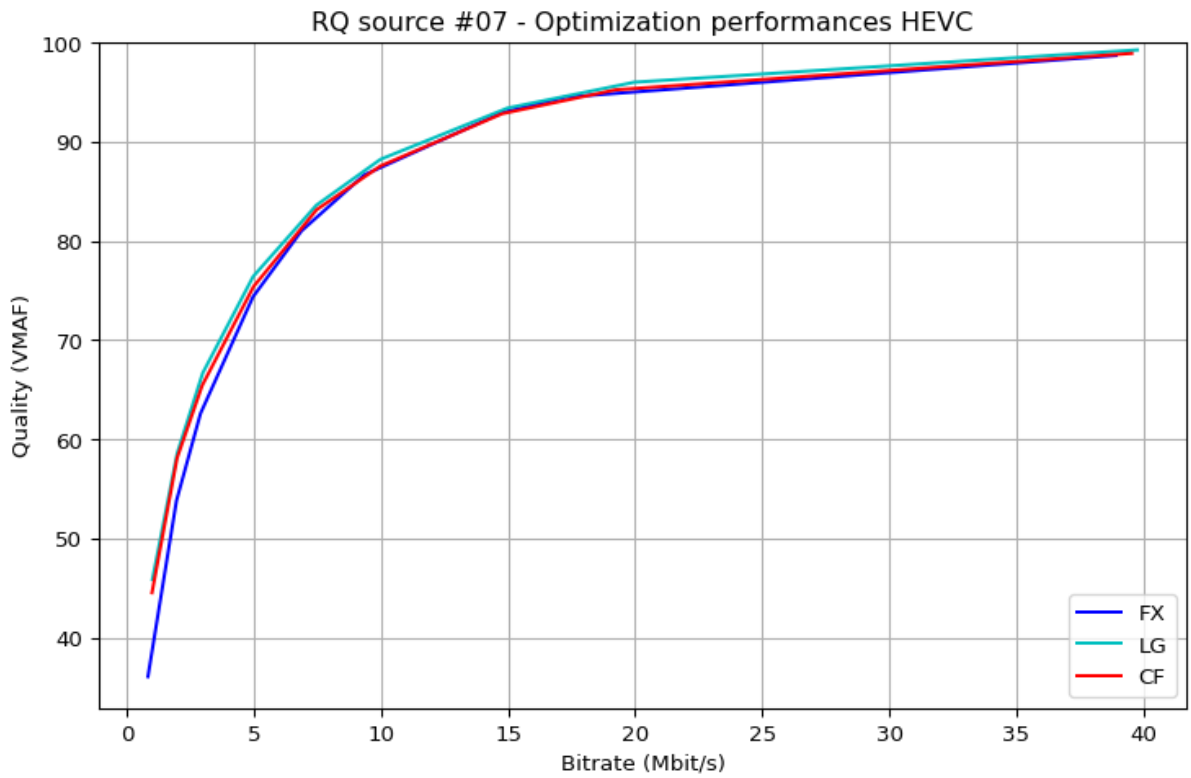


Figure 3.8, HEVC full-poins encoding in the range [15,45].

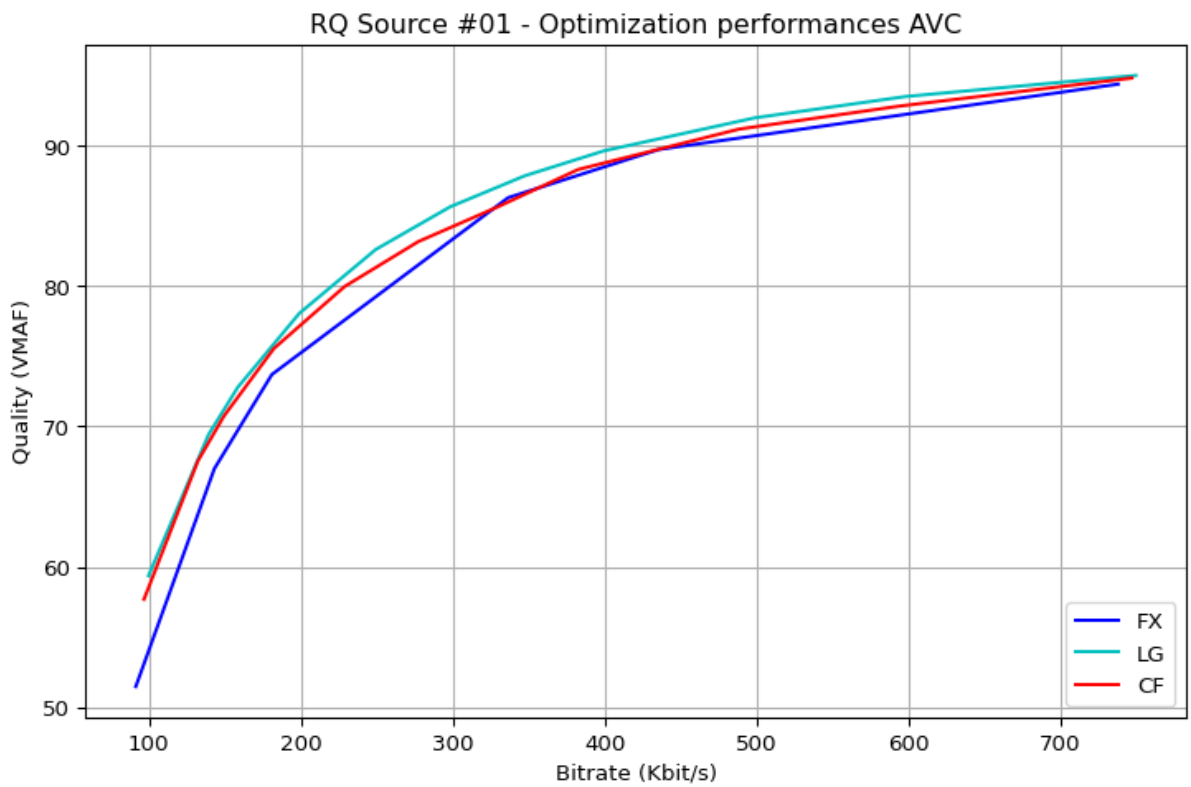


Figure 3.9, AVC encoding in the range [20,45] at 13 points for FX and LG, and 5 for CF.

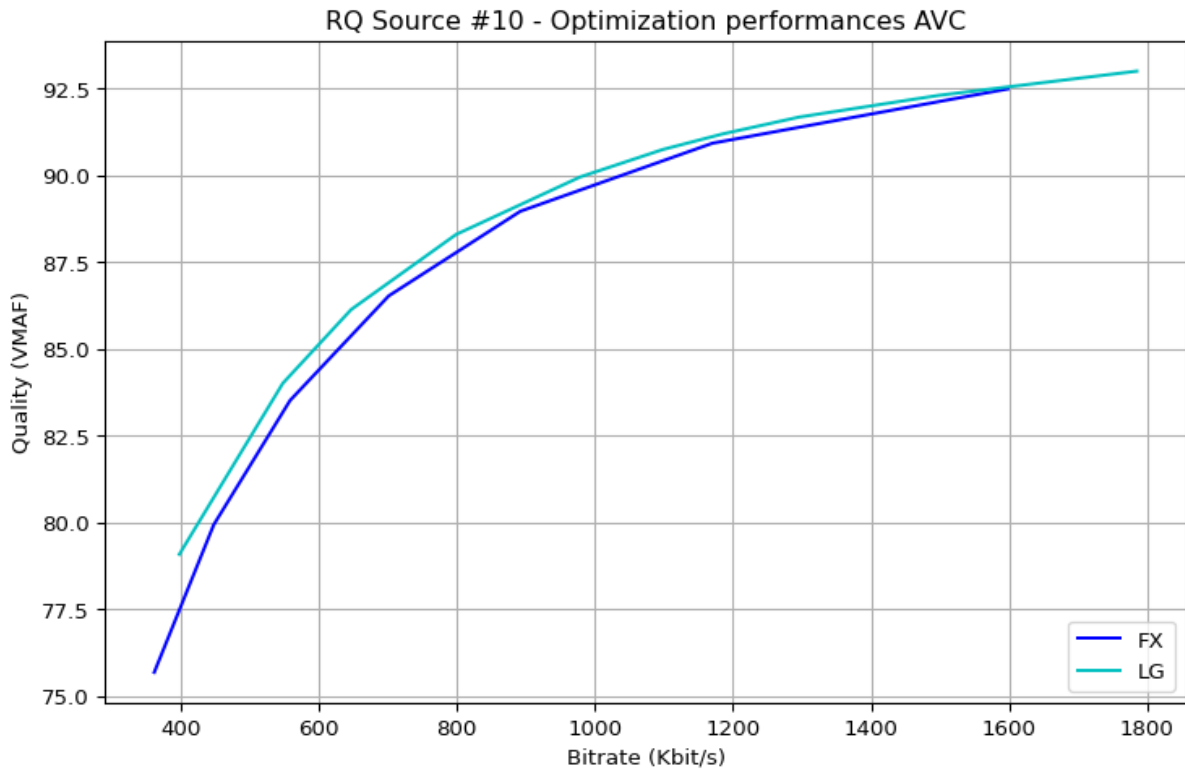


Figure 3.10, AVC encoding in the range [10,40] at half points for FX and LG, and 5 for CF.

