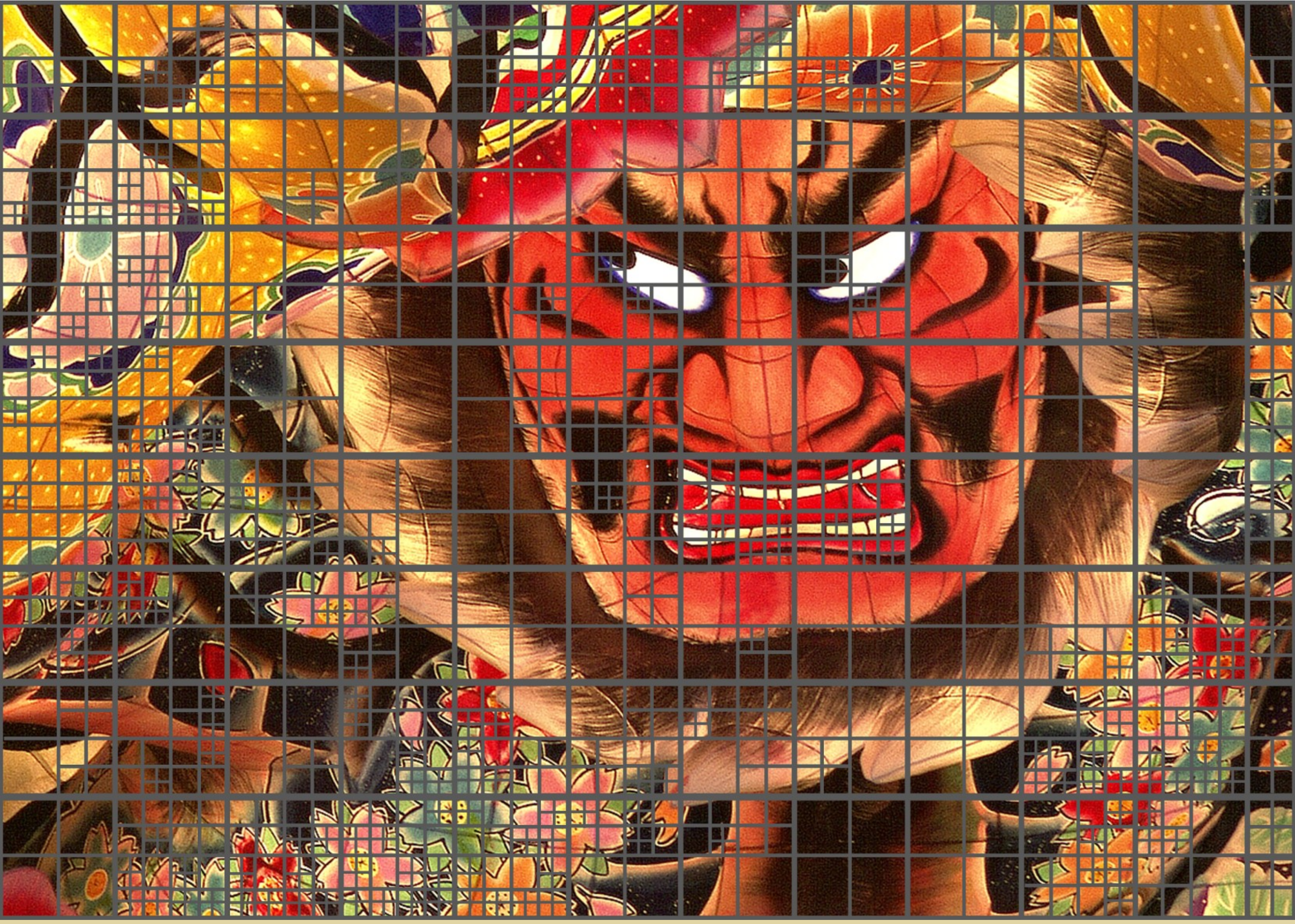


Guilherme Ribeiro Corrêa

# COMPUTATIONAL COMPLEXITY REDUCTION AND SCALING FOR HIGH EFFICIENCY VIDEO ENCODERS

Tese de Doutoramento na área científica de Engenharia Eletrotécnica e de Computadores, especialidade Telecomunicações, orientada pelo Professor Doutor Luís Alberto da Silva Cruz e apresentada ao Departamento de Engenharia Eletrotécnica de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Agosto 2014

· U C ·

UNIVERSIDADE DE COIMBRA

**Departamento de Engenharia Eletrotécnica e de Computadores**

**Faculdade de Ciências e Tecnologia**

**Universidade de Coimbra**

# Computational Complexity Reduction and Scaling for High Efficiency Video Encoders

GUILHERME RIBEIRO CORRÊA

Coimbra

August 2014

# Computational Complexity Reduction and Scaling for High Efficiency Video Encoders

by

**Guilherme Ribeiro Corrêa**

under the supervision of

Prof. Dr. Luis Alberto da Silva Cruz
(adviser)

Prof. Dr. Pedro António Amado Assunção
(co-adviser)

Prof. Dr. Luciano Volcan Agostini
(co-adviser)

A thesis submitted to the

Departamento de Engenharia Eletrotécnica e de Computadores

Faculdade de Ciências e Tecnologia

Universidade de Coimbra

in partial fulfilment of the requirements for the degree of

**Doctor of Philosophy**.

August 2014

*The Road goes ever on and on*
*Down from the door where it began.*
*Now far ahead the Road has gone,*
*And I must follow, if I can,*
*Pursuing it with eager feet,*
*Until it joins some larger way*
*Where many paths and errands meet.*
*And whither then? I cannot say.*

J.R.R. Tolkien

*To my parents,*
*for their endless*
*love and support.*

# Abstract

In the last decades, the rapid advances of semiconductor technologies fostered a large development in the field of multimedia systems, mainly due to the continuous increase of computational resources and the availability of reliable communication infrastructures. Several video compression standards have been developed in this period, aiming at reducing transmission bit rates without decreasing the video quality. The High Efficiency Video Coding (HEVC) standard, recently launched by the Joint Collaborative Team on Video Coding (JCT-VC), is the state of the art in video compression and is expected to gradually substitute its predecessor, the H.264/AVC standard. HEVC provides improved compression ratios in comparison to previous standards, but such gains are associated with large increases in the encoding computational complexity and consequently longer processing times, which may compromise the encoder operation in portable devices and in real-time systems, especially for high-resolution videos. The research work presented in this thesis addresses the subject of computational complexity of HEVC encoders with contributions extending from the analysis of HEVC compression efficiency and computational complexity to the reduction and scaling of its encoding complexity. The first contribution of this thesis is an investigation and detailed analysis of the HEVC encoding tools which allowed identifying the most computationally demanding operations of the encoding process. The second contribution of this thesis comprises a set of five new algorithms for reducing and scaling the encoding complexity of HEVC encoders. All of them take advantage from the flexibility of the frame partitioning structures allowed by the standard, namely the Coding Units and the Prediction Units, which were identified as responsible for a large share of the encoding computational complexity. The best complexity scaling algorithm proposed in this work allows downscaling the encoding complexity to 50% of its original value with negligible loss of compression efficiency and down to 20% with medium to small loss. The third thesis contribution consists of a set of early termination methods based on data mining techniques, which are able to reduce the computational complexity required to find the best frame partitioning structures, namely the Coding Trees, the Prediction Units and the Residual Quadtrees, in up to 65% with very small compression efficiency loss. Finally, the fourth contribution of this thesis is an encoding time control system that employs the three previous contributions of this research to adjust the encoding time whenever necessary and maintain it under a specified target. The system uses pre-defined encoding configurations created by combining the early termination schemes and by changing the parameterisation of the most computationally demanding tools of HEVC. Overall, the methods proposed in this thesis are especially useful in power-constrained portable multimedia devices to reduce energy consumption and to extend the battery life. Besides, they can also be applied to portable and non-portable multimedia devices operating in real time with limited computational resources.

# Resumo

Nas últimas décadas, os avanços na indústria de semicondutores possibilitaram um grande desenvolvimento no campo de sistemas multimédia, principalmente devido ao contínuo aumento de poder computacional e à disponibilidade de infraestruturas de comunicação confiáveis. Diversos padrões de compressão de vídeo foram desenvolvidos neste período com o objetivo de reduzir as taxas de bits sem afetar a qualidade do vídeo codificado. O padrão High Efficiency Video Coding (HEVC), recentemente lançado pelo Joint Collaborative Team on Video Coding (JCT-VC), tornou-se o estado-da-arte em compressão de vídeo e deve gradualmente substituir o seu predecessor, o H.264/AVC, dentro de poucos anos. O HEVC provê elevados níveis de compressão em comparação com outros padrões, mas tais ganhos são associados a grandes aumentos na complexidade computacional e, consequentemente, no tempo da codificação, prejudicando ou até mesmo impedindo a operação do codificador em dispositivos portáteis e em sistemas de tempo real, especialmente para vídeos de alta resolução. O foco desta tese concentra-se na complexidade computacional de codificadores HEVC, com contribuições que se estendem desde a análise da eficiência de compressão e da complexidade computacional do padrão até a redução e o ajuste dinâmico da sua complexidade de codificação. A primeira contribuição desta tese é uma análise detalhada das funcionalidades e ferramentas de codificação que compõem o HEVC, a qual foi realizada com vistas à identificação das operações mais complexas do processo de codificação. Cinco algoritmos para escalonamento dinâmico da complexidade de codificação representam a segunda contribuição da tese. Todos eles se baseiam no ajuste das novas estruturas de particionamento de trama introduzidas pelo novo padrão, nomeadamente as Unidades de Codificação e as Unidades de Predição, as quais foram identificadas como responsáveis por uma grande parcela da complexidade computacional do codificador HEVC. O melhor algoritmo de escalonamento desenvolvido provê reduções de até 50% na complexidade computacional com perdas negligenciáveis na eficiência da compressão e reduções de até 80% com perdas pequenas ou médias. A terceira contribuição desta tese consiste em um conjunto de esquemas de finalização antecipada baseados em técnicas de mineração de dados, os quais procuram reduzir a complexidade computacional demandada pelos processos de decisão das melhores estruturas de particionamento de trama, nomeadamente as Árvores de Codificação, as Unidades de Predição e as Árvores Residuais Quadráticas. Embora os esquemas não permitam escalonamento dinâmico, reduções de até 65% na complexidade computacional foram obtidas com perdas muito pequenas na eficiência de compressão. Finalmente, a quarta contribuição da tese consiste em um sistema de controlo que utiliza as três contribuições anteriores com a finalidade de ajustar o tempo de codificação sempre que necessário, com o objetivo de mantê-lo abaixo de um determinado alvo. O sistema de controlo utiliza configurações de codificação pré-definidas, as quais foram criadas a partir de combinações dos esquemas de finalização antecipada e de modificações na parametrização das ferramentas com maior complexidade computacional no codificador HEVC. Sobretudo, os métodos propostos nesta tese são especialmente úteis em dispositivos multimédia portáteis com limitação energética, possibilitando a redução do consumo de energia elétrica e um consequente prolongamento da duração das suas baterias. Além disso, os métodos também podem ser aplicados a dispositivos multimédia portáteis e não-portáteis que operam em tempo real com recursos computacionais limitados.

**Key words:**

High Efficiency Video Coding (HEVC), partitioning structures, coding mode decision, decision trees, computational complexity, complexity reduction, complexity scaling.

**Palavras-chave:**

High Efficiency Video Coding (HEVC), estruturas de particionamento, decisão de modo de codificação, árvores de decisão, complexidade computacional, redução de complexidade, escalonamento de complexidade.

# Acknowledgements

This research would not be possible without the aid of my adviser, Prof. Luis Alberto da Silva Cruz, to whom I am extremely grateful. Prof. Luis has guided every step I took during my PhD and provided me with the best working conditions at the University of Coimbra. I would like to thank him for being a very dedicated mentor, for being always present and available to discuss new ideas whenever I needed and for finding the economic means to support my stay in Portugal during my first year.

I must also thank Prof. Pedro António Amado Assunção, my co-adviser in Portugal, for all the discussions about this research and for all the meticulous reviews on my papers, articles and this thesis. I thank Prof. Luciano Volcan Agostini, my co-adviser in Brazil, for starting and maintaining the inter-institutional cooperation between the Federal University of Pelotas and the Institute for Telecommunications, which enabled my coming to Portugal.

I would like to thank my partner, Gonçalo, for being by my side during these years in Portugal, for being patient when I had to work until late or in the weekends and for all the love and support.

I thank my parents, Élida and Vitório, for their life lessons, love and economic help whenever I needed. I am sorry for being so far and I hope one day I can reward all this time away from them.

Finally yet importantly, I thank the Brazilian and the Portuguese people who pay many taxes every day and thus provide the government with funds to foment scientific investigation, including the research presented in this thesis, which was supported by the Brazilian National Council for Scientific and Technological Development (CNPq) and Portuguese Foundation for Science and Technology (FCT).