Despite the limited number of shots, the comparison of optimization performances between the Lagrangian methods with exhaustive coding and curve fitting techniques and the Fixed CRF reference demonstrate noticeable differences. As depicted in *Figure 3.8* for AV1 and in *Figure 3.9* for HEVC, both Lagrangian methods outperform FX, particularly in LB. The trend is even more significant when longer test sequences are considered, as illustrated in *Figure 3.9*, which contains an SD animated sequence and in *Figure 3.10*, a natural FullHD content. The dependency on content will be further discussed in *Chapter 3.3*.

Finally, the comparison between the Lagrangian and brute force methods reveals the efficacy of both. Although their performances are similar, with the two curves almost overlapping as seen in *Figure 3.11*, the LG proves to be an efficient optimization technique, delivering results that are close to the best reference while significantly reducing computational complexity, in this case with a difference below 1%. This is true for cases where the number of points in the interval is dense enough, for example 12 CRF points in the interval [20, 40], more than one out of two. However, when the number of points is low, the LG may not sufficiently represent the curve behaviour, thus returning worse results. In these cases it is more advantageous to use CF. In *Figure 3.12* for example, only 6 elemental encodes per shot were encoded and then optimised using both LG and CF.

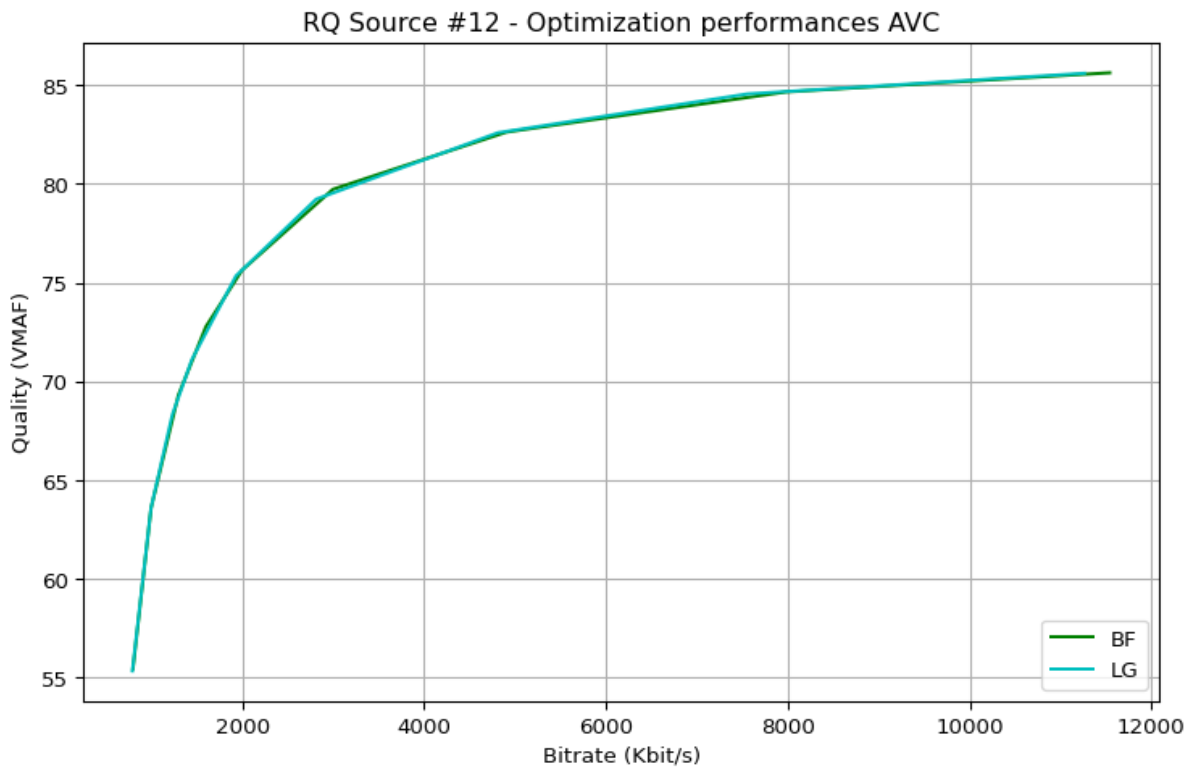


Figure 3.11, AVC encoding at 12 points in the range [20,40].

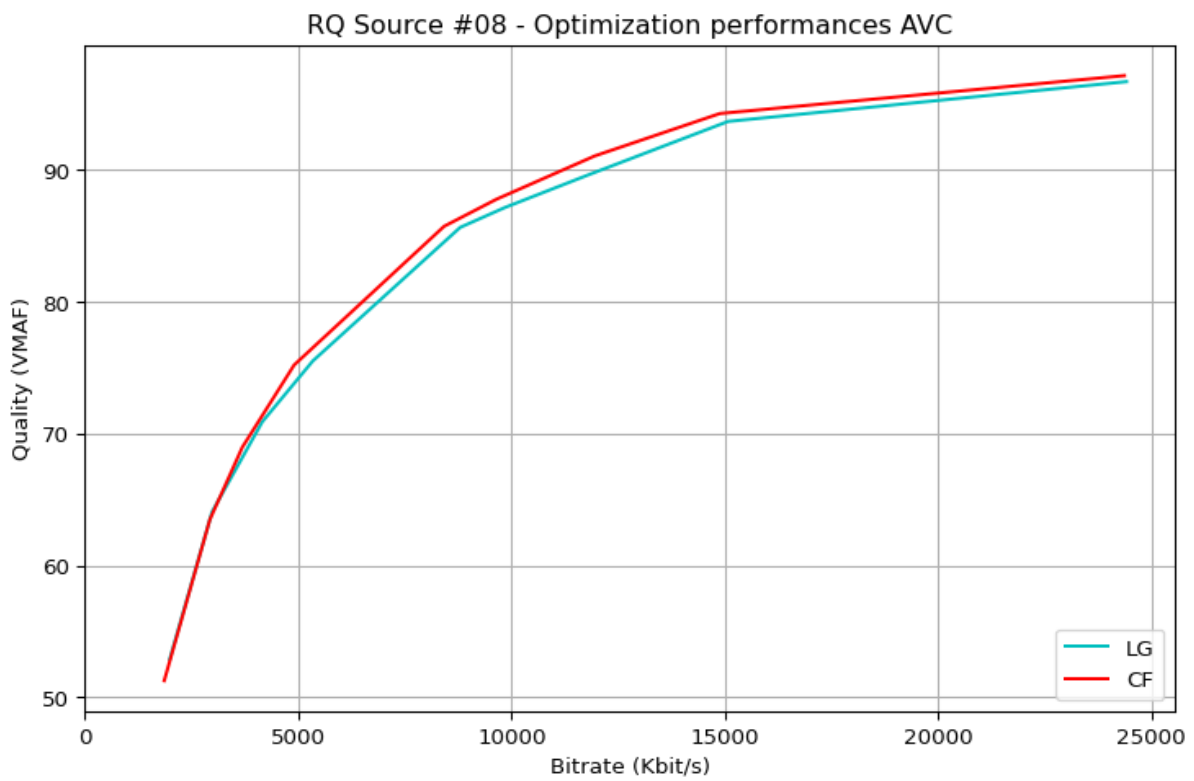


Figure 3.12, AVC encoding at 6 points in the range [15,40].

Number of elemental encodes

Encoding parameters like the number of CRF values and the range of CRF points used, play a crucial role in optimising video quality, as previously discussed in this chapter. It is important to note that these ranges vary depending on the test sequence, optimization method, and encoder used. Therefore, it is important to analyse how optimization behaviour changes for the presented methods with changes in the number of elemental encodes. As previously stated, the Lagrangian method performs better with denser points, whereas CF only works with a limited number of points since it is able to approximate the missing ones. In *Figure 3.12* and *Figure 3.13*, both optimised based on a few points, one can notice differences in the optimization performance between these two methods especially in MB. However, at the extremes of low and high bitrates, these methods tend to coincide, producing similar results.

In addition, it should be noted that also FX, BF, and LG are capable of operating with fewer points, but unlike CF, the optimization processes will only take into account the points that have been encoded instead of considering the entire range. *Figures 3.14* and *Figure 3.15* confirm that even though LG performs well with half the number of points, the best outcome is obtained at full points. This is true for FX as well, particularly in MB, but even in LB disregarding the left extreme.

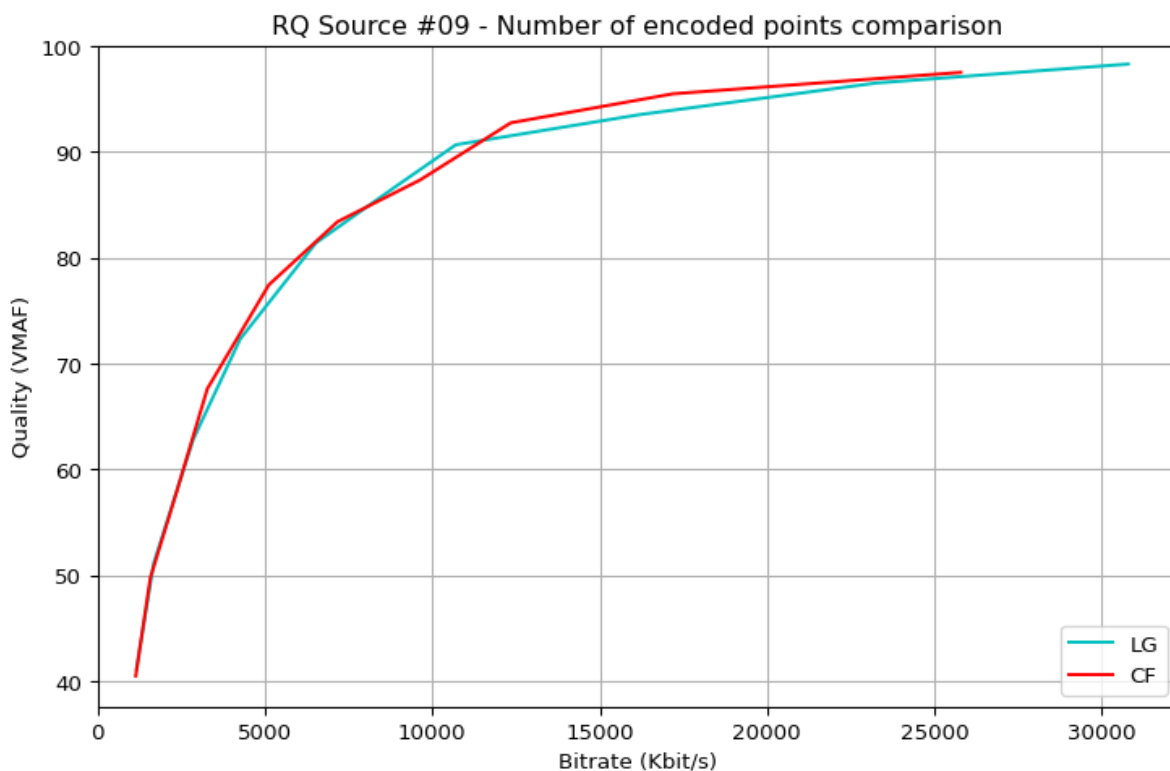


Figure 3.13, AVC encoding at 5 points in the range [15,40].

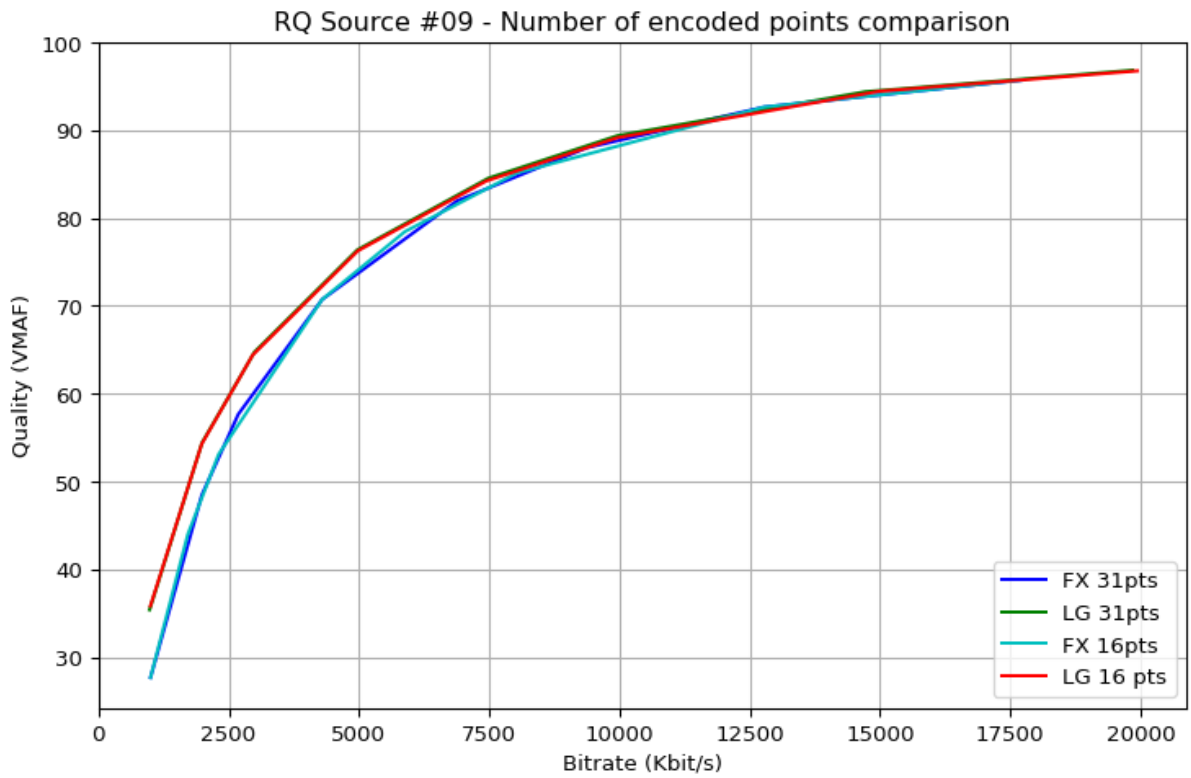


Figure 3.14, AVC encoding, [15,45] range.

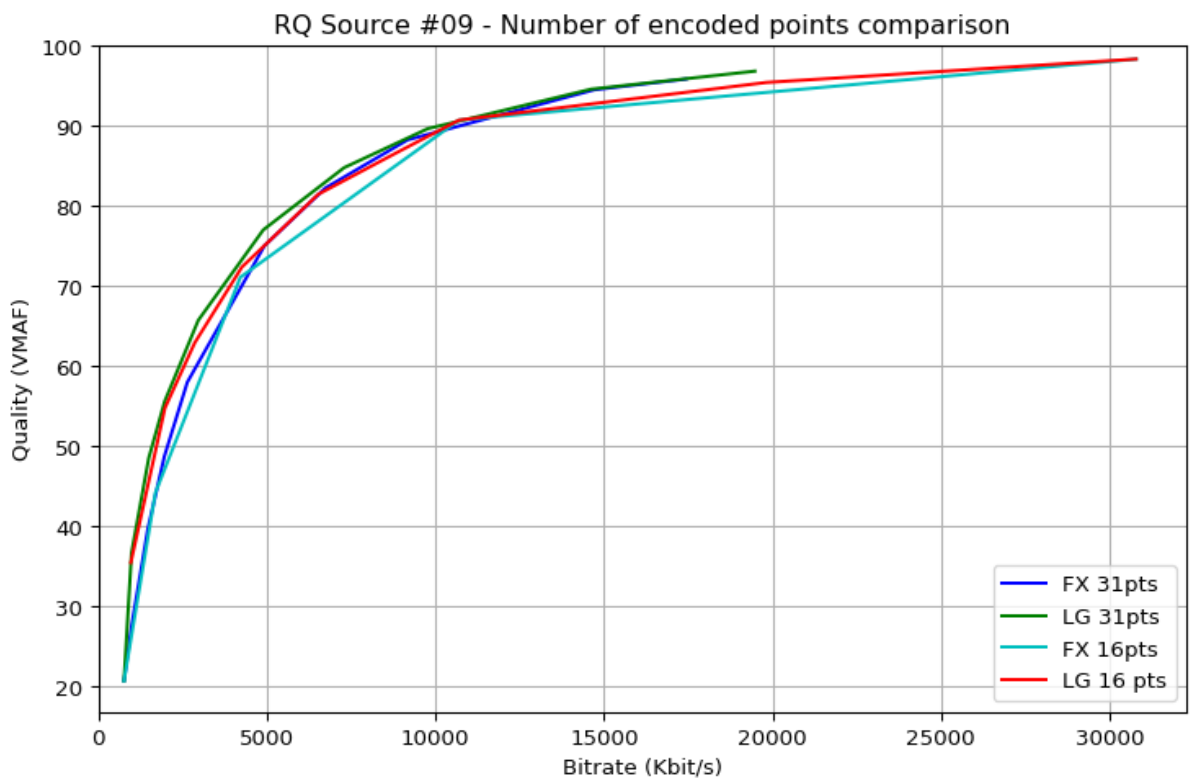


Figure 3.15, AV1 encoding, [25,55] range.

VMAF and PSNR

PSNR has been the traditional metric in video quality assessment for many years, but nowadays VMAF is becoming more and more popular. Nonetheless, the Dynamic Optimizer is designed to work also with PSNR, either minimising quality with a bitrate limit or setting one or more PSNR quality targets and minimising the bitrate. In this section, the optimisation performance of LG is evaluated under both metrics, by setting to the same source the same bitrate target. In *Figure 3.16*, the y-axis shows VMAF, and the first curve was generated by optimising VMAF, while the second one was optimised using PSNR but its corresponding VMAF values are plotted. Indeed, the algorithm returns both VMAF and PSNR values for each elemental encode, so they can be used interchangeably. The opposite is shown in *Figure 3.17*, where the y-axis displays PSNR values, even though the first curve was optimised using VMAF. While in the second chart there is almost no difference between the two versions, in the first one (*Figure 3.16*), where VMAF is used as the reference quality metric, we can notice an albeit minimal advantage in HB and around 4 Mbps of bitrate.