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AI | All Intra |
| ALF | Adaptive Loop Filter |
| AMP | Asymmetric Motion Partition |
| ARFF | Attribute-Relation File Format |
| AVC | Advanced Video Coding |
| BD | Bjøntegaard Delta |
| BD-rate | Bjøntegaard Delta bit rate |
| BD-PSNR | Bjøntegaard Delta PSNR |
| CABAC | Context-Adaptive Binary Arithmetic Coding |
| CB | Coding Block |
| CBF | Coded Block Flag |
| CCUPU | Constrained Coding Units and Prediction Units |
| CIF | Common Intermediate Format |
| CTB | Coding Tree Block |
| CTC | Common Test Conditions |
| CTDE | Coding Tree Depth Estimation |
| CTU | Coding Tree Unit |
| CU | Coding Unit |
| DBF | Deblocking Filter |
| DCT | Discrete Cosine Transform |
| DMV | Differential Motion Vector |
| DPB | Decoded Picture Buffer |
| DST | Discrete Sine Transform |
| EPZS | Enhanced Predictive Zonal Search |
| Fc | Constrained Frame |
| FDCR | Fixed Depth Complexity Reduction |
| FME | Fast Motion Estimation |
| fps | frames per second |
| FRME | Fractional Motion Estimation |

| | |
|---|---|
| FS | Full Search |
| Fu | Unconstrained Frame |
| GBFOS | Generalised Breiman, Friedman, Olshen and Stone Algorithm |
| GOP | Group of Pictures |
| GPB | Generalised P and B-picture |
| HD | High Definition |
| HE | High Efficiency |
| HETR | High-Efficiency Encoding Time Reduction |
| HEVC | High Efficiency Video Coding |
| HM | HEVC Model |
| HP | High Profile |
| I/O | Input/Output |
| IBD | Internal Bit Depth |
| IBDD | Internal Bit Depth Decrease |
| IBDI | Internal Bit Depth Increase |
| IDR | Instantaneous Decoding Refresh |
| IGAE | Information Gain Attribute Evaluation |
| IEC | International Electrotechnical Commission |
| IME | Integer Motion Estimation |
| IP | Intra Prediction |
| IQ | Inverse Quantisation |
| ISO | International Organisation for Standardisation |
| IT | Inverse Transform |
| JCT-VC | Joint Collaborative Team on Video Coding |
| KDD | Knowledge Discovery from Data |
| KLD | Kullback-Leibler Divergence |
| LC | Low Complexity |
| LCB | Largest Coding Block |
| LCTC | Low-Complexity Encoding Time Control |
| LD | Low Delay |
| LDP | Low Delay P |
| LM | Linear Mode |
| MB | Macroblock |
| MC | Motion Compensation |
| MCTDL | Motion-Compensated Tree Depth Limitation |
| MD | Mode Decision |
| ME | Motion Estimation |
| MPEG | Moving Picture Experts Group |
| MPM | Most Probable Modes |
| MSE | Mean-Squared Error |
| MSM | Merge/SKIP Mode |
| MTDM | Maximum Tree Depth Map |
| MV | Motion Vector |
| MVP | Motion Vector Prediction |

| | |
|---|---|
| NSQT | Non-Square Transforms |
| PB | Prediction Block |
| PRD | Power-Rate-Distortion |
| PRDO | Power-Rate-Distortion Optimisation |
| PSNR | Peak Signal-to-Noise Ratio |
| PU | Prediction Unit |
| Q | Quantisation |
| QP | Quantisation Parameter |
| QZB | Quasi-zero-blocks |
| RA | Random Access |
| RAP | Random Access Point |
| R-D | Rate-Distortion |
| R-D-C | Rate-Distortion-Complexity |
| RDCO | Rate-Distortion-Complexity Optimisation |
| RDO | Rate-Distortion Optimisation |
| RDOQ | Rate-Distortion Optimised Quantisation |
| RGB | Red, Green and Blue |
| RMD | Rough Mode Decision |
| RQT | Residual Quadtree |
| SA | Search Area |
| SAD | Sum of Absolute Differences |
| SAO | Sample Adaptive Offset |
| SATD | Sum of Absolute Transformed Differences |
| SCB | Smallest Coding Block |
| SMP | Symmetric Motion Partition |
| SSE | Sum of Squared Error |
| SUMHexS | Simple Unsymmetrical-Cross Multi-Hexagon-Grid Search |
| T | Transform |
| TB | Transform Block |
| TSS | Three-Step Search |
| TU | Transform Unit |
| TZS | Test Zone Search |
| UMHexS | Unsymmetrical-Cross Multi-Hexagon-Grid Search |
| VCEG | Video Coding Experts Group |
| VDCR | Variable Depth Complexity Reduction |
| VGA | Video Graphics Array |
| YCbCr | Luminance, Blue Chrominance and Red Chrominance |
| Y-PSNR | Luminance PSNR |

Chapter 1

# Introduction

In the last decades, the rapid advances of semiconductor technologies fostered a large expansion in the consumer market of multimedia-ready devices due to the continuous increase of computational resources and availability of reliable communication infrastructures. Digital video has been available since the beginning of the so-called Information Age, but only after the expansion and popularisation of personal computers and high-speed networks this type of information became so widely present in our daily lives.

Nowadays, digital televisions, portable computers, personal digital assistants (PDA) and even mobile phones are among the most popular consumer equipment able to receive and display high-resolution video in real time. Very common are also those devices that can capture and transmit digital video through wired and wireless channels. Furthermore, the current trend in most portable devices with embedded digital cameras is to include the capability of encoding and decoding high-resolution digital video streams.

Despite the recent evolution in portable devices, particularly in terms of communications technology and computational power, the limited battery capacity still imposes major constraints in multimedia applications demanding high computational power, such as those dealing with video encoding. In such cases, the user experience might be limited by the reduced battery capacity. Furthermore, even in those cases in which battery capacity is not an issue, encoding and decoding high-resolution digital video streams in real time is still a challenge, especially when considering the computational requirements of the most recent video coding standards.

Previous studies which examine typical use scenarios of portable devices have shown that a very significant amount of power consumption (from 40% to 60%) is related to video encoding and decoding operations, with the encoder typically requiring the largest share of computing time and power consumption [1-4]. It has been claimed in [5] that more than two-thirds of this computational complexity corresponds to the encoding process, whereas the rest is divided between transmission and input/output (I/O) operations. Even though the results presented in [5] are just an estimate for low-resolution video, the recent adoption of higher resolutions increased even more the computational needs of the video encoding processes, since greater computational efforts are necessary to process the increasingly larger amounts of video information. Furthermore, a consequence of current video coding standard evolution is that the use of more efficient signal processing tools is significantly augmenting the number of operations-per-pixel required in the newest video codecs.

This is the case of the state-of-the-art coding standard, the High Efficiency Video Coding (HEVC) [6], finished in March of 2013 by the Joint Collaborative Team on Video Coding (JCT-VC), from the International Telecommunication Union (ITU) [7] and the Joint Technical Committee 1 of the International Organisation for Standardisation and the International Electrotechnical Commission (ISO/IEC JTC1) [8]. HEVC achieves 40%-50% bit rate reduction in comparison with its predecessor, the H.264/AVC video coding standard [9], at the same subjective image quality. However, to reach such goal, HEVC incorporates several new tools, which increased the encoder computational complexity in a range from 9% to 502% in comparison to H.264/AVC High Profile [10].

Even though several works addressing complexity-aware video coding have been proposed in the last years and designed for use in previous video encoding standards, none of them can be directly applied to the particular case of HEVC, which is based on encoding structures and tools quite different from those of previous standards, as shown later in this text. At the beginning of the development of the research work described in this thesis, there had been no work published in the technical literature focusing on computational complexity analysis, reduction and scaling for HEVC. To the authors' knowledge, the first published work on computational complexity scaling for the HEVC standard was a product of this thesis' research [11].

In general terms, the novel contributions of this thesis include a performance and computational complexity assessment of HEVC and methods for reducing and scaling the encoding computational complexity while still maintaining a compromise with compression efficiency, allowing the encoder's use in both power- and computational-constrained applications.

## 1.1 Terminology for Computational Complexity

As complexity does not have a single universal meaning and is sometimes a confounding concept, its definition in the context of this thesis is provided in this section, as well as the related concepts of complexity reduction and scaling.

Computational complexity is a term used to describe the amount of calculations performed in a task. In the specific case of this research, the computational complexity refers to the calculations performed in the whole encoding process or in a specific part of it. As the number of computations affects directly the total time of processor usage, computational complexity is always measured in terms of processing time in the experiments presented in this thesis, unless explicitly stated otherwise. In order to simplify the text, the term complexity is sometimes used as a synonym of computational complexity throughout the thesis.

Computational complexity reduction is a term used to describe those methods that yield fixed decreases in the computational complexity of a task. Once applied to a determined algorithm, a computational complexity reduction technique is able to decrease the amount of computational resources required to complete the encoding process to a level that is dependent upon the video source characteristics. In this context, the amount of complexity reduction achieved is unknown until the completeness of the task.

Computational complexity scaling is the process of adaptively adjusting the computational complexity of a task in order to reach a desired target complexity, which can be defined by a user or a system (e.g., operating system or transmission equipment). Complexity scaling methods are able to decrease or increase the computational effort employed in video encoding by adjusting the encoder parameters on the fly until the computational complexity is below a given upper limit. A feedback mechanism is generally used in such cases to guide the adjustment steps that are applied to reach the target complexity.

## 1.2 Research Summary

In this section, the main contributions of this research are listed and a summary of the achievements related to each contribution is presented. Chapters 4, 5, 6 and 7 present detailed discussions on each topic and their respective results.

### 1.2.1 Encoding Performance and Complexity Assessment

When developing complexity-aware video coding systems, one should focus on those tools or processes that are the most computationally intensive and look for alternatives or adaptations that yield good encoding performance at the cost of a smaller computational complexity. As HEVC is a recent standard, there were no works related to its complexity analysis when this research started, so that a computational complexity assessment was essential for its development.

An experimental investigation was carried out in order to identify the tools that most affect the encoding efficiency and computational complexity of the HEVC encoder. A set of encoding configurations was created to investigate the impact of each tool, varying the encoding parameters and comparing the results with a baseline encoder. The results of this study provided relevant information to implement complexity-constrained encoders by taking into account the trade-off between complexity and coding efficiency. This work was published in [10].

A second research study was carried out in order to identify the influence of the frame partitioning structures of HEVC in both computational complexity and coding efficiency. The results of this work showed that the nature of the partitioning structures used in HEVC leads to nested Rate-Distortion Optimisation (RDO) loops, which is the main contributing factor to the increased encoding computational complexity. Part of this work was published in [11] and served as support to the development of other solutions proposed in this thesis.

### 1.2.2 Algorithms for Computational Complexity Scaling

A set of algorithms were proposed for computational complexity scaling based on dynamic adjustment of frame partitioning structures, namely the Coding Tree Units

(CTUs), Coding Units (CUs) and the Prediction Units (PUs). All algorithms aim at adjusting the computational effort employed to decide the best frame partitioning structures according to a target computational complexity, which is limited by the amount of computational resources available in the encoder. Five approaches have been proposed:

- Fixed Depth Complexity Scaling (FDCS) [11];
- Variable Depth Complexity Scaling (VDCS) [12, 13];
- Motion-Compensated Tree Depth Limitation (MCTDL) [14];
- Coding Tree Depth Estimation (CTDE) [15, 16];
- Constrained Coding Units and Prediction Units (CCUPU) [17, 18].

### 1.2.3 Data Mining for Computational Complexity Reduction

A set of procedures were proposed for deciding whether the partition structure decision should be terminated early or run to the end of an exhaustive search for the best configuration. The proposed methods are all based on decision trees obtained through data mining (DM) techniques. By extracting intermediate data from the encoding process, three sets of decision trees were devised to avoid running the RDO algorithm to its full extent:

- Early termination for determining Coding Trees [19, 20];
- Early termination for determining Prediction Units [20-22];
- Early termination for determining Residual Quadtrees.

These methods were then jointly implemented in order to provide further complexity reductions to the encoding process [20]:

- Early termination for determining Coding Trees and Prediction Units;
- Early termination for determining Coding Trees and Residual Quadtrees;
- Early termination for determining Prediction Units and Residual Quadtrees;
- Early termination for determining Coding Trees, Prediction Units and Residual Quadtrees.

### 1.2.4 Complexity Reduction and Scaling Applied to Encoding Time Control

The CCUPU method mentioned in 1.2.2, the techniques for complexity reduction mentioned in 1.2.3 and the encoder configurations that most affect the computational complexity of HEVC, identified in the analysis mentioned in 1.2.1, were combined and applied to the development of a Rate-Distortion-Complexity (R-D-C) optimised system that provides encoding time control for HEVC encoders. By adjusting the encoder operating point according to the best performing configurations identified in an extensive R-D-C efficiency analysis, the system provides encoding time control of medium to fine granularity, aiming at maintaining the encoding time per GOP under a specified target.

## 1.3 Thesis Organisation

This thesis is organised as follows:

Chapter 2 presents an overview of video coding and decoding technology, including the basic concepts of digital video compression, the general operation of a video compression system, a detailed description of HEVC, and discussions on RDO and computational complexity of video encoding.

Chapter 3 presents an overview of the state-of-the-art research on computational complexity reduction and scaling techniques for video encoding systems. A description of current methods for modelling, reducing and scaling the expenditure of computational resources on video codecs is presented along with future trends on complexity management for video codecs implemented in power-constrained or computationally constrained devices.

Chapter 4 presents a performance evaluation study of coding efficiency versus computational complexity for the HEVC standard. Two experimental investigations are described: the first one consists in identifying the tools that most affect the encoding efficiency and computational complexity of HEVC, while the second one consists in analysing the impact of using different frame partitioning structure configurations in both encoding efficiency and complexity.

Chapter 5 presents a set of heuristic methods for computational complexity scaling of HEVC encoders based on dynamic adjustment of the frame partitioning structures used in each CTU. The methods rely on spatio-temporal correlation in order to decrease the number of encoding possibilities tested in the RDO process.

Chapter 6 presents a data mining-based approach which reduces the encoding computational complexity of HEVC by decreasing the number of partitioning structure possibilities tested for each CTU in the RDO process. The chapter describes how the obtained decision trees were trained, implemented and validated and finally presents experimental results.

Chapter 7 proposes an encoding time control system that makes use of the findings of chapter 4, 5 and 6. A detailed experimental analysis allowed identifying encoder operating points that yield high R-D-C efficiency, which were then used to dynamically adjust the HEVC encoder operation to meet a given target time per GOP.

Chapter 8 presents the conclusions of this thesis, summarising the major results, identifying some possible extensions of the project and pointing out future research directions in the area.

Appendix A lists the 16 peer-reviewed submissions resulting from the research presented in this thesis, which include one book chapter, four journal articles and 11 conference papers.

Appendix B presents the *Common Test Conditions* (CTC) document, which describes the experimental setup for most of the tests performed in this work and the video sequences that are used in the tests. Additionally, the first frame of each video sequence is presented.

Appendix C lists the decision trees obtained in the work presented in chapter 6, showing their graphic representations.

Appendix D presents an extensive table that describes all the encoder configurations tested in the R-D-C analysis of chapter 7 and their respective R-D-C efficiency results. A look-up table (LUT) used by the encoding time control algorithm proposed in chapter 7 is also presented in the appendix.

# Chapter 2

# Video Coding Background

This chapter provides background information about video coding theory and practice. Initially, the basic concepts of video coding and the basic model of a hybrid video compression system are presented in this chapter. A discussion on the Rate-Distortion Optimisation (RDO) method is then followed by a description of the High Efficiency Video Coding (HEVC) standard. Finally, the chapter presents an introduction to the topic of computational complexity of HEVC, which is the focus of this thesis.

## 2.1 Basic Concepts

A digital video is a sequence of digital images to be presented sequentially to the viewer at a temporal rate high enough to ensure a smooth transition-free visual perception. In general, all individual images of a video, known as frames or pictures, have the same horizontal and vertical dimensions measured in pixels. Pixel is the name given to the numerical value of the picture elements, which are organised in matrix form.

For the purpose of encoding, a frame is usually divided into several $M \times N$ blocks of pixels, where $M$ is the number of rows and $N$ is the number of columns. Each video coding standard defines a different block size or even a variable range for it. Usually, the modules that compose a video encoder operate using blocks of pixels as basic processing units.

The frame dimensions, also called as spatial resolution, can assume arbitrary values in different video signals, even though there are some pre-defined formats that are broadly used in industry, such as the Common Intermediate Format (CIF), the Video

Graphics Array (VGA), and the 1080p, with 352×288 pixels, 640×480 pixels and 1920×1088 pixels, respectively. The higher the spatial resolution, the larger is the number of pixels in it and, consequently, more detailed is the perception of the video content.

The temporal rate at which frames are presented also influences the perceived video quality. The higher the number of frames shown in a determined period, the smoother is the motion noticed and the transition from one frame to another. The number of frames per second is called the temporal resolution, sampling rate or frame rate. In general, the temporal resolution varies between 15 and 60 frames per second (fps) in typical video sequences, although higher frame rates are becoming common.

Other important aspects to be considered when encoding a video sequence are the adopted colour system and sub-sampling pattern. The most used colour systems are RGB (Red, Green and Blue) and YCbCr (Luminance, Blue Chrominance and Red Chrominance). Thousands of distinct colours can be perceived from the different combinations of the elements that compose these systems. In RGB, colours are formed through different combination of the R (red), G (green) and B (blue) primary components. However, as the human visual system is much more sensible to luminance than to colour information, the YCbCr system was created to take advantage of this characteristic by allowing a sub-sampling of the chrominance (colour) information [23].

The use of sub-sampling patterns is by itself a kind of video compression, since it allows simply discarding part of the video data without causing perceptible visual impacts. Several spatial sub-sampling configurations can be used, but the most common are the 4:2:2 and the 4:2:0. In 4:2:2, there are two blue chrominance (Cb) and two red chrominance (Cr) samples for each four luminance (Y) samples. In 4:2:0, there is only one Cb and one Cr sample for each four Y samples [24]. The 4:2:0 configuration was used in all experiments presented in this thesis.

## 2.2 Lossless and Lossy Compression

Image and video compression techniques are generally based on exploiting data redundancy and data irrelevance. Lossless compression algorithms aim at reducing redundancy in data, representing it with a smaller amount of bits without causing any loss of information. In lossless techniques for image and video compression, the compressed data can be completely recovered after the decompression process, so that the original and the reconstructed image or video are exactly the same. On the other hand, lossy compression techniques incur in some loss of information and the compressed data cannot be fully restored during the decompression process. Lossy algorithms for image and video compression usually exploit characteristics of the human visual system and remove data portions that are not relevant for receivers, retaining only information that can be perceived in such a way that the final, decoded image seems to have no or small difference to the original one for almost all observers.

Three basic types of redundancy can be exploited in lossless digital video compression: spatial, temporal and entropic. Spatial or intra-frame redundancy arises due to correlation between pixels in the same image. If correlation exists in the spatial domain (i.e., neighbouring pixels have similar values), redundancy can be reduced through intra-frame prediction, a technique present in most current image and video coding standards. This type of correlation is visible to the human eye, since the neighbouring pixels present similar values [25]. Temporal or inter-frame redundancy arises due to similarities between temporally adjacent frames. The values of a set of pixels in a determined region of a video might not change from one frame to another or might vary a little, as in the case of an image background (sky or wall, for example). In other cases, the same pixels might reappear in a frame displaced in relation to a previous one, as in the case of an object moving in a scene (a bird, for example). Efficient techniques for reducing inter-frame redundancy yield high levels of compression and for this reason all current video coding standards exploit such techniques [25]. Entropic redundancy is related to the occurrence frequency of the encoded symbols in a video. The higher the probability of a symbol occurring, the lower is the amount of information associated to it. Entropic redundancy is removed by the entropy coding module in most video coding standards [26].

Even though they provide no loss of information, redundancy reduction techniques achieve limited compression ratios. However, in many applications information loss is not exactly a problem, as long as the reconstructed data is still comprehensible by the receiver. In the specific case of digital images and video sequences, some loss of information can be tolerated by the viewer as long as it does not incur in annoying visual artefacts. In return for accepting such loss, much higher compression ratios can be achieved with lossy methods than with lossless methods.

Lossy compression is generally performed with the aid of a quantisation process, which aims at representing a large (sometimes infinite) number of possible distinct inputs as a much more limited number of codewords, as in a many-to-few mapping [25]. In image and video compression, this is achieved by exploiting characteristics of the human visual system. The human eye can perceive small differences in brightness over a relatively large area (low frequency), but cannot perceive very well such variations in small areas (high frequency) [25]. This way, the amount of information that represent high frequency components is said to be irrelevant and can be reduced in the quantisation process, which is done by simply dividing each component in the frequency domain by a constant and then rounding it to the nearest integer. As a result, many high frequency components are rounded to small integer numbers or to zero, which yields very large compression ratios. Notice, however, that the original values cannot be restored after being rounded to the nearest integer, which characterises information loss.

## 2.3 Hybrid Video Compression

Hybrid video compression is based on the following signal and data processing operations: (*i*) inter and intra-frame prediction, (*ii*) de-correlating transform, (*iii*) quantisation and (*iv*) entropy coding. The prediction step is usually followed by the transform and quantisation of prediction residues, which is then followed by the entropy coding.

Fig. 2.1 shows the basic components of a generic video encoder. The solid black arrows indicate the data flow at the encoder from the input video sequence to the generated output bitstream. In intra-frame prediction, each block is predicted from the pixels of reconstructed, previously encoded neighbour blocks according to a determined prediction mode. In inter-frame prediction, the block is predicted from pixels belonging to previously encoded frames.

Motion Compensation (MC) is used in inter-frame prediction to exploit the fact that in most video sequences the only difference between two adjacent frames results from camera or object motion. By using MC, the encoder is able to encode only the difference between two frames, discarding the redundant information between them. In order to better capture the scene motion occurring between adjacent frames, block-based Motion Estimation (ME) is usually employed to find in previously encoded frames the best matching blocks in comparison to those in the current frame. The location of the best match in the reference frame is defined by a $(x, y)$ coordinate, called a Motion Vector (MV), which describes the reference block offset relatively to the current block.

In both intra and inter-frame prediction, the predicted and the current block are rarely equal, so that the difference between them must be calculated, encoded and transmitted. This difference is called prediction residue or prediction error. If ME is used in inter-frame prediction, the corresponding MV has to be encoded and transmitted together with the residue (*Motion info*, in Fig. 2.1). In the case of intra-frame prediction, the mode used for prediction is encoded with the residue (*Intra prediction info*, in Fig. 2.1).

Before being encoded, the residue is processed by a mathematical transform ($T$ module, in Fig. 2.1), which converts the values from the spatial domain to the frequency domain in order to de-correlate the residue and concentrate the energy in a few coefficients. Then, a quantisation step ($Q$ module, in Fig. 2.1) is applied to transformed coefficients to eliminate small values associated to spectral components that are not perceptually relevant, decreasing the amount of data to be encoded without losing important information. Finally, the entropy coding processes the symbols that represent quantised transform coefficients to reduce their redundancy.

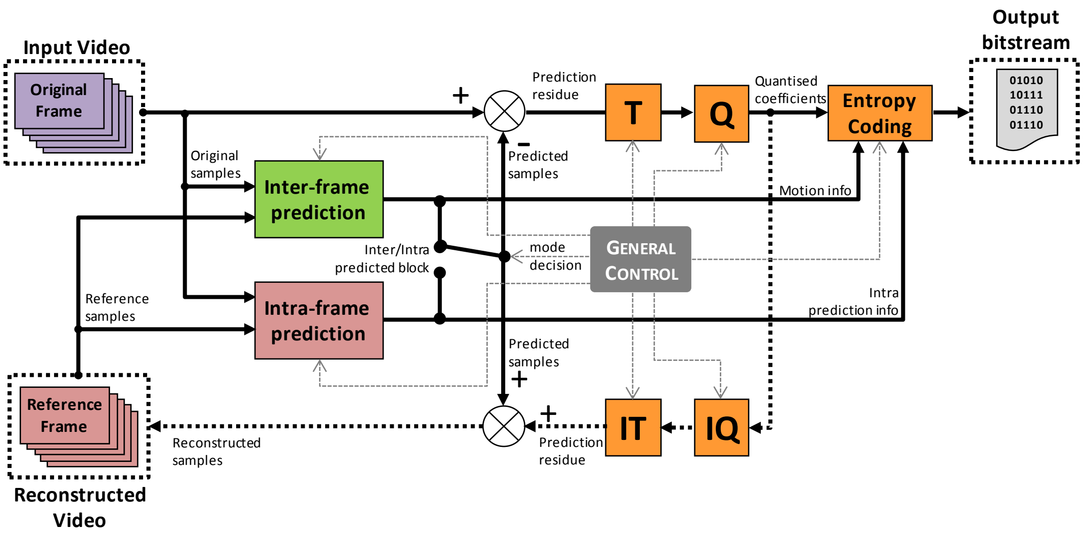


Fig. 2.1: Generic hybrid video compression system (encoder).

Many methods for intra/inter-frame prediction, data transformation, quantisation and entropy coding are used in the current video coding standards. The *General Control* module presented in Fig. 2.1 is responsible for deciding which modes are tested by each module and which mode results in the best encoding performance. These decisions, shown as dashed grey lines in the figure, are performed according to the encoder implementation and affect directly its compression efficiency. A frequently used solution for this problem is the Rate-Distortion Optimisation (RDO), which is explained in section 2.5. The specific operations of the modules present in the HEVC standard are explained in section 2.6.

Since the prediction signal used to compute the residue sent to the decoder must be exactly the same signal used at the decoder to reconstruct the decoded images, a reconstruction loop must be present at the encoder. Such reconstruction loop is actually a decoder operating inside the encoder, which ensures that intra/inter-prediction at the encoder use the same sample values as the decoder. The data flow in the reconstruction loop is presented with dashed black arrows in Fig. 2.1.

The decoder is presented in Fig. 2.2. The input bitstream is parsed and decoded by the entropy decoder and information such as MVs, reference frame indices, intra prediction mode and coding mode are sent to their respective modules. The inverse quantisation (*IQ* module, in Fig. 2.2) and the inverse transform (*IT* module, in Fig. 2.2) process the quantised and transformed coefficients, respectively, generating the prediction residue that is added to the predicted samples obtained from the inter-frame or intra-frame decoding modules. Finally, the reconstructed samples (i.e., the residue added to the predicted samples) are stored and used as references by the intra-frame or the inter-frame prediction modules.