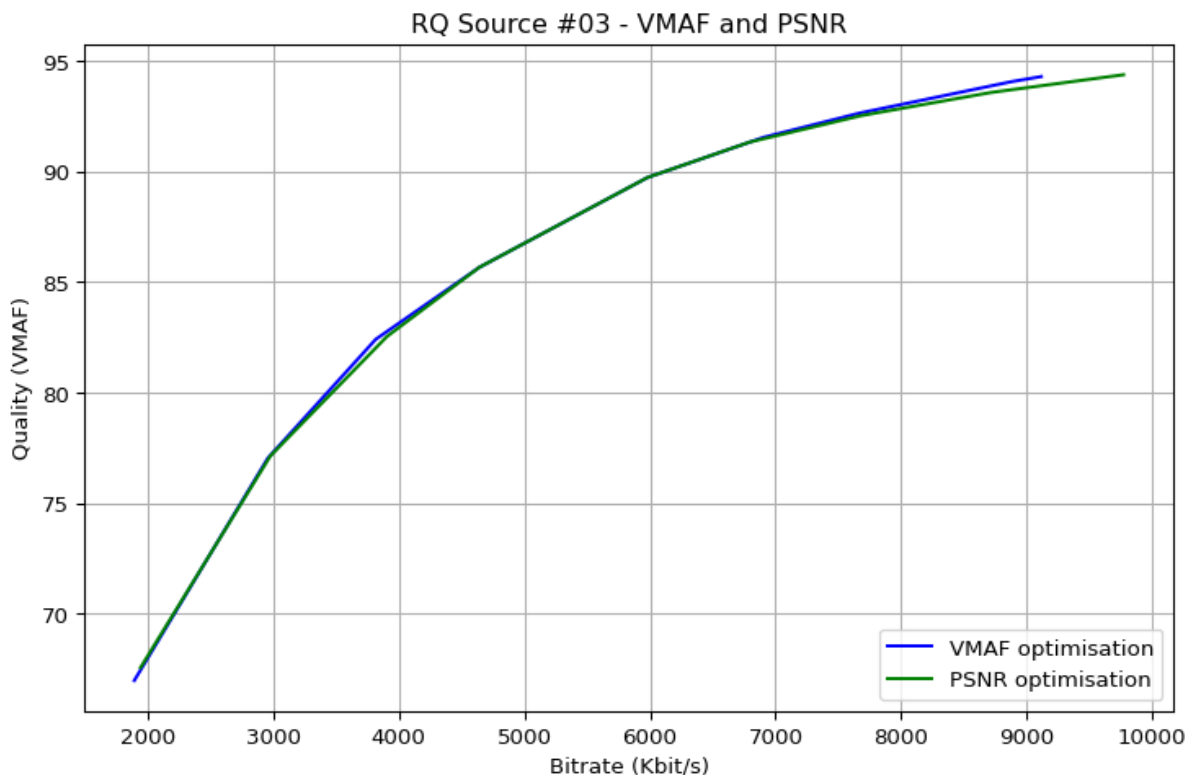


Figure 3.16, VMAF-based optimisation with AV1 encoding in the range [25,60] at 15 points. The difference is minimal due to the short duration of the sequence.

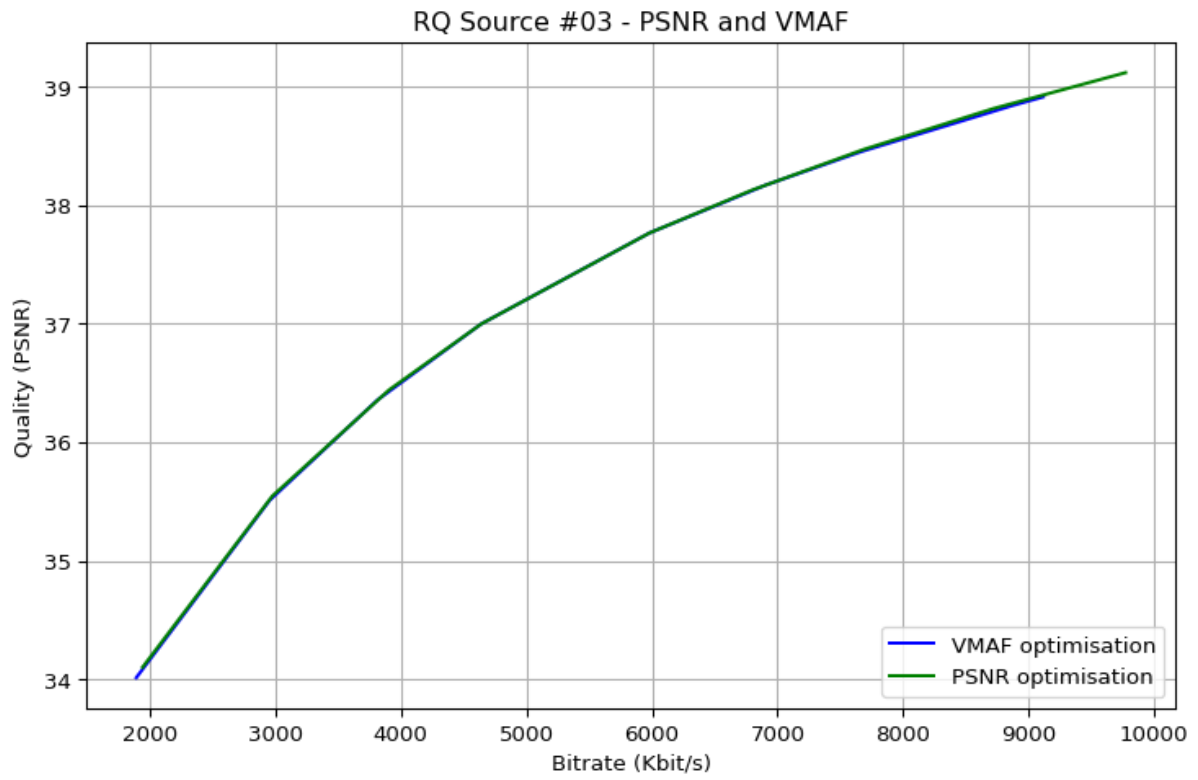


Figure 3.17, PSNR-based optimisation with AV1 encoding in the range [25,60] at 15 points. In this case the difference is hardly noticeable.

3.3 CONTENT DEPENDENCY

This section aims to investigate whether there is a relation between optimization performances and content type and complexity of the video test scenes. In selecting the test sequences, we took into account both the spatial and temporal information that describe videos in terms of fine details and motion. It is noteworthy that all three test sets (*Table 3.3*) from which we took the test sequences came with a detailed description of the encoding complexity of each scene. To explore this relationship, we generated a sequence by concatenating 5 shots from the *p01* set, each with varying degrees of complexity. The first and last shots, in particular, were far more complex than the others, as shown in the SI/TI matrix in *Figure 3.18*. By assessing the quality of the sequences obtained from the Dynamic Optimizer using FX, LG, and CF with a 78 VMAF score target, we observed that the scores fluctuated (*Figure 3.18*) often in response to the complexity of the scene, in order to try to maintain consistent the level of quality. However, the three methods exhibited different behaviour depending on the complexity of the shots.

Both in *Figure 3.18* and *Figure 3.19*, FX never achieve the highest quality, and in instances of low quality peaks, CF tends to be more present. For the highest quality peaks, LG performs the best, as it attempts to optimise the parameters more than the other methods. On the other hand, in high complexity scenes, CF consistently performs at the same or at a lower level than the other methods. This may be due to the fact that complex shots have a more pronounced RD curve that is less smooth and soft than a standard complexity shot curve. This results in a less precise fitting algorithm that causes the estimated points to deviate from the actual curve, resulting in a drift from the optimum. Conversely, in low complexity shots, the RD curve is less concave and less inclined, with a benefit on the final quality optimisation.

Ultimately, we examine the distinct trends of the four chosen encoders across two primary categories of video content, natural and synthetic. Here, the term synthetic is used to describe animated pictures created by computer-generated graphics. Both for synthetic and natural video content, AV1 outperforms all other encoders in terms of optimization performance and its behaviour is consistent across the two types of content. Interestingly, in synthetic video (*Figure 3.20*), HEVC yields better performance than VP9, while there are no significant differences with the trends of the other optimisations. On the other hand, for natural video (*Figure 3.21*), VP9 generally outperforms HEVC, except for a small portion of LB where HEVC wins. Lastly, AVC regrows in HB over HEVC, nearly reaching VP9.