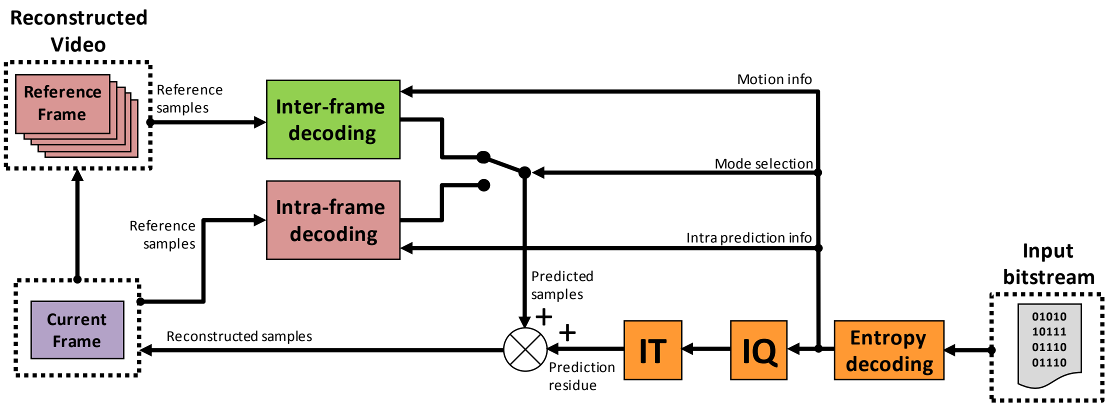


Fig. 2.2: Generic hybrid video compression system (decoder).

## 2.4 Distortion Metrics

Since all video coding standards introduce distortion, it is important to review the objective quality measures commonly used in video coding. Quality is a very difficult parameter to define and evaluate, because it depends on the viewer and such subjective nature of video quality is not easy to measure objectively. Although there are computable quality measures that estimate subjective grades, they are not used in this work because there is no wide consensus yet on which ones are the best and also because they are not commonly used in video codec design and evaluation. Objective metrics, on the other hand, are based on direct comparisons between pixels in two different images or blocks, such as the original and the reconstructed image, and for this reason they are easily computable and widely understood even in regard to their limitations.

The most used and known objective distortion metric is the Peak Signal-to-Noise Ratio (PSNR) [24], defined in (Eq. 1), where *MAX* is the maximum value that a sample can assume ($2^{n-1}$, where *n* is the number of bits per sample) and *MSE* is the Mean-Squared Error (MSE) for the image or block, calculated as in (Eq. 2). In (Eq. 2), *m* and *n* are the image dimensions and *O* and *R* represent the original and reconstructed luminance or chrominance samples, respectively. MSE is by itself a distortion metric that quantifies the difference between the samples from two images or blocks. Just as MSE, there are other metrics with the same purpose, which perform this function with larger or smaller computational complexity. This is the case of the Sum of Absolute Differences (SAD), a low-complexity distortion metric broadly used in video encoders. SAD is computed as shown in (Eq. 3), where *O*, *R*, *m* and *n* assume the same meaning as in (Eq. 2).

$$PSNR_{dB} = 20 \cdot \log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right) \qquad \text{(Eq. 1)}$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(R_{i,j} - O_{i,j}\right)^2 \qquad \text{(Eq. 2)}$$

$$SAD = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left|R_{i,j} - O_{i,j}\right| \qquad \text{(Eq. 3)}$$

### 2.4.1 Bjøntegaard Model

Based on the PSNR distortion metric, Gisle Bjøntegaard proposed in [27] a model that measures the compression efficiency difference between two algorithms. In general terms, the Bjøntegaard delta PSNR (BD-PSNR) measure corresponds to the average PSNR difference in decibels (dB) for two different encoding algorithms considering the same bit rate. Similarly, the Bjøntegaard delta rate (BD-rate) reports the average bit rate difference in percent for two different encoding algorithms considering the same PSNR.

A third order logarithmic polynomial fitting is used to approximate a Rate-Distortion (R-D) curve given by a set of $N$ bit rate values ($R_1$, ..., $R_N$) with their corresponding PSNR measurements ($D_1$, ..., $D_N$), as in (Eq. 4), where $\hat{D}(R)$ is the fitted distortion in PSNR, $R$ is the output bit rate, and $a$, $b$, $c$, $d$ are fitting parameters.

$$\hat{D}(R) = a \cdot \log^3 \cdot R + b \cdot \log^2 \cdot R + c \cdot \log \cdot R + d \qquad \text{(Eq. 4)}$$

To simplify notation, (Eq. 4) is rewritten as (Eq. 5), considering $r$ as the logarithm of the bit rate (i.e., $r = \log R$).

$$\hat{D}(r) = a \cdot r^3 + b \cdot r^2 + c \cdot r + d \qquad \text{(Eq. 5)}$$

With four R-D pairs (i.e., four bit rate values and their corresponding PSNR measurements) obtained from actual encodings, the four fitting parameters can be solved for a curve. Then, the average PSNR difference between two R-D curves corresponding to two different algorithms can be approximated by the difference between the integrals of the fitted curves divided by the integration interval, as in (Eq. 6), where $\Delta D$ is the BD-PSNR between the two fitted R-D curves $\hat{D}_1(r)$ and $\hat{D}_2(r)$, and $r_L$ and $r_H$ are the integration bounds obtained as in (Eq. 7) and (Eq. 8), respectively [27]. In (Eq. 7) and (Eq. 8), $r_{x,y}$ represents the bit rate value of point $y$ belonging to curve $\hat{D}_x(r)$, and $N_1$ and $N_2$ are the number of points in each curve.

$$\Delta D = \frac{\int_{r_L}^{r_H} (\hat{D}_2(r) - \hat{D}_1(r))dr}{r_H - r_L} \qquad \text{(Eq. 6)}$$

$$r_L = \max\{\min(r_{1,1},...,r_{1,N_1}), \min(r_{2,1},...,r_{2,N_2})\} \quad \text{(Eq. 7)}$$

$$r_H = \min\{\max(r_{1,1},...,r_{1,N_1}), \max(r_{2,1},...,r_{2,N_2})\} \quad \text{(Eq. 8)}$$

The bit rate as a function of the distortion is also expressed in the Bjøntegaard model through the third order polynomial given in (Eq. 9). The average bit rate difference between two R-D curves is approximated as in (Eq. 10), where Δ*R* is the BD-rate between the two fitted R-D curves $\hat{r}_1(D)$ and $\hat{r}_2(D)$, and $D_L$ and $D_H$ are the integration bounds obtained as in (Eq. 11) and (Eq. 12), respectively [27]. In (Eq. 11) and (Eq. 12), $D_{x,y}$ represents the distortion value of point *y* belonging to curve $\hat{r}_x(D)$, and $N_1$ and $N_2$ are the number of points in each curve.

$$\hat{r}(D) = a \cdot D^3 + b \cdot D^2 + c \cdot D + d \quad \text{(Eq. 9)}$$

$$\Delta R = 10^{\frac{1}{D_H - D_L}\int_{D_L}^{D_H}[\hat{r}2(D)-\hat{r}1(D)]dD} - 1 \quad \text{(Eq. 10)}$$

$$D_L = \max\{\min(D_{1,1},...,D_{1,N_1}), \min(D_{2,1},...,D_{2,N_2})\} \quad \text{(Eq. 11)}$$

$$D_H = \min\{\max(D_{1,1},...,D_{1,N_1}), \max(D_{2,1},...,D_{2,N_2})\} \quad \text{(Eq. 12)}$$

All methods proposed in this thesis for complexity reduction and scaling are compared to the original encoder version and to related works using the Bjøntegaard measures.

## 2.5 Rate-Distortion Optimisation

In order to achieve optimal R-D efficiency, a video encoder must be able to select the best coding modes for any particular video sequence. In other words, given a particular bit rate constraint, the encoder must select a coding mode that returns minimal image distortion. This constrained optimisation problem can be mathematically described as follows:

Let *S* represent all allowable modes and *i* represent an element of *S*, (i.e., $i \in S$). Then,

$$\begin{aligned} & i^* = \arg\min_{i \in S} D(i) \\ & \textit{subject to } R(i) \leq R_T \quad , \end{aligned} \qquad \text{(Eq. 13)}$$

where *i** is the optimal mode that minimises the distortion, *D*(*i*) is the distortion obtained with mode *i*, *R*(*i*) is the number of bits obtained with mode *i* and $R_T$ is the bit rate constraint.

This optimisation problem can be solved using Lagrangian methods, in which the distortion term is weighted against the bit rate term giving rise to an unconstrained problem [28]. The Lagrangian minimisation is represented in (Eq. 14), where *J*(*i*) is the R-D cost, *D*(*i*) and *R*(*i*) are the resulting distortion and bit rate when using mode *i*, and *λ* is the Lagrange multiplier [9, 29]. The coding mode *i** that returns the minimum cost *J*(*i*) is selected as the solution of the following unconstrained problem.

$$\begin{aligned} & i^* = \arg\min_{i \in S} J(i) \\ & \textit{where } J(i) = D(i) + \lambda \cdot R(i) \end{aligned} \qquad \text{(Eq. 14)}$$

As there are no simple models to describe the relationship between the coding modes and the R-D cost *J*, the RDO process implemented in current video encoders tests all possible coding modes and selects the one that results in the smallest R-D cost. Fig. 2.3 shows an example of an R-D space and an optimal R-D curve. In the figure, each point corresponds to a different mode tested and those that present the smallest distortion (*D* axis) for a given target bit rate (*R* axis) are in the optimal R-D curve.

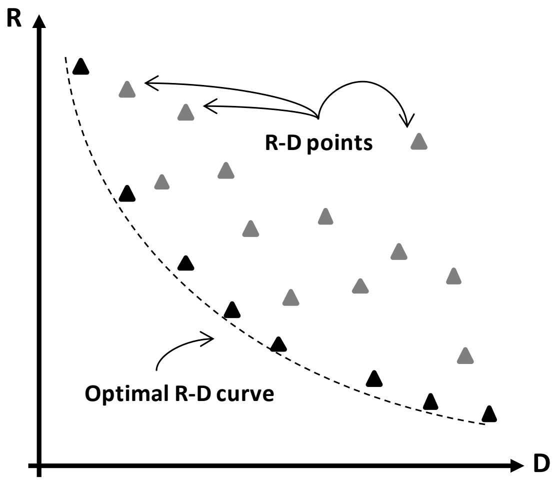


Fig. 2.3: Rate-Distortion (R-D) points and optimal curve.

The RDO technique is very effective and yields the best possible choice among all the encoding parameter sets. However, if every combination of operating modes is tested and evaluated by exhaustive search over the parameter and mode space, the computational complexity becomes a limiting factor. In practical applications, a number of modes must be ignored in the RDO process to comply with the limitations imposed by available computational resources, but the selection of such modes is not a trivial task. Understanding the operations of video coding standards is the key to design a video coding system that manages efficiently the computational resources consumed by the encoding process.

## 2.6 High Efficiency Video Coding

The High Efficiency Video Coding (HEVC) standard was finalised in March of 2013 by the Joint Collaborative Team on Video Coding (JCT-VC), a joint project of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). The standard has been designed to address the growing popularity of High Definition (HD) video and the emergence of beyond-HD formats, with support to encode multiple video views [6]. Currently, the High Efficiency Video Coding (HEVC) standard is gradually being introduced in the market and is expected to substitute H.264/AVC during the next years as the state-of-the-art video coding standard. This new standard is able to reduce bitrates by about 40%-50% over H.264/AVC through the use of several new tools and operating modes [9]. Obviously, as more operating modes are made available, the computational complexity involved in the encoding process of HEVC becomes higher than that of previous standards, as shown later in this thesis.

Fig. 2.4 presents the block diagram for the HEVC encoder. Initially, each frame of the input video is split into equal-sized block-shaped regions. The first frame in the video sequence is encoded using only intra-frame prediction, since there are no frames previously encoded to be used as reference in inter-frame prediction. The remaining frames may use both intra and inter-frame prediction.

During inter-frame prediction, the encoder predicts each block from previously encoded frames by using ME and MC. MVs obtained from the ME process determine the relative location of the best prediction block in the reference frame, which are used in the MC process. In intra-frame prediction, the *Prediction* module in Fig. 2.4 uses samples from the original and neighbouring reconstructed blocks from the current frame to predict a block according to a particular prediction mode, which is determined by the *Mode Estimation* module in Fig. 2.4 .

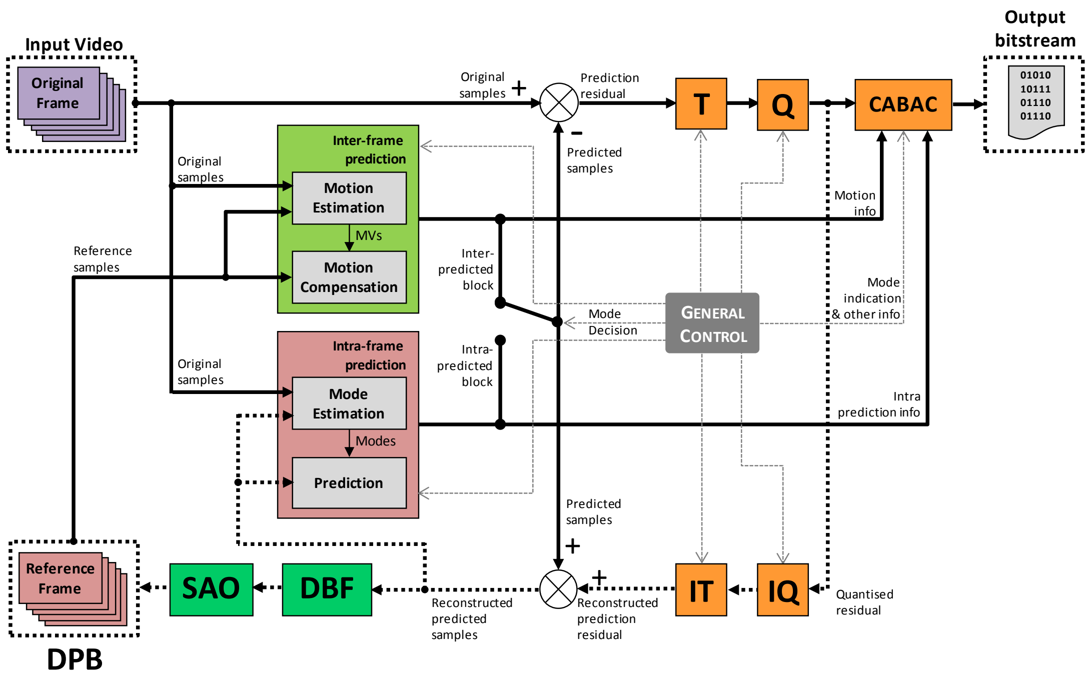


Fig. 2.4: Block diagram of a typical HEVC encoder.

After computing the prediction, the encoder calculates the residual, which is transformed by a linear spatial transform (*T* module, in Fig. 2.4), quantised (*Q* module, in Fig. 2.4), and entropy coded (*CABAC* module, in Fig. 2.4), generating bits that are multiplexed with motion and intra prediction information, mode indication, and other encoding information, finally resulting in the compressed bitstream.

Similarly to the generic hybrid video encoder presented in Fig. 2.1, the HEVC encoder also duplicates the decoder processing loop, such that both the decoder and the encoder can use the same reference samples for intra/inter-frame prediction. Therefore, the quantised residue is fed to the inverse quantisation and inverse transform (*IQ* and *IT* modules, in Fig. 2.4) in order to reconstruct the residual information, which is added to the predicted samples to generate the reconstructed samples. The result of the addition is delivered to the Deblocking Filter (*DBF* module, in Fig. 2.4) and the Sample Adaptive Offset (*SAO* module, in Fig. 2.4) to smooth out artefacts caused by block-wise processing and quantisation. The filtered, reconstructed samples are finally stored in the decoded picture buffer (*DPB*, in Fig. 2.4) and used for prediction in future frames. The decoding loop is presented with dashed black arrows in Fig. 2.4.

After this brief overview of the HEVC encoding process, the next sub-sections explain in more detail the new features introduced by the standard.

## 2.6.1 Encoding Structures

Even though HEVC is also based on the classic block-based video coding scheme of previous standards, significant modifications were introduced to its encoding data structures, especially regarding frame partitioning. The research presented in this thesis relies heavily on methods for deciding the best frame partitioning structures during the encoding process, so that this sub-section provides the necessary background for the comprehension of the next chapters.

### 2.6.1.1 Video Partitioning Structures

In HEVC, each video sequence is divided into Groups of Pictures (GOP), which are limited by two frames that constitute Random Access Points (RAP) from which the decoder can start decoding without needing any previous frames. Fig. 2.5 illustrates a video sequence divided into a number of GOPs, where the frames in black represent RAPs.

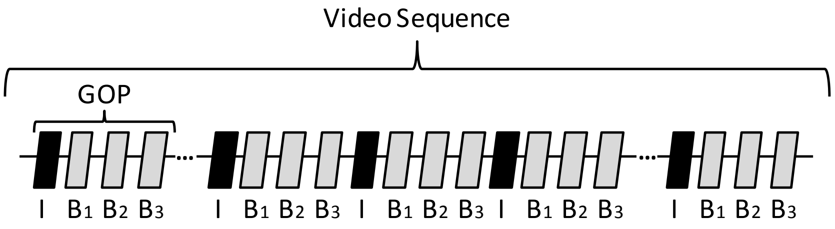


Fig. 2.5: Video sequence divided into a number of GOPs.

A GOP is composed of a set of pictures, or frames, which may be divided into a set of slices. Each slice is a part of the frame that can be decoded independently from other slices in the same frame, which is very useful in case of data losses or when applying parallel processing strategies, as shown later. Fig. 2.6 illustrates a frame divided into two slices.

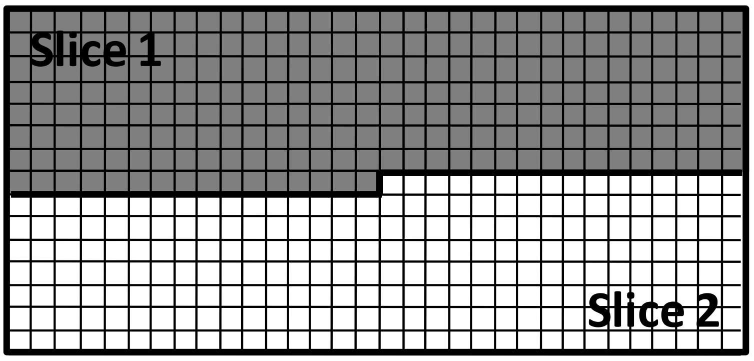


Fig. 2.6: Frame divided into two slices and several CTUs.

A slice is composed of sequential Coding Tree Units (CTU), which are further divided into Coding Units (CU). These structures are explained in detail in the next section. All CUs inside each slice are encoded according to the slice type, which may be I, P or B. In I slices, all CUs are encoded using only the intra-frame prediction. In a P slice, besides the intra-frame prediction used in I slices, the CUs can also be encoded using inter-frame prediction with one reference for MC (i.e., uni-directional prediction). In B slices, besides intra-frame prediction and uni-directional inter-frame prediction, CUs can also be encoded using inter-frame prediction with two references for MC (i.e., bi-directional prediction). Further details are presented in section 2.6.2.2.

#### 2.6.1.2 Frame Partitioning Structures

A slice is partitioned into a number of square blocks of equal size called Coding Tree Units (CTU), as shown in Fig. 2.6. Considering the 4:2:0 sub-sampling configuration, each CTU is composed of one luminance Coding Tree Block (CTB) of size $W \times W$ and two chrominance CTBs of size $(W/2) \times (W/2)$, where $W$ may be equal to 8, 16, 32 or 64. The luminance CTB and the two chrominance CTBs form the CTU, which is considered the basic processing unit of HEVC.

Each CTB in a CTU can be divided into smaller blocks, called Coding Blocks (CB), in the form of a quadtree structure, called the Coding Tree. The CTB is the root of this quadtree, which may assume variable depth, according to the encoder configuration. The Largest Coding Block (LCB) size and the Smallest Coding Block (SCB) size used by the encoder are defined in its configuration, but the maximum and minimum allowed size for a CB in HEVC are 64×64 and 8×8, respectively, so that up to four Coding Tree depths are possible. The same tree structure is usually applied to luminance and chrominance CTBs, except when the minimum size for chrominance blocks is reached.

In the HEVC Model (HM) encoder [30], the Coding Tree structure is defined through a iterative splitting process, which evaluates all possibilities in an RDO-based scheme, until the minimum CB size, which is usually 8×8 for luminance CTBs, is reached.

Fig. 2.7 shows an example of a 64×64 CTB divided into several CBs. The example shows the final Coding Tree division chosen after all the possibilities are evaluated. The tree leaves (grey blocks) are the final CBs belonging to the encoded quadtree. This flexible encoding structure allows the use of large CBs to encode large homogenous regions of a frame and small CBs in regions with more detailed texture.

For intra and inter-frame prediction, each CB may be divided into two or four Prediction Blocks (PBs), which are separately predicted. All PBs in a CB are predicted with either inter-frame or intra-frame prediction, so that the CB is said to be an inter CB or an intra CB. Fig. 2.8 presents all possible PB splittings that can be used for a CB. These possibilities are called PB splitting modes from now on in this thesis in order to distinguish them from the prediction modes, which will be defined later in this chapter. In HM [30], the best PB splitting mode is chosen with recourse to an RDO-based scheme, which evaluates the prediction using all PB splitting mode possibilities and compares their use in terms of bit rate and distortion. The total number of possibilities varies according to the CB size, as specified in Table 2.1. The same PB splitting mode is used for luminance and chrominance PBs, which together form the Prediction Unit (PU).

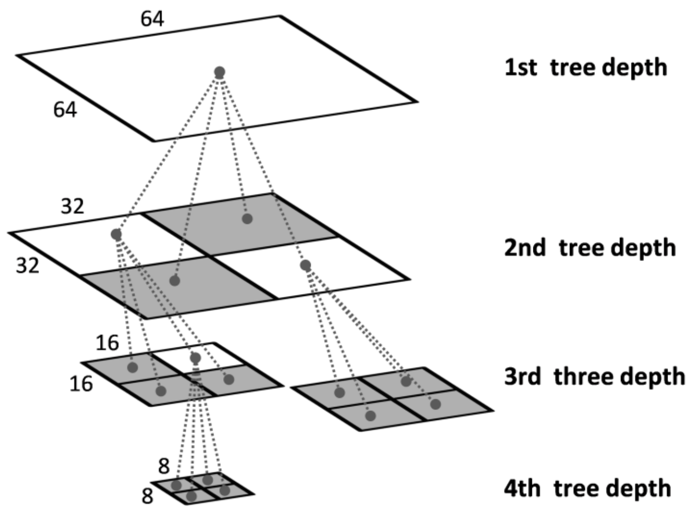


Fig. 2.7: Coding Tree structure of a CTB divided into CBs.

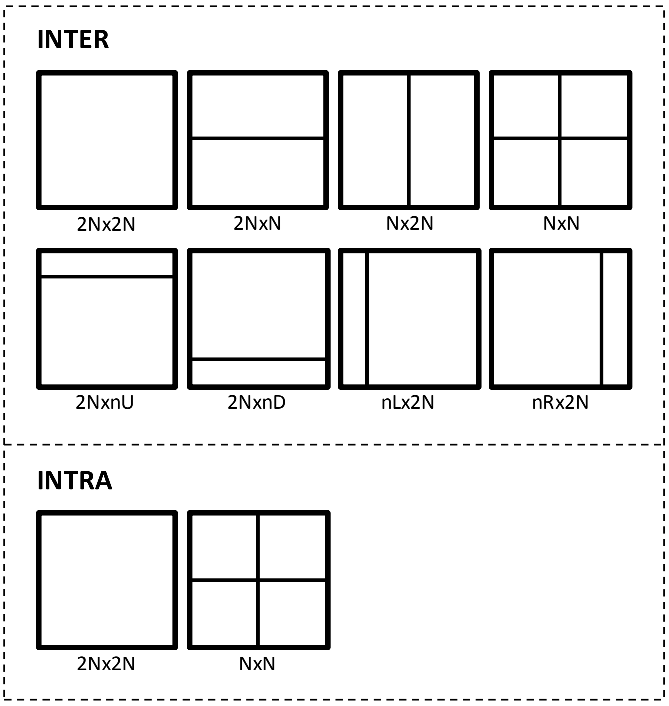


Fig. 2.8: Inter-frame and intra-frame PB splitting modes available in HEVC.

Table 2.1: PB splitting modes available for each CB size in HEVC.

| CB size | PB splitting modes | |
|---|---|---|
| | Inter | Intra |
| Larger than SCB | *2N×2N, 2N×N, N×2N, 2N×nU, 2N×nD, nL×2N, nR×2N* | *2N×2N* |
| SCB | *2N×2N, 2N×N, N×2N, N×N* | *2N×2N, N×N* |

When transform coding the prediction residual, each CB is assumed to be the root of another quadtree-based structure called Residual Quadtree (RQT). Likewise the CTBs, each CB is recursively partitioned into Transform Blocks (TB), which are the basic units to which both transform and quantisation operations are applied. The leaves of the RQT (grey nodes, in Fig. 2.9) are the final TBs, which are chosen in an RDO-based scheme in the HM encoder. The maximum and minimum TB sizes used by the encoder are defined in its configuration, but the maximum allowed size for a TB is 32×32 and the minimum size is 4×4, so that up to four RQT depths are possible. The same RQT structure is used for both luminance and chrominance CBs. A Transform Unit (TU) is formed by the luminance TB and its two associated chrominance TBs.

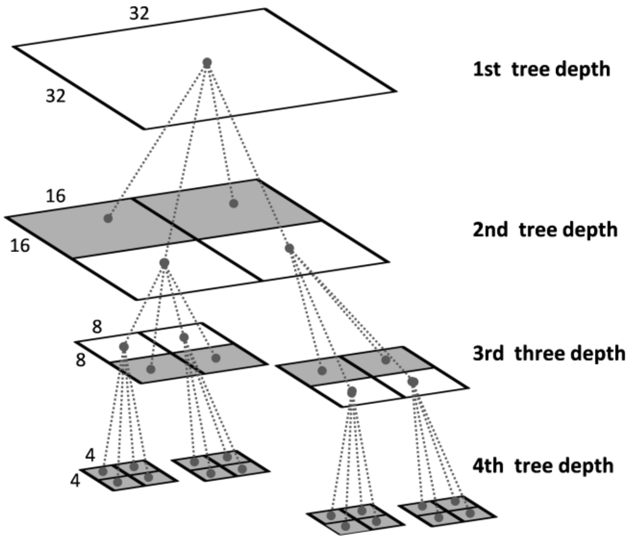


Fig. 2.9: RQT structure of a CB divided into TBs.

### 2.6.1.3 Parallel Processing Structures

The HEVC standard defines three data partitioning and processing orders designed to facilitate parallel processing. The use of these features may be decided depending on the encoder application context.

Tiles are defined as rectangular image regions, which enable a coarse but easy-to-implement parallel processing without requiring sophisticated thread synchronisation. Tiles can be independently decoded and encoded sharing some header information. When using tiles, the encoder segments the frame into rectangular regions, as shown in the example of Fig. 2.10, encodes and transmits them in raster scan order. All data dependencies are broken at the tile boundaries, so that independent encoding is performed for each one of them. The only exception is the Deblocking Filter (DBF), which can be applied across tile boundaries to reduce visual artefacts.