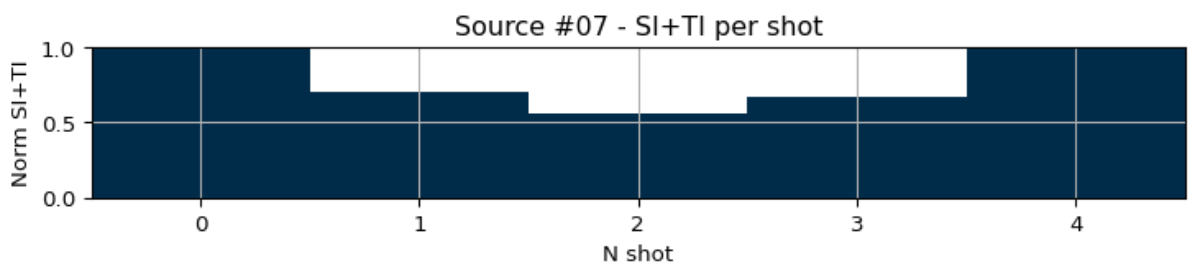
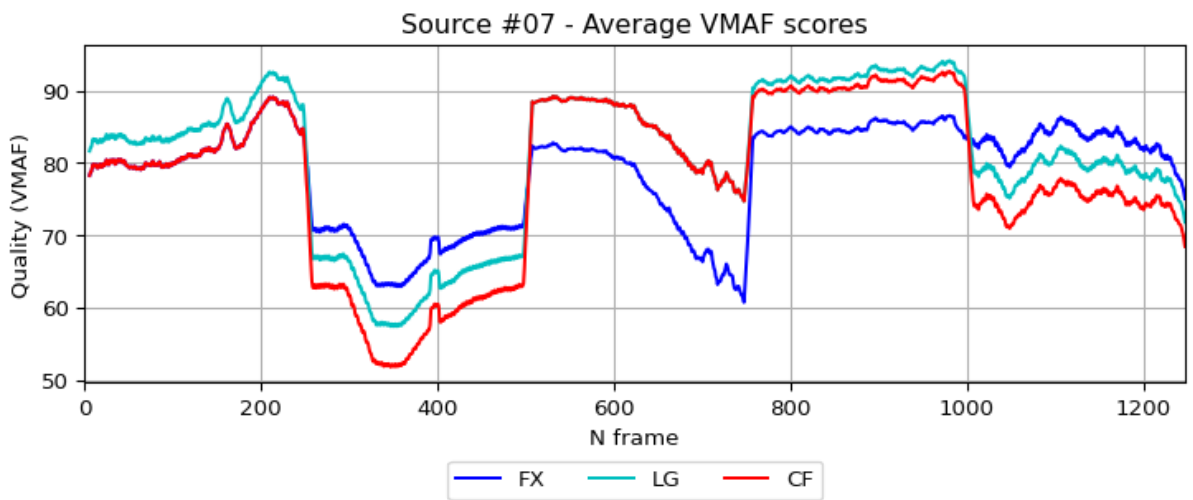
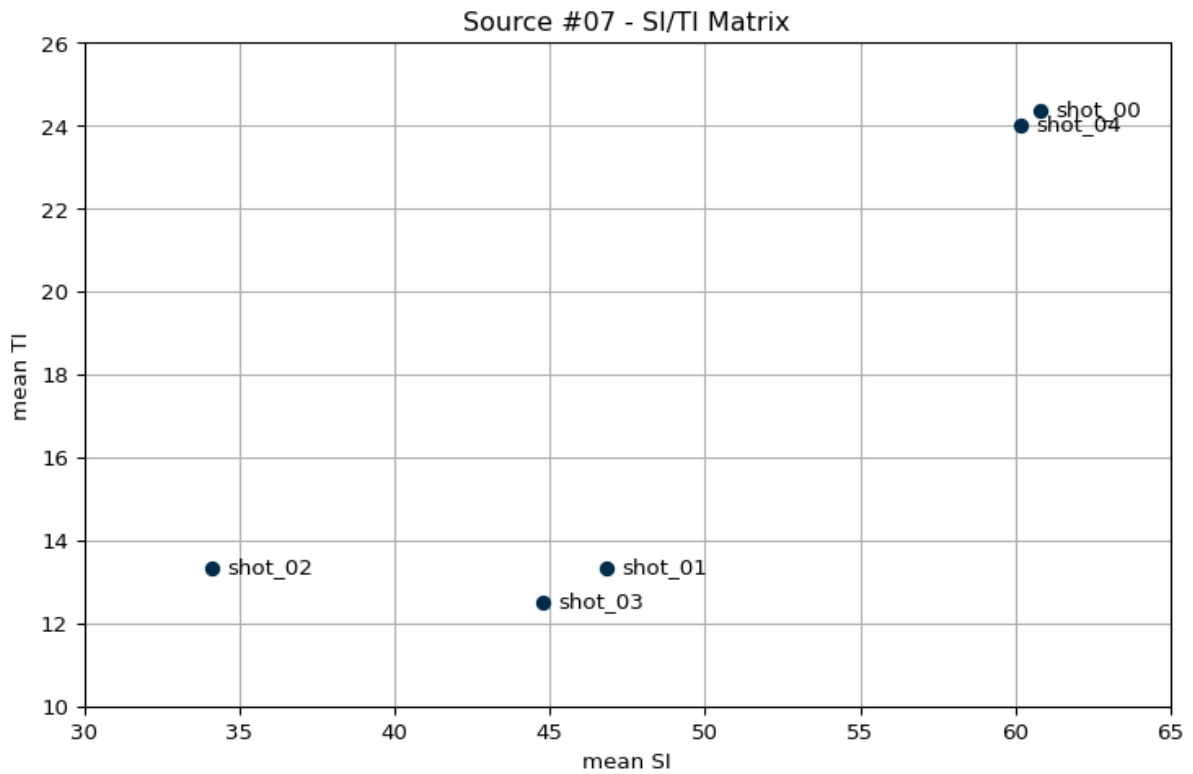


Figure 3.18, at the top, the SI/TI matrix for the 5 detected shots of *edit_svt_10sec*. In the middle, per-frame VMAF scores averaged with a window of 10 points wide. At the bottom, sum of the SI and TI scores per shot mapped onto the range [0,1].

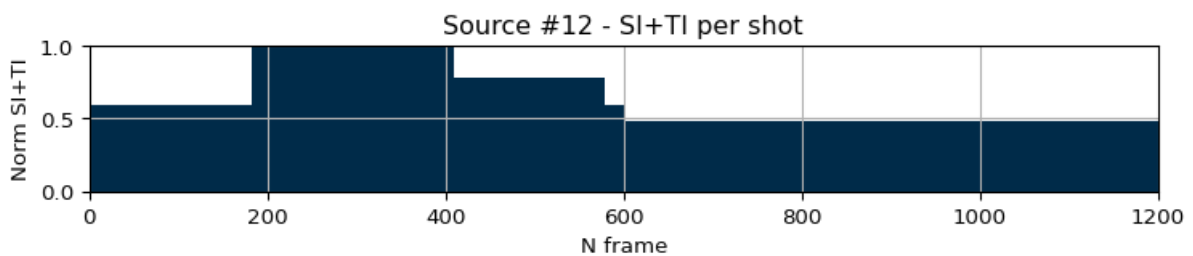
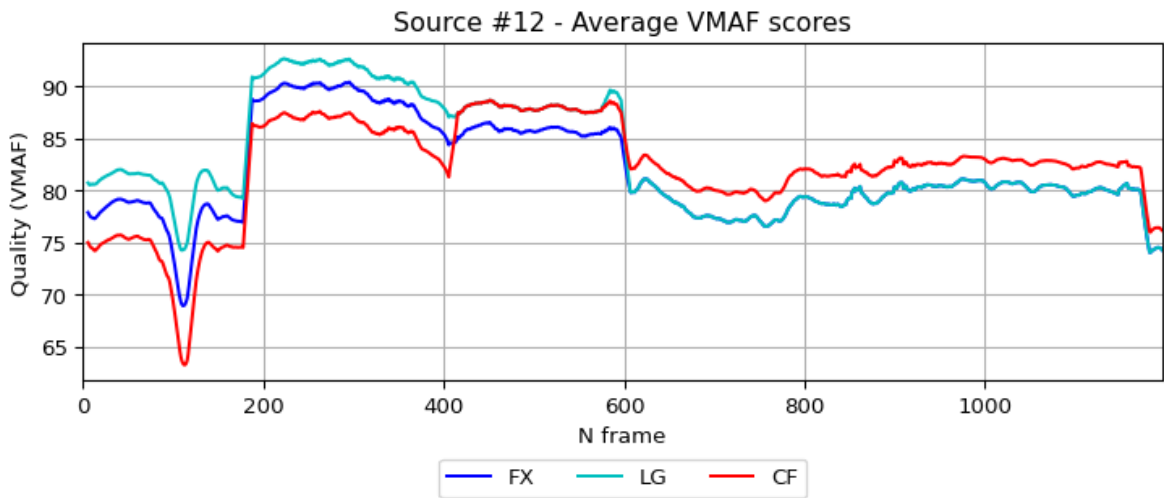
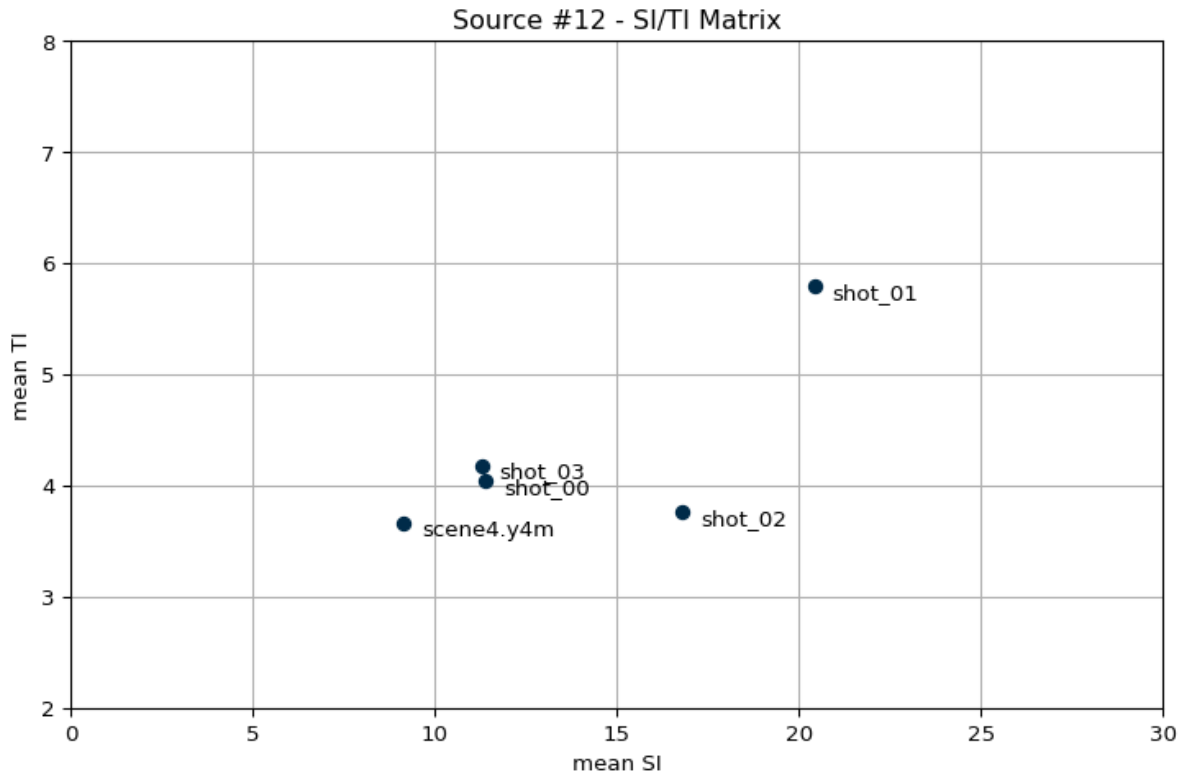