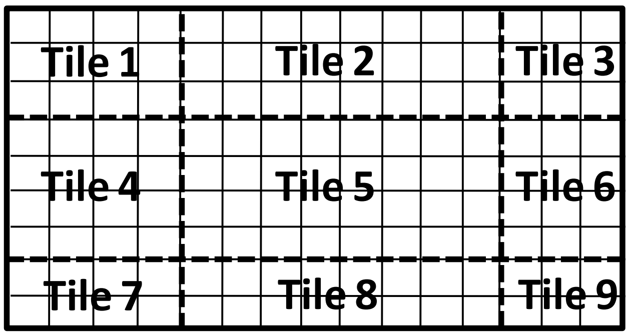


Fig. 2.10: A frame divided into nine tiles.

By using Wavefront Parallel Processing (WPP), a much finer degree of parallelism can be achieved. The slice is divided into rows of CTUs, which are processed according to the order presented in Fig. 2.11. The first CTU row of each slice is encoded in an ordinary way. The second CTU row starts being encoded after the two first CTUs in the first row are encoded, the third CTU row starts being encoded after the two first CTUs in the second row are encoded, and so on. The main advantage in the use of WPP instead of Tiles is the possibility of performing inter-frame or intra-frame prediction across the WPP boundaries, increasing the encoder compression efficiency.

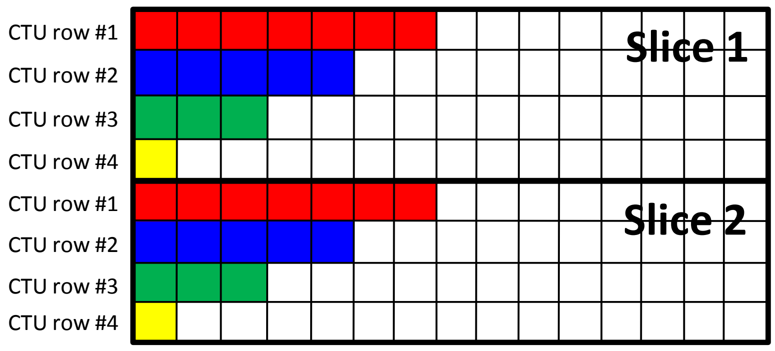


Fig. 2.11: Wavefront Parallel Processing encoding order.

Dependent Slice Segments allows data associated with a determined WPP or Tile to be encoded and assembled in a separate logical data packet for transmission, making that data available for fragmented packetisation with lower latency than if it were all coded together in one slice.

## 2.6.2 Encoding Process

This section presents a description of the modules that compose the HEVC encoder (Fig. 2.4). As the intra/inter-frame prediction, transform (T) and quantisation (Q) modules are the most important for the comprehension of the work described in the following chapters, they are presented in more detail, while a briefer overview is presented for the remaining modules.

### 2.6.2.1 Intra-Frame Prediction

As previously explained, the intra-frame prediction is responsible for decreasing spatial redundancy by predicting the samples of the current PB from those of neighbouring PBs already encoded in the same slice, i.e., located at the left and above the current PB. As previously shown in Fig. 2.8, a CB can be split according to two different splitting modes for intra prediction: *2N×2N* and *N×N*. Fig. 2.12 shows an example of intra-frame prediction in a $N\times N$ PB of size 16×16, where the samples within the current PB are presented in white, the left neighbouring samples in light grey, the top neighbouring samples in dark grey, and the top-left diagonal sample in black. In total, $4N+1$ samples are used for intra prediction in an $N\times N$ PB. If the CB has dimensions equal to the SCB, both *2N×2N* and *N×N* splitting modes are available for intra prediction. Otherwise, only the *2N×2N* splitting mode is available.

The intra-frame prediction of HEVC is very similar to that of H.264/AVC and is essentially extended to allow more prediction modes. As Fig. 2.13 shows, HEVC supports a total of 33 angular modes, a DC (flat) mode, and a planar (surface fitting) mode [6]. When one of the angular modes is used, the PB is predicted directionally from spatially neighbouring samples that were previously reconstructed, but not yet filtered, as shown in Fig. 2.4 by the input arrows of the intra-frame prediction module. The angular prediction consists of simply copying the neighbouring samples to the predicted block [31]. The DC mode computes an average of the neighbouring reference samples and uses it for all samples within the PB, building a flat surface for the whole block. Alternatively, the Planar mode calculates average values of two linear predictions using four corner reference samples, building an amplitude surface with a horizontal and vertical slope derived from the boundaries [6].

In order to decrease the computational complexity of the intra-frame prediction process, the HM reference software implements a Rough Mode Decision (RMD), or intra prediction mode estimation, which is followed by the actual prediction with pixel estimate computations. The RMD method was proposed in [32] and incorporated into the HM encoder as a solution to decrease the number of R-D cost computations for the intra mode decision. It consists in constructing a candidate mode list with the three best modes for 64×64, 32×32 and 16×16 PBs and the eight best modes in the case of 8×8 and 4×4 PBs.

The best modes are those which resulted in the smallest low-complexity R-D cost ($J_{LRD}$), computed as in (Eq. 15), where $SATD$ is the sum of absolute values of the Hadamard transformed coefficients of the prediction residue, $\lambda_{PRED}$ is the Lagrangian multiplier and $R_{PRED}$ is the number of bits necessary to encode the prediction mode information. Besides the best modes determined in this process, the list of candidate modes is also composed of the Most Probable Modes (MPM), which are inferred from the neighbouring PBs.

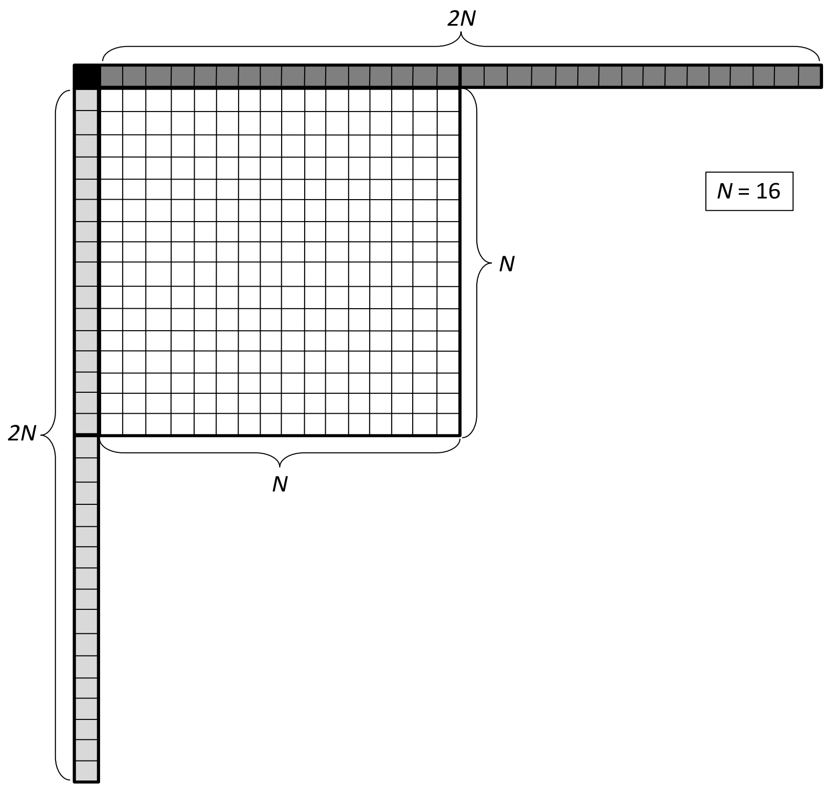


Fig. 2.12: Neighbouring samples used for intra prediction in an *N*×*N* PB of size 16×16.

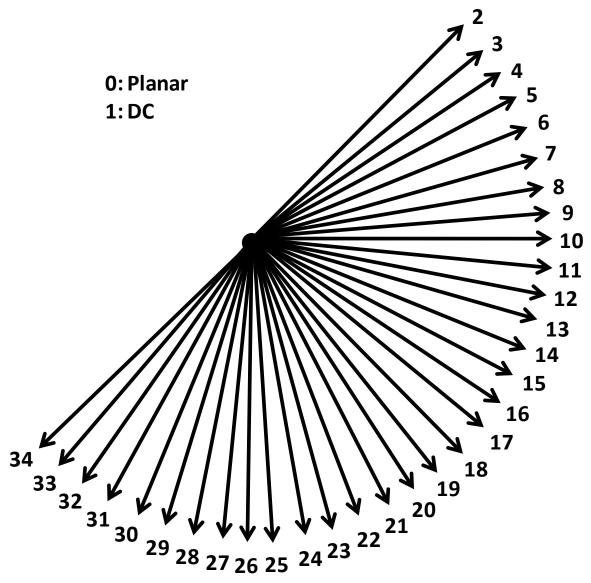


Fig. 2.13: Intra prediction modes tested for each PB.

$$J_{LRD} = SATD + \lambda_{PRED} \cdot R_{PRED} \qquad \text{(Eq. 15)}$$

After defining the list of candidate modes, the full R-D cost ($J_{FRD}$) for each of these modes is computed using (Eq. 16) [32], where $SSE_{Luma}$ and $SSE_{Chroma}$ are the sums of squared errors between the original block and the predicted block for luminance and chrominance signals, respectively. $\lambda_{MODE}$ is the Lagrangian multiplier and $R_{MODE}$ is the number of bits to encode the candidate mode. The mode with the smallest $J_{FRD}$ is then used in the actual prediction, which precedes the residue computation and transform coding.

$$J_{FRD} = (SSE_{Luma} + 0.57 \cdot SSE_{Chroma}) + \lambda_{MODE} \cdot R_{MODE} \qquad \text{(Eq. 16)}$$

The Lagrangian multipliers $\lambda_{PRED}$ and $\lambda_{MODE}$ in (Eq. 15) and (Eq. 16), respectively, are calculated in the HM encoder as a function of the QP, the number of reference frames and the temporal encoding configuration [30].

By using the RMD method, the computational complexity of the overall intra-frame prediction is decreased because the full R-D is computed only for those modes selected for the candidate list. However, the prediction step is still performed for every possible mode for a PB in order to calculate the low-complexity R-D costs.

#### 2.6.2.2 Inter-Frame Prediction

The inter-frame prediction of HEVC is based on MC, which allows predicting a PB using equal-sized areas from previously encoded frames used as references. The MC process forms a block of shifted pixels from the referenced frame by using a Motion Vector (MV), which describes the displacement of the PB from the reference to the current frame. The predicted block is then subtracted from the current block to compute the prediction residues that are inputted to the T and Q modules.

The process of determining the MV for a PB is the ME, which is the most demanding operation of the HEVC encoder in terms of computational complexity, as chapter 4 will show. The role of ME is to search within the reference frames for the most similar region to the current PB. When the best matching region is found, the ME algorithm computes the corresponding MV with a vertical and a horizontal component, indicating the relative location of that region with respect to the current PB position.

Similarly to H.264/AVC, HEVC supports ME with one quarter of pixel accuracy for luminance samples and one eighth of pixel accuracy for chrominance samples in the case of 4:2:0 format. In order to obtain the fractional-position luminance samples, two separable one-dimensional filters (one eight-tap filter and one seven-tap filter) are applied horizontally and vertically to generate the half-pixel and quarter-pixel samples, respectively. Fig. 2.14 shows two examples of fractional-position samples (*a* and *b*, in black) that were generated based on the integer position samples (in grey) located in the same line or column (dashed rectangles).

In the example given in Fig. 2.14, samples *a* and *b* are calculated as in (Eq. 17) and (Eq. 18), respectively, where the >> operator denotes arithmetic right shift, *B* is the bit depth of the reference samples (usually *B* = 8) and *qfilter* is the vector with filter coefficients for quarter-pixel interpolation, given in Table 2.2. Table 2.2 also presents the filter coefficients for half-pixel interpolation (*hfilter*) [6].

$$a = \left( \sum_{i=-3}^{3} A_{i,0} \cdot qfilter[i] \right) >> (B-8) \quad \text{(Eq. 17)}$$

$$b = \left( \sum_{j=-2}^{4} A_{0,j} \cdot qfilter[1-j] \right) >> (B-8) \quad \text{(Eq. 18)}$$

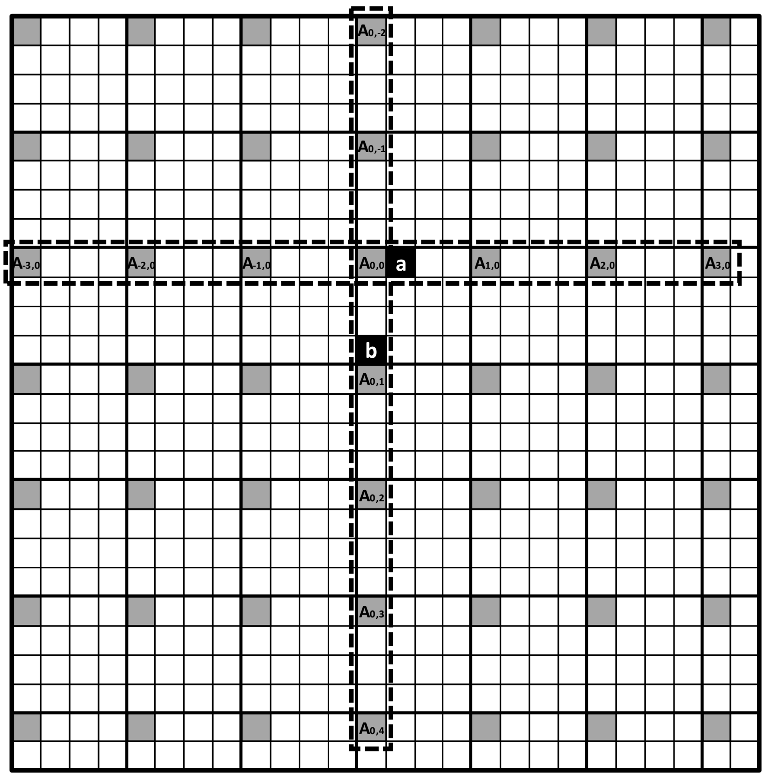


Fig. 2.14: Examples of sub-pixel interpolations from full-pixel samples.

Table 2.2: Filter coefficients for half-pixel and quarter-pixel luminance sample interpolation.

| index | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|
| hfilter | -1 | 4 | -11 | 40 | 40 | -11 | 4 | 1 |
| qfilter | -1 | 4 | -10 | 58 | 17 | -5 | 1 | |

Fractional chrominance samples are calculated similarly to the luminance samples by applying one-dimensional four-tap interpolation filters. A set of four four-tap filters are available for chrominance interpolation and the applied filter is chosen according to the distance between the fractional and the integer pixel. More details on luminance and chrominance interpolation filters can be found in [6].

Another important aspect of the ME process is the use of multiple reference frames. The encoder maintains two lists (*List 0* and *List 1*) of reconstructed images to be used as references in the next frames. *List 0* contains the indices of past frames in display order and *List 1* contains the indices to future frames encoded out of order, as explained in section 2.6.3.1. P slices allow the use of only one reference picture indexed by *List 0* and B slices can use up to two references indexed by both *List 0* and *List 1*. When bi-prediction is used in B slices, the encoder computes an average of the prediction performed from *List 0* and the prediction performed from *List 1*. Fig. 2.15 presents an example of four PBs in a frame, which are predicted from multiple references.

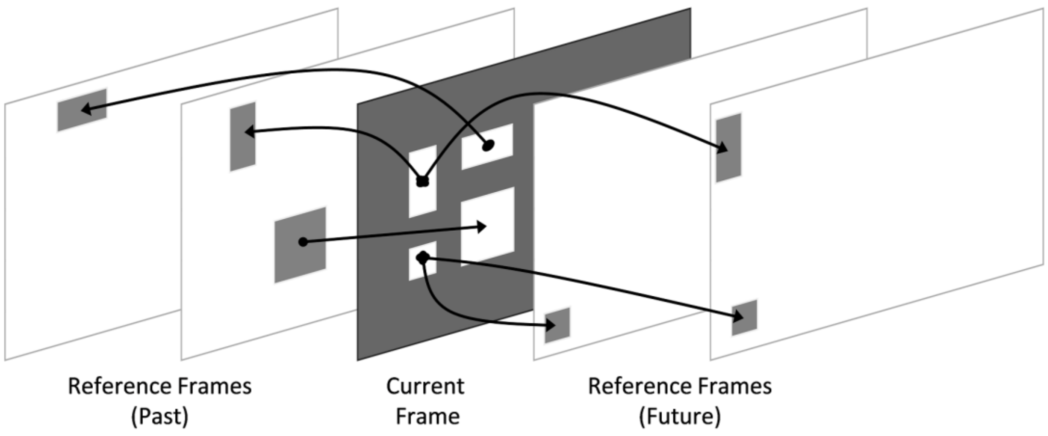


Fig. 2.15: Multiple reference frame prediction.

Block matching ME is used to determine the actual MVs representing the displacement of PBs in the current frame in relation to the best matching area in the reference frame. Several algorithms have been proposed for searching candidate blocks and the algorithmically simplest but the most demanding in terms of computation cost is the Full Search (FS). In FS, the best match is found by searching all possible candidate blocks in the search area (SA), which leads to the optimal result. However, the computational complexity involved in this process is very high and faster approaches need to be used.

Several Fast Motion Estimation (FME) techniques have been proposed in the literature in order to decrease the number of candidate blocks in the SA. Examples of algorithms implemented in the H.264/AVC reference software are the Unsymmetrical-Cross Multi-Hexagon-Grid Search (UMHexS), the Simple UMHexS (SUMHexS), and the Enhanced Predictive Zonal Search (EPZS) [33]. All of them are sub-optimal algorithms with R-D performance equal or smaller than that obtained with FS, with the advantage of significantly decreasing the ME computational complexity. The HM encoder uses both the FS and the sub-optimal Test Zone Search (TZS) algorithm [34] for ME with a fixed search range that can be configured before the encoding process starts.

Besides the MC/ME-based prediction described in the previous paragraphs, the inter-frame prediction of HEVC allows the use of a Merge/SKIP Mode (MSM) [35], which is conceptually similar to the *SKIP* mode in H.264/AVC. With MSM, the encoder can derive the motion information (i.e., the MV, one or two reference picture indices and the list associated to each index) from spatially or temporally neighbouring blocks, forming a merged region that shares the same MVs and picture indices. When MSM is selected for a CB, the encoder sends an index for a list containing all spatial and temporal neighbouring PBs available, known as the merge list. This index identifies which neighbouring PB is to be used as the source of motion information for the current PB. Besides the merge list index, the encoder also sends the reference picture list number and reference picture index to which the neighbour PB belongs. The *SKIP* mode also exists in HEVC, but it is treated as a special case of MSM. The *SKIP* mode is used when all Coded Block Flags (CBF) are equal to zero in a determined CB. In this case, only a *SKIP* flag and its corresponding merge index are transmitted to the decoder. The CBF will be explained in section 2.6.2.3.

When MSM is not used, the MV is encoded through the use of a Motion Vector Prediction (MVP) method [36]. Similarly to MSM, the encoder chooses a MV predictor among a set of multiple candidates derived from spatially neighbouring blocks. Then, the difference between the actual MV and the MV predictor, called Differential Motion Vector (DMV), is transmitted to the decoder together with the index of the candidate MV predictor chosen from the MVP list. When the reference index of the neighbouring PB is not the same as the current PB, the MV is scaled according to the temporal distance between the current and the reference pictures.

#### 2.6.2.3 Transform and Quantisation

The transform module ($T$ block, in Fig. 2.4) receives the prediction residue as input and transforms it to the frequency domain. The transformed residues are then processed by the quantisation module ($Q$ block, in Fig. 2.4). The HEVC standard uses a two-dimensional transform which is computed by applying one-dimensional transforms in the horizontal and vertical directions of the block. Transforms are computed for all TB tested in an RDO-based scheme.

HEVC uses integer Discrete Cosine Transform (DCT)-based transforms of sizes 32×32, 16×16, 8×8 and 4×4, which are applied to the TB, according to its size. In HEVC only the 32×32 transform matrix is defined and the remaining transformation matrices are derived from it by using part of its entries. For example, the transformation matrix for 16×16 TBs is presented in (Eq. 19) [6]. The matrices for 8×8 and 4×4 TBs can be derived from it by selecting only the first eight entries of lines 0, 2, 4, 6, 8, 10, 12, 14 and the first four entries of lines 0, 4, 8, 12, respectively.

An alternative 4×4 integer transform based on the Discrete Sine Transform (DST) is applied to 4×4 luma residue blocks resulting from intra-frame prediction. The reasoning behind the application of such transform is that it better fits the statistical property that residue magnitudes tend to increase as the distance from the boundary samples that are used for prediction becomes larger. In comparison to the 4×4 DCT-based transform, the use of DST-based transform to encode intra-predicted blocks results in a bit rate reduction of 1%. The 4×4 DST-based transformation matrix is presented in (Eq. 20) [6].

$$H = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 90 & 87 & 80 & 70 & 57 & 43 & 25 & 9 & -9 & -25 & -43 & -57 & -70 & -80 & -87 & 90 \\ 89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 & -89 & -75 & -50 & -18 & 18 & 50 & 75 & 89 \\ 87 & 57 & 9 & -43 & -80 & -90 & -70 & -25 & 25 & 70 & 90 & 80 & 43 & -9 & -57 & -87 \\ 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 & 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\ 80 & 9 & -70 & -87 & -25 & 57 & 90 & 43 & -43 & -90 & -57 & 25 & 87 & 70 & -9 & -80 \\ 75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 & -75 & 18 & 89 & 50 & -50 & -89 & -18 & 75 \\ 70 & -43 & -87 & 9 & 90 & 25 & -80 & -57 & 57 & 80 & -25 & -90 & -9 & 87 & 43 & -70 \\ 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\ 57 & -80 & -25 & 90 & -9 & -87 & 43 & 70 & -70 & -43 & 87 & 9 & -90 & 25 & 80 & -57 \\ 50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 & -50 & 89 & -18 & -75 & 75 & 18 & -89 & 50 \\ 43 & -90 & 57 & 25 & -87 & 70 & 9 & -80 & 80 & -9 & -70 & 87 & -25 & -57 & 90 & -43 \\ 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 & 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\ 25 & -70 & 90 & -80 & 43 & 9 & -57 & 87 & -87 & 57 & -9 & -43 & 80 & -90 & 70 & -25 \\ 18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 & -18 & 50 & -75 & 89 & -89 & 75 & -50 & 18 \\ 9 & -25 & 43 & -57 & 70 & -80 & 87 & -90 & 90 & -87 & 80 & -70 & 57 & -43 & 25 & -9 \end{bmatrix} \quad \text{(Eq. 19)}$$

$$H = \begin{bmatrix} 29 & 55 & 74 & 84 \\ 74 & 74 & 0 & -74 \\ 84 & -29 & -74 & 55 \\ 55 & -84 & 74 & -29 \end{bmatrix} \quad \text{(Eq. 20)}$$

The quantisation procedure used in HEVC is essentially the same as that used in H.264/AVC, basically consisting of a non-linear discrete mapping of the coefficient values into integer quantisation indices. It is implemented as a multiplication by a constant, an addition of a rounding factor and a right-shift controlled by a Quantisation Parameter (QP), which varies from 0 to 51. All transformed coefficients of a TB are quantised and inverse-quantised depending on the QP value. Table 2.3 presents the constant $Q_{QP\%6}$ values used in the quantisation calculation, where % is an operator that computes the remainder of the division between QP and 6. The quantisation is done according to (Eq. 21), where $L$ is the quantised coefficient (quantisation output), $C$ is the transformed coefficient (quantisation input), *offset* is a rounding factor and $N$ is the TB dimension [37].

$$L = \frac{C \cdot Q_{QP\%6} + offset}{2^{21+\frac{QP}{6}-\log_2 N}} \quad \text{(Eq. 21)}$$

Table 2.3: $Q_{QP\%6}$ values used in the coefficient quantisation.

| | QP%6 | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| $Q_{QP\%6}$ | 26214 | 23302 | 20560 | 18396 | 16384 | 14564 |

Rate-Distortion Optimised Quantisation (RDOQ) [38] has been adopted in HEVC as a non-normative quantisation optimisation technique. When the technique is enabled, the encoder computes four quantised coefficient candidates for each coefficient in each TB and then selects the best candidate in terms of R-D efficiency through repetitive processing.

After the transform and quantisation are finished, the Coded Block Flag (CBF) indicates whether or not the TB includes residual information. If the TB includes residue information the CBF value is set to 1; otherwise, it is set to 0.

### 2.6.2.4 Inverse Quantisation and Inverse Transform

The inverse quantisation (IQ) and inverse transform (IT) modules are responsible for performing the inverse operations of the quantisation and transform modules, as their name suggest.

The IQ is defined in (Eq. 22), where $CQ$ is the inverse quantised coefficient (IQ output), $L$ is the quantised coefficient (IQ input) and $N$ is the TB dimension [37]. The possible $IQ_{QP\%6}$ values are presented in Table 2.4, where % is an operator that computes the division remainder.

$$CQ = \frac{L \cdot IQ_{QP\%6} \cdot 2^{\frac{QP}{6}}}{2^{\log_2 N - 1}} \qquad \text{(Eq. 22)}$$

Table 2.4: $IQ_{QP\%6}$ values used in the coefficient quantisation.

| | QP%6 | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| $IQ_{QP\%6}$ | 40 | 45 | 51 | 57 | 64 | 72 |

Once the inverse quantised coefficients are calculated, the IT corresponding to the TB dimension is applied to them, resulting in the inverse transformed residue prediction. The inverse transformation matrices are defined as the transposes of their corresponding forward transformation matrices. For example, the inverse transformation matrix for 16×16 TBs is the transpose of the matrix presented in (Eq. 19). Similarly, the inverse transformation matrix of 4×4 DST is the transpose of the matrix in (Eq. 20).

### 2.6.2.5 Entropy Coding

The entropy coding process of HEVC uses only one tool, which is also available in H.264/AVC: the Context-Adaptive Binary Arithmetic Coding (CABAC). The CABAC receives the quantised coefficients, reorganises them according to one of the three available scanning orders (diagonal up-right, horizontal and vertical), selects a probability model for each syntactic element according to its context, updates the probability models and finally encodes the element. As the CABAC operation is not directly affected by any of the complexity reduction and scaling methods proposed in this thesis, it will not be described in further detail. More information about this module can be found in [6, 39].