Figure 3.19, at the top, the SI/TI matrix for the 4 detected shots of *bar_dinner*. In the middle, per-frame VMAF scores averaged with a window of 10 points wide. At the bottom, sum of the SI and TI scores per shot mapped onto the range [0,1].

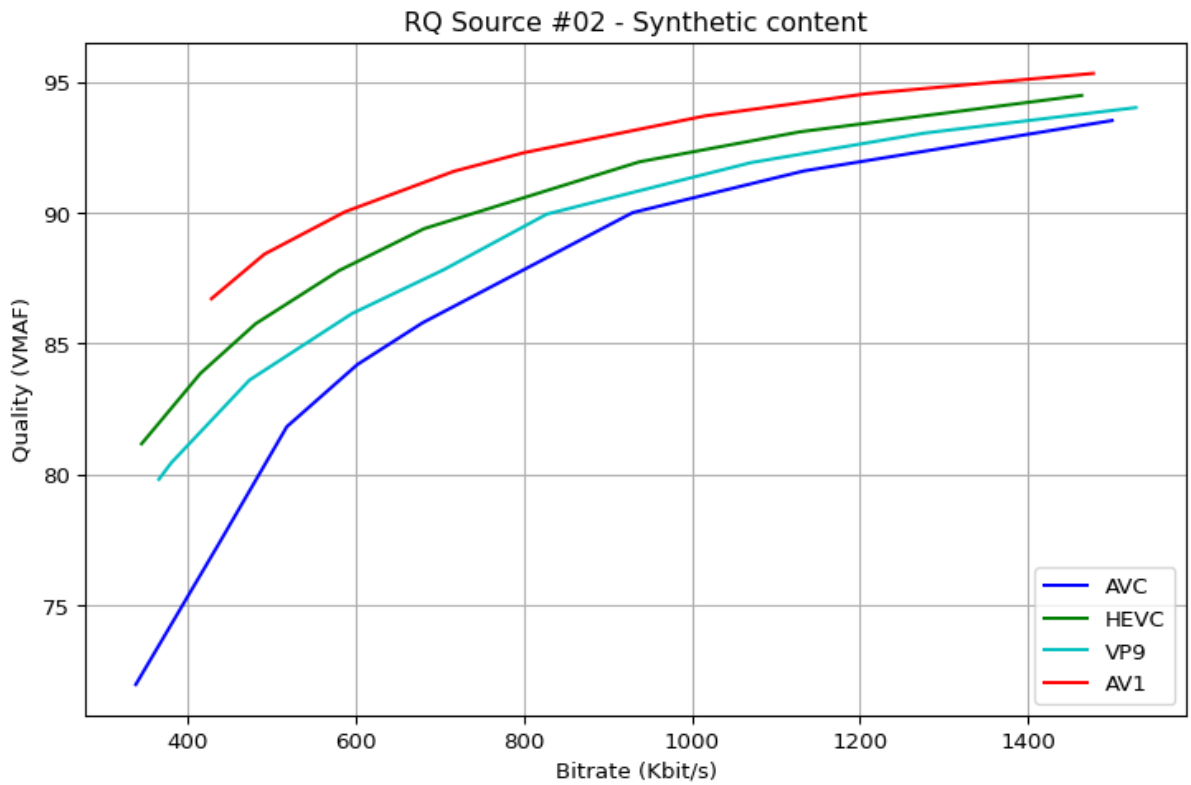


Figure 3.20, CR encoding at 5 points, [15,40] AVC and HEVC, [15,50] AV1 and VP9.

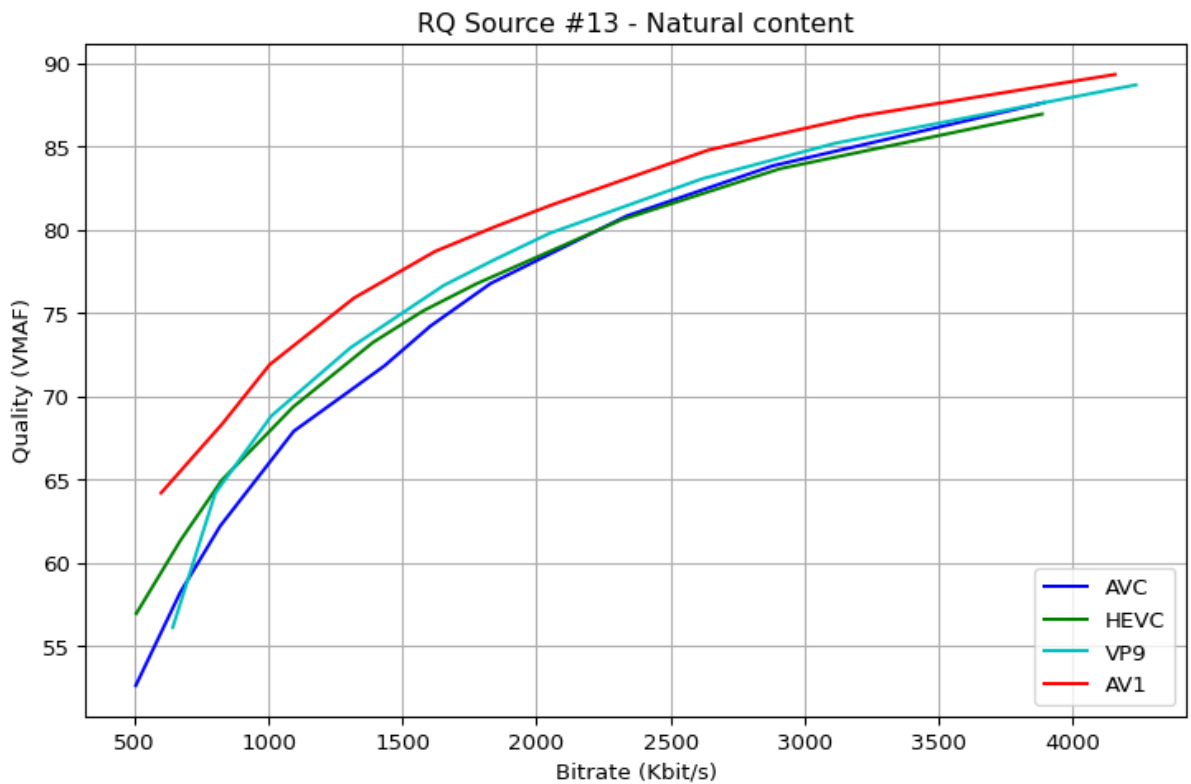


Figure 3.21, CF encoding at 5 points, [20,45] AVC and HEVC, [20,55] AV1 and VP9.

3.4 IMPLICATIONS

Several key takeaways can be gathered from the previous data and from the overall study, which started from the work of Katsavounidis and Guo (2018) and their first paper on dynamic optimisation in Netflix. The paper was our starting point, but now it becomes a precious resource to compare final results. As regards the first finding, it is clear that the Dynamic Optimizer is most effective when applied to long sequences with a range of complexities, as demonstrated by its good performance on *Source #10 Meridian* from Netflix itself. However, the results appear to be less significant when applied to resources with just a few shots, and since the test sequences used in this study are not fully representative of the Netflix catalogue, it is not proven that the implemented Dynamic Optimiser is able to reach the optimisation strength claimed in the starting paper, and therefore its figures can be only partially confirmed.

Despite this limitation, the study does prove the efficacy of the Dynamic Optimizer in more practical terms. The primary objective of this research was to implement a system on the basis of the functional and operational principles described by Netflix. From an implementation point of view, the study provides several key outcomes based on the three methods proposed and analysed.

- BF is the best performing method in terms of optimisation, as it returns the best quality and rate combinations, but it cannot be used as it is because it is not feasible from a software implementation point of view. It can only be used as a reference and moreover only in limited cases.
- LG is almost as performant as BF when the number of elemental encodes is enough or when they are dense enough around the target value. The initial range is crucial for its performance. LG is the most versatile and flexible method, because it is able to reach a suboptimal solution very close to the optimal one with a great reduction in the computational complexity.
- CF provides a good approximation of the solution, but it is the least efficient method in terms of optimisation. It becomes the best performer when the number of elemental encodes is low and encoding costs are limited. Further work in this direction is to be hoped because for the other methods, the encoding costs may become very high.

According to the results presented in this study in the last part of *Chapter 3*, several other factors than the optimization method play a crucial role in determining the success of the optimization process.

- Content is one of the most important factors, it affects the complexity of the encoding and the success of the optimisation. The use of VMAF and CRF enables us to have a great control over the complexity in different content situations, allowing the system to adapt to rapid changes. The LG method is particularly good in adapting to these changes.
- While PSNR can be used for optimization, the results suggest that its performance may not be as good as that of VMAF. On the other hand, VMAF aggregated and mean values should also consider the discontinuity caused by the difficulty of computing VMAF across shots boundaries, as the VMAF for a frame depends on features extracted from the previous one.
- AVC, being the oldest coding standard, has the lowest performances, but it is still the most widely used and supported. Its speed also makes it a popular choice. It is better suited for natural content, in particular at high bitrates, but it should be avoided in case of synthetic contents.
- In contrast, HEVC is better suited for computer graphic generated video sources while VP9 on natural content.
- Our findings show that AV1 is the best encoder for dynamic optimization across the whole range of bitrates and complexities. However, it is preferred not to use it for low resolutions, such as SD or lower, where AVC is more appropriate. Instead, for FullHD and higher resolutions, it is suitable.

CONCLUSIONS

Numerous studies have been published in the research literature describing and evaluating the Dynamic Optimizer framework. However no freely available software implementations are available until now. The study detailed in this work aims at identifying the most effective video encoder and optimization strategy for achieving the best between quality and bitrate. For this reason, we have developed and released the first open-source software implementation of a video Dynamic Optimizer, which includes three distinct optimization techniques. The Dynamic Optimizer uses Constant Rate Factor as the parameter to establish the compression level of each shot. That single parameter determines the trade-off between quality and bitrate and allows for efficient perceptual optimization using VMAF. The VMAF perceptive video quality metric is fundamental within the Dynamic Optimizer, first of all because of its accuracy in comparison with traditional metrics like PSNR, but especially because it takes into account the human visual system, allowing for visual perceptual optimization of any video codec. The three implemented methods, taking advantage of these two advanced tools in perceptual video processing, are able to fulfil the the role of a dynamic optimizer, offering different solutions for specific use cases and granting a gain, both in terms of quality or bitrate reduction, up to 10% against traditional video sequences encoded without any encoder-specific optimisation technique.

The software is suitable for further optimization and improvements for two main reasons. The first one is to try to reach optimization performances of the first introduced Dynamic Optimizer, the one used by Netflix to provide its streaming services. The second one is because, if improved, the algorithm can become an attractive alternative to traditional video encoding techniques and eventually be included into next-generation encoding schemes.

The implemented version described in this work shows promising results in achieving high-quality video at the lowest possible bitrate. However, further work is needed to improve its effectiveness and expand its applications. One area of improvement is the shot change algorithm, which can enhance the accuracy of content analysis and optimization. Another improvement would be to test the system using other rate control methods, such as the QP parameter, to determine the most effective control of the rate for different types of content and use scenarios. In this sense, including greater control over parameters within single

shots through the GOP structure, for example fine-tuning the number of I-B-P frames, can potentially improve optimization results. Moreover, even though the curve fitting algorithm is effective, alternative techniques should also be explored to improve the accuracy of bitrate predictions. To improve quality assessment instead, and solve the problem of the wrong VMAF score at the beginning of a new shot, we should try to use VMAF with real *integer_motion* information, avoiding to encode each shot separately. Both encoding and assessment can be performed on the entire sequence, working with pointers to the timecode of shots' edges to gather RD data from each segment, and potentially saving encoding time and resources.

It is important to acknowledge that our results may not be considered valid for all types of video content, and further research is needed to explore the performance of the Dynamic Optimizer and other encoders in different scenarios. Releasing an open-source version of the Dynamic Optimizer will allow for further development and improvements by the wider community for always enhanced video quality and streaming services. With the continued rise in demand for video streaming services and the emergence of new formats and standards, it is important to explore alternative techniques for video encoding optimization. As demonstrated in this study, dynamic optimization has shown great promise in achieving higher quality at the lowest possible bitrate, like the latest coding algorithms. Continued exploration and development of dynamic optimization techniques can potentially play a major role in advancing the fields of video coding and compression.

Code Availability. The software implementation developed for this paper is available on GitHub at <https://github.com/CheminDavide/Dynopt>.

BIBLIOGRAPHY

Abdulhussain, S. H., Ramli, A. R., Saripan, M. I., Mahmmod, B. M., Al-Haddad, S. A. R., and Jassim, W. A. (2018). Methods and Challenges in Shot Boundary Detection: A Review. *Entropy*, 20(4), 1-42.

Akhshabi, S., Begen, A. C., and Dovrolis, C. (2011, February 23-25). *An Experimental Evaluation of Rate Adaptation Algorithms in Adaptive Streaming over HTTP*. Proceedings of the Second Annual ACM Conference on Multimedia Systems, San Jose, California, USA.

Allam, R., and Dinana, H. (2021). The Future of TV and Online Video Platforms: A Study on Predictors of Use and Interaction with Content in the Egyptian Evolving Telecomm, Media & Entertainment Industries. *SAGE Open*, 11(3).

Antsiferova, A., Yakovenko, A., Safonov, N., Kulikov, D.L., Gushin, A., and Vatolin, D.S. (2021, September 27-30). *Objective video quality metrics application to video codecs comparisons: choosing the best for subjective quality estimation*. 31th International Conference on Computer Graphics and Vision, Nizhny Novgorod, Russia.

Barany, F. (2007). Signal Bandwidth vs. Resolution for Analog Video. *Analog Devices*.

Batsi, S., and Kondi, L. P. (2020). Improved temporal pooling for perceptual video quality assessment using VMAF. *Electronic Imaging*, 11, 68-1.

Battista, S., Meardi, G., Ferrara, S., Ciccarelli, L., Maurer, F., Conti, M., and Orcioni, S. (2022). Overview of the Low Complexity Enhancement Video Coding (LCEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), 560-576.

Bing, B. (2015). *Next-generation video coding and streaming*. John Wiley and Sons.

Bitmoving. (2022, December). *The 6th Annual Bitmoving Video Developer Report. Shaping the future of video 2022/2023*. Retrieved February 18, 2023 from

<https://bitmovin.com/wp-content/uploads/2022/12/bitmovin-6th-video-developer-report-2022-2023.pdf>

Blau, Y., and Michaeli, T. (2019, June 9-15). *Rethinking Lossy Compression: The Rate-Distortion-Perception Tradeoff*. Proceedings of the 36 th International Conference on Machine Learning (ICML), Long Beach, California, USA.

Bross, B., Wang, Y. K., Ye, Y., Liu, S., Chen, J., Sullivan, G. J., Ohm, J. R. (2021). Overview of the Versatile Video Coding (VVC) Standard and its Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10), 3736-3764.

Chikkerur, S., Sundaram, V., Reisslein, M., and Karam, L. J. (2011). Objective Video Quality Assessment Methods: A Classification, Review, and Performance Comparison. *IEEE Transactions on Broadcasting*, 57(2), 165-182.

Choi, K., Chen, J., Rusanovskyy, D., Choi, K. P., and Jang, E. S. (2020). An Overview of the MPEG-5 Essential Video Coding Standard. *IEEE Signal Processing Magazine*, 37(3), 160-167.

Cisco. (2020). *Cisco Annual Internet Report (2018–2023) White Paper*. Retrieved December 17, 2022, from <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf>

Cover, T. M., and Thomas, J.A. (2012). *Elements of Information Theory*. John Wiley & Sons.

Datareportal. (2023, January 26). *Digital 2023 Global Overview Report*. Retrieved February 18, 2023, from <https://datareportal.com/reports/digital-2023-global-overview-report>

De Cock, J., Li, Z., Manohara, M., and Aaron, A. (2016, September 25-28). *Complexity-based consistent-quality encoding in the cloud*. 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, Arizona, USA.

Deloitte. (2018). The future of the TV and video landscape by 2030. Retrieved February 18, 2023, from https://www2.deloitte.com/content/dam/Deloitte/be/Documents/technology-media-telecommunications/201809%20Future%20of%20Video_DIGITAL_FINAL.pdf

Dror, G. (2017). How to Encode Video for the Future. *Beamr*. Retrieved March 10, 2023, from http://beamrvideomedia.s3.amazonaws.com/pdf/Beamr_How_to_Encode_Video_for_the_Future.pdf

FFmpeg. (n.d.). *AV1 Video Encoding Guide*. Retrieved March 15, 2023, from <https://trac.ffmpeg.org/wiki/Encode/AV1>

FFmpeg. (n.d.). *FFmpeg and VP9 Encoding Guide*. Retrieved March 15, 2023, from <https://trac.ffmpeg.org/wiki/Encode/VP9>

FFmpeg. (n.d.). *FFmpeg Filters Documentation*. Retrieved March 15, 2023, from <https://ffmpeg.org/ffmpeg-filters.html>

FFmpeg. (n.d.). *FFmpeg Utilities Documentation*. Retrieved March 15, 2023, from <https://ffmpeg.org/ffmpeg-utils.html>

FFmpeg. (n.d.). *General Documentation*. Retrieved March 15, 2023, from <https://www.ffmpeg.org/general.html>

FFmpeg. (n.d.). *H.264 Video Encoding Guide*. Retrieved March 15, 2023, from <https://trac.ffmpeg.org/wiki/Encode/H.264>

FFmpeg. (n.d.). *H.265/HEVC Video Encoding Guide*. Retrieved March 15, 2023, from <https://trac.ffmpeg.org/wiki/Encode/H.265>

GlobalWebIndex. (2019). *Traditional vs. Digital Media: Global Trends*. GWI Trend Report 2019. Retrieved February 19, 2023, from https://www.amic.media/media/files/file_352_2142.pdf

Guo, L., Valliammal, A. K. G., Tam, R., Pham, C., Opalach, A., and Ni, W. (2021, November 9). Bringing AV1 Streaming to Netflix Members' TVs. *Netflix Technology Blog*. Retrieved April 15, 2022, from <https://netflixtechblog.com/bringing-av1-streaming-to-netflix-members-tvs-b7fc88e42320>

Gushchin, A., Antsiferova, A., and Vatolin, D. (2021, September 27-30). *Shot Boundary Detection Method Based on a New Extensive Dataset and Mixed Features*. 31th International Conference on Computer Graphics and Vision, Nizhny Novgorod, Russia.

Gådin, D., Hermanson, F., Marhold, A., Sikström, J., and Winman, J. (2022). Making Video Streaming More Efficient Using Per-Shot Encoding (Dissertation). Retrieved March 10, 2023, from

<http://www.diva-portal.org/smash/get/diva2:1665985/FULLTEXT01.pdf>

Han, J., Li, B., Mukherjee, D., Chiang, C.-H., Grange, A., Chen, C., Su, H., Parker, S., Deng, S., Joshi, U., Chen, Y., Wang, Y., Wilkins, P., Xu, Y. (2021). A technical overview of AV1. *Proceedings of the IEEE*, 109(9), 1435-1462.

Howard, P. G., and Vitter, J. S. (1994). Arithmetic coding for data compression. *Proceedings of the IEEE*, 82(6), 857-865.

Hwang, J. (2009). *Multimedia Networking: From Theory to Practice*. Cambridge University Press.

Ignatoski, M., Lerga, J., Stanković, L., and Daković, M. (2020). Comparison of Entropy and Dictionary Based Text Compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian. *Mathematics*, 8(7), 1-14.

ITU. (1994). Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video. *ITU-T Recommendation H.262 and ISO/IEC 13 818-2 (MPEG-2)*.

ITU (2016), Handbook on Digital Terrestrial Television Broadcasting Networks and Systems Implementation. *ITU Handbooks on Radiocommunications*.

ITU-T. (2008). Subjective video quality assessment methods for multimedia applications. *International Telecommunication Union, Recommendation P.910*.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Prentice Hall.

Javadtalab, A., Omidyeganeh, M., Shirmohammadi, S., and Hosseini, M. (2011, July 11-15). *A Rate Control Algorithm for x264 High Definition Video Conferencing*. Proceedings IEEE International Conference on Multimedia and Expo, Barcelona, Spain.

Katsavounidis, I. (2018, March 5). Dynamic optimizer - a perceptual video encoding optimization. *Netflix Technology Blog*. Retrieved April 15, 2022, from <https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f>

Katsavounidis, I., and Guo, L. (2018, August 19-23). *Video codec comparison using the dynamic optimizer framework*. Proceedings SPIE 10752. Applications of Digital Image Processing XLI, San Diego, California, USA.

Katsenou, A. V., Sole, J., and Bull, D. R. (2021). Efficient bitrate ladder construction for content-optimized adaptive video streaming. *IEEE Open Journal of Signal Processing*, 2, 496-511.

Keith, J. (2005) *Video Demystified* (4th edition). Elsevier.

Li, Z., Aaron, A., Katsavounidis, I., Moorthy, A., and Manohara, M. (2016, June 6). Toward a practical perceptual video quality metric. *Netflix Technology Blog*. Retrieved January 19, 2022, from <https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652>

Manohara, M., Moorthy, A., De Cock, J., Katsavounidis, I., and Aaron, A. (2018, March 9). Optimized shot-based encodes: Now streaming. *Netflix Technology Blog*. Retrieved January 19, 2022, from <https://netflixtechblog.com/optimized-shot-based-encodes-now-streaming-4b9464204830>

Mansri, I., Doghmane, N., Kouadria, N., Harize, S., and Bekhouch, A. (2020, October 19-22). *Comparative Evaluation of VVC, HEVC, H.264, AV1, and VP9 Encoders for Low-Delay Video Applications*. 2020 Fourth International Conference on Multimedia Computing, Networking and Applications (MCNA), Valencia, Spain.

Menon, V. V., Amirpour, H., Ghanbari, M., and Timmerer, C. (2022, May 23-27). *OPTE: Online Per-Title Encoding for Live Video Streaming*. ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, Singapore.

Mukherjee, D., Bankoski, J., Grange, A., Han, J., Koleszar, J., Wilkins, P., Xu, Y., and Bultje, R. (2013). The latest open-source video codec vp9 - An overview and preliminary results. *Picture Coding Symposium (PCS)*, 390-393.

Nebeker, N. (1998) Fifty Years of Signal Processing: The IEEE Signal Processing Society and Its Technologies 1948-1998. *The IEEE Signal Processing Society*, 20.

Netflix. (n.d.). *Using VMAF with FFmpeg*. Retrieved March 15, 2023, from <https://github.com/Netflix/vmaf/blob/master/resource/doc/ffmpeg.md>

Oliver, B. M., Pierce, J. R., and Shannon, C.E. (1948). The Philosophy of PCM. *Proceedings of the IRE*, 36(11), 1324-1331.

Oppenheim, A. V., and Schaffer, R. W. (2010). *Discrete-time signal processing* (3rd ed.). Pearson.

Ortega, A., and Ramchandran, K. (1998). Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine*, 15(6), 23-50.

Owen, F. F. E. (1982). *PCM and digital transmission systems*. McGraw-Hill.

Pratheek., R., and Suma, M.N. (2013). Performance Analysis of DPCM and ADPCM. *IJCA Special Issue on International Conference on Electronic Design and Signal Processing ICEDSP*, 3, 19-23.

Punchihewa, A., and Bailey, D. (2020). *A Review of Emerging Video Codecs: Challenges and Opportunities*. 35th International Conference on Image and Vision Computing New Zealand (IVCNZ), Wellington, New Zealand.

Robitza, W. (2017). *CRF Guide (Constant Rate Factor in x264, x265 and libvpx)*. Retrieved March 15, 2023, from <https://slhck.info/video/2017/02/24/crf-guide.html>

Robitza, W., Rao Ramachandra Rao, R., Göring, S., and Raake, A. (2021). *Impact of spatial and temporal information on video quality and compressibility*. 2021 13th International Conference on Quality of Multimedia Experience (QoMEX), Montreal, QC, Canada.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3), 379-423.

Sikora, T. (1997). MPEG digital video-coding standards. *IEEE Signal Processing Magazine*, 14(5), 82-100.

Simon, H., and Van Veen, B. (2003). *Signals and Systems* (2nd ed.). Prentice Hall.

Sullivan, G. J., Ohm, J. R., Han, W. J., and Wiegand, T. (2012). Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12), 1649-1668.

Tanenbaum, A. S., and Wetherall, D. (2013). *Computer Networks*. Pearson Prentice Hall.

Tanyer, S. G. (2012, April 18-20). *The Cumulative Distribution Function for a finite data set*. 2012 20th Signal Processing and Communications Applications Conference (SIU), Mugla, Turkey.

Tekalp, A. M. (1995). *Digital Video Processing*. Prentice Hall.

TV key facts. (2022, November). *Total Video International Trends*. RTL AdAlliance.

Vassiliadis, S., Hakkennes, E. A., Wong, J. S. S. M., and Pechanek, G. G. (1998, August 27). *The sum-absolute-difference motion estimation accelerator*. Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204), Vasteras, Sweden.

Wallace, G. K. (1992). The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1), 28-34.

Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600–612.

Wedi, T. (2003). Motion compensation in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13, 577–586.

Wiegand, T., and Schwarz, H. (2011). Source coding: Part I of fundamentals of source and video coding. *Foundations and Trends in Signal Processing*, 4(1–2), 1-222.

Wiegand, T., Sullivan, G. J., Bjontegaard, G., and 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.

Wien, M. (2015). *Video Coding Fundamentals*. In High Efficiency Video Coding. Signals and Communication Technology. Springer.

Yeung, R.W. (2008). *Information theory and network coding*. Springer.

Young, I. T., Gerbrands, J. J., and Van Vliet, L. J. (1998). *Fundamentals of image processing*. In Madisetti, V. K. and Williams, D. B. (Eds.), *The Digital Signal Processing Handbook*. CRC Press.

Zeng, H., Xu, J., He, S., Deng, Z., and Shi, C. (2022). Rate Control Technology for Next Generation Video Coding Overview and Future Perspective. *Electronics*, 11(23), 1-22.

Politecnico di Torino

Master degree program

in Cinema and Media Engineering

AY 2022-2023

March-April graduation session 2023

Master Thesis

The Dynamic Optimizer Framework

Video encoding, assessment and comparison

Author Chemin Davide

Supervisor Masala Enrico



**Politecnico
di Torino**