#### 2.6.2.6 In-Loop Filters

Two filtering steps are applied to the reconstructed samples (i.e., after adding the predicted samples to the reconstructed prediction residue), before writing them into the DPB, as shown in Fig. 2.4. The first filter is the Deblocking Filter (DBF), which was already present in the H.264/AVC standard, and the second filter is the Sample Adaptive Offset (SAO), which is a new tool introduced in HEVC.

The DBF is applied to boundaries of CBs, PBs, and TBs larger than 4×4 pixels [40]. Vertical and horizontal edges are filtered according to a border filtering strength, which varies from 0 (no filtering) to 4 (maximum filtering strength) and also depends on the border characteristics. The SAO, on the other hand, is applied to all samples of the image, classifying the reconstructed pixels into different categories and then reducing distortion by adding an offset to the pixels in each category [41]. This classification is performed taking into consideration the pixel intensity and edge properties, e.g., based on gradient.

During the HEVC standardisation process, a third filter called Adaptive Loop Filter (ALF) was proposed. However, the ALF was not included in the final standard. Nevertheless, as part of the work presented in this thesis was done before the HEVC standard was finalised, the ALF filter was considered in the study presented in chapter 4.

Since the in-loop filter operation is not directly affected by any of the methods proposed in this thesis, the DBF and SAO operations will not be further detailed in this chapter, but additional information can be found in [40, 41].

### 2.6.3 Test Conditions, Temporal Configurations and Profiles

As in the case of previous standards, the HEVC specification [42] describes the standard syntax and decoding procedures. The encoder can be freely implemented as long as the generated bitstream respects the syntax and decoding rules defined by the standard. In order to guide tests and proposals submitted during the standardisation process, the JCT-VC group coordinated the development of a reference software encoder and decoder, colloquially known as HM, and published the *Common Test Conditions* (CTC) document [43], which defines a set of four temporal configurations to be used with the HM reference software for tests and comparisons of competing proposals for modification of the standard. Additionally, the standard also specifies three profiles, which are conformance points that define a set of tools or algorithms that can be implemented in the encoder according to its application (e.g., different profiles can be used for different types of device).

Four temporal configurations which differ in terms of temporal prediction from one another are defined in the CTC: *All Intra*, *Low Delay*, *Low Delay P*, and *Random Access*. Furthermore, two profiles are defined for video coding: *Main* and *Main 10*. The combination of the four temporal configurations and the two profiles constitute the eight testing conditions used in the latest versions of HM [43]. The CTC document is presented in the Appendix B of this thesis.

#### 2.6.3.1 Temporal Configurations

The encoding temporal configuration defines which frames can be used as references in the prediction process.

In the *All Intra* (AI) configuration, all images in the video sequence are encoded as Instantaneous Decoding Refresh (IDR) pictures, which are pictures that contain only I slices and therefore may be the first picture in decoding order, since they make no reference to others. When encoding video according to this configuration, inter-frame prediction is not performed and there are no pictures stored in the reference lists. The QP value is held unchanged during the encoding of the entire video sequence. Fig. 2.16 shows an example of a video sequence encoded with the AI configuration, where the encoding order is presented at the top of each picture. In this configuration, the encoding and the display order are the same.

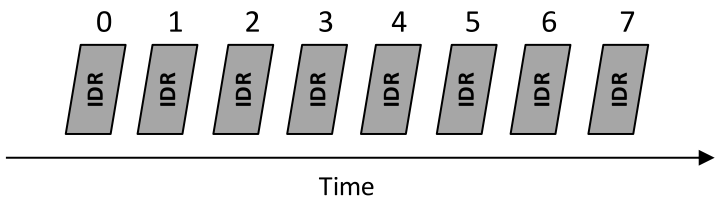


Fig. 2.16: Graphical presentation of the AI configuration.

In the *Low Delay* (LD) configuration, only the first image in the GOP is encoded as an IDR picture. The remaining images are encoded as Generalised P and B-pictures (GPB), which are B pictures that only allow using reference frames that appear before the current frame in display order. In other words, a GPB picture is able to use only reference pictures whose encoding order is smaller than the current picture. Both reference lists (*List 0* and *List 1*) are identical. Fig. 2.17 shows an example of a GOP encoded with the LD configuration. The number associated with each picture represents the encoding order. Notice that even though bi-directional prediction is allowed, each picture only uses previous frames (in display order) as references. The *Low Delay P* (LDP) configuration is a variation of the LD configuration that functions in a very similar way. The only difference is that only uni-directional prediction is allowed. This way, only one reference list (*List 0*) is maintained in the DPB.

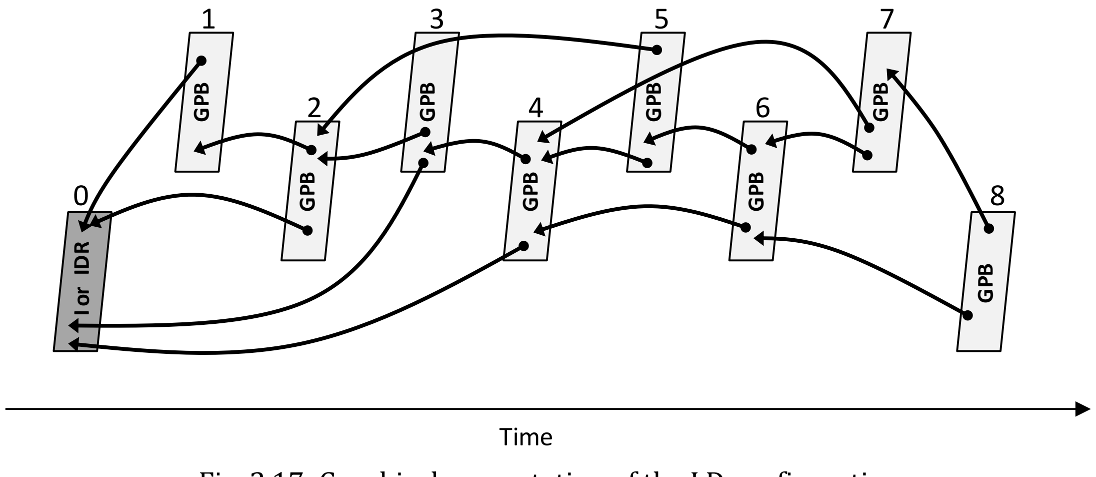


Fig. 2.17: Graphical presentation of the LD configuration.

Finally, the *Random Access* (RA) configuration is characterised by the utilisation of a temporal hierarchical B structure, which is illustrated in Fig. 2.18. The number associated to each picture represents the encoding order and the picture display order is represented from left to right. The first picture in a video sequence is encoded as an IDR and the remaining intra pictures are encoded as non-IDR intra pictures, characterising an open GOP. An open GOP means that frames outside the current GOP can be used as references. Every frame between two intra frames is encoded as a B picture. In the first temporal layer of Fig. 2.18, a GPB picture is used. In the second and third temporal layers, ordinary B pictures are used (i.e., bi-directional inter-frame prediction using both reference lists, which are not necessarily identical). Finally, the fourth and last temporal layer also uses ordinary B pictures, but these cannot reference each other.

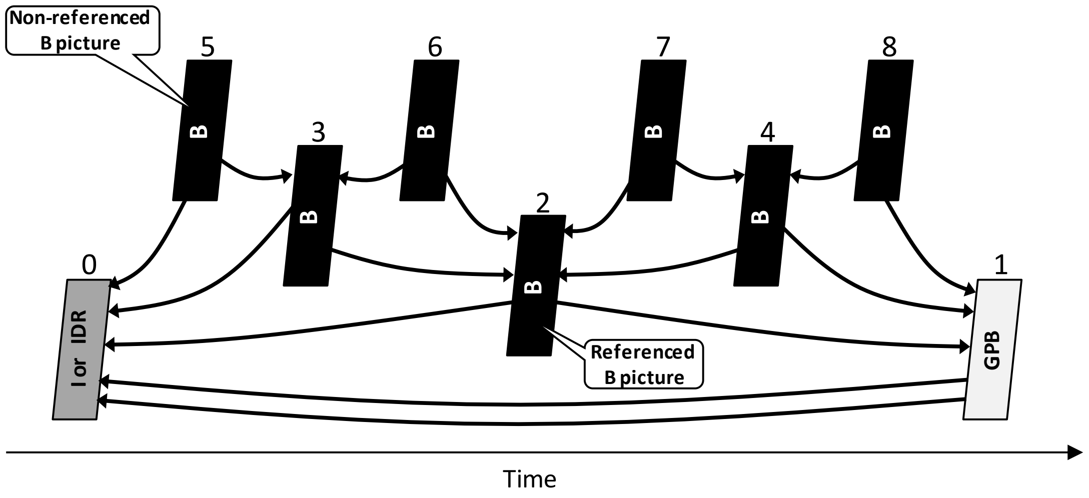


Fig. 2.18: Graphical presentation of the RA configuration.

### 2.6.3.2 Profiles

At the beginning of the HEVC standardisation, two profiles were defined: the *Low Complexity* (LC) and the *High Efficiency* (HE). The two profiles were different from each other in terms of coding tools and enabled functionalities, but several changes happened until the finalisation of the standard, so that at the end only one difference remained between them: the internal bit depth. For this reason, the LC and the HE profiles were removed from the standard and two new profiles were defined: *Main* and *Main 10*. The *Main* profile uses all tools described in the HEVC specification draft [42], most of which were covered in this chapter. The *Main 10* profile contains the same tools of *Main*, but the bit depth for both luminance and chrominance samples is set to 10 bits instead of 8 bits.

When the *Main* profile is used with a 10-bit source, each source sample is scaled to an 8-bit value prior to encoding in a process called Internal Bit Depth Decrease (IBDD). The scaling is obtained by applying the function $y = (x+2)/4$ to the input value $x$ and clipping the result $y$ to the [0, 255] range. Oppositely, when the *Main 10* configuration is used with an 8-bit source each sample is scaled to a 10-bit value before encoding by applying the $y = 4*x$ function to the input value $x$. This process is called Internal Bit Depth Increase (IBDI) and it allows greater precision in the video codec operations (i.e., improved encoding efficiency) at the cost of an increase in memory requirements, mainly to store reference pictures in the DPB.

All experiments presented in this thesis were performed using the *Main* profile (or the *Low Complexity* profile, in early versions of HM).

## 2.7 Computational Complexity of HEVC

As part of the research presented in this thesis, the computational complexity of the HM encoder was extensively characterised and quantified through several tests and comparisons, as presented in [10]. Such experiments concluded that the HM encoder presents a computational complexity increase varying from 9% up to 502% in comparison to the H.264/AVC encoder (High Profile), depending on the HEVC encoding configuration. Detailed experiments and results are presented in chapter 4.

The computational complexity and implementation of both HM encoder and decoder were also analysed in [44]. It was observed that in the AI encoder configuration the most time-consuming modules are the transform and quantisation (24.4%), due to the RDOQ technique, the intra prediction (16.6%) and the entropy coding (2.2%). In the RA encoder configuration, the ME takes up a significant portion of encoding time (38.8% for SAD calculations and 19.8% for fractional pixel search refinement). The decoding computational complexity is dominated by the IT (15.9%) and the filters (12.9%) in the AI configuration, and by the MC (24.8%) and the filters (12.4%) in the RA configuration.

Besides the encoding modules of HEVC, the encoding structures presented in this chapter, especially the frame partitioning structures, are also responsible for the high computational complexity involved in the encoding task. With a naïve implementation of the RDO process without simplifications, the process of defining the optimal combination of CBs, PBs and TBs would involve encoding each CTB using all possibilities allowed by the encoder, comparing their R-D costs and finally choosing the best one, which characterises a very complex procedure.

Also as a part of this research work, an analysis of the computational complexity for defining the frame partitioning structures of HEVC was performed. This analysis allowed concluding that the nature of such structures leads to nested encoding loops, such that CBs at large tree depths are encoded inside CBs at smaller tree depths, significantly increasing the encoding computational complexity. Detailed experiments and results of this analysis are also presented in chapter 4.

# Chapter 3

# Computational Complexity of Video Encoders – a Review

This chapter presents an overview of the state-of-the-art research on computational complexity of video encoders. First, a description of current methods for modelling the computational complexity of state-of-the-art video encoders is presented. Then, methods for reducing and scaling the expenditure of computational resources in video codecs are presented. Finally, the conclusions of the chapter highlight the research challenges and open topics not fully addressed in the available literature.

## 3.1 Modelling Computational Complexity

Estimating if the encoding operations will exceed the system's available computational resources is an important problem in the design of complexity-aware video coding applications. As the computational resources cannot be retrieved once spent, the system must be able to detect such cases without the need of performing the actual operations. Therefore, modelling computational complexity of video codecs is necessary to allow estimating the computational complexity required by a certain operation prior to its execution. Nevertheless, this is not a trivial task because video encoders are composed of several interdependent tools with parameterised functionalities that can influence significantly the level of computational complexity. Furthermore, according to [45], in the case of a complex multimedia system an overall view may not be enough for fully understanding inter-related modules as a whole; even though detailed information about all the constitutive functional modules and their theoretical complexity may be available, the overall codec behaviour and complexity are very difficult to be understood, because they depend on the input data characteristics.

In the recent past, modelling the computational complexity of video encoders and decoders was investigated in several works, some of which are presented in the next sub-sections. Even though there are still few works published on the topic aiming at the HEVC encoding/decoding process (section 3.1.2), many works for H.264/AVC are available in the technical literature (section 3.1.1).

### 3.1.1 Computational Complexity Modelling for H.264/AVC

In [46-48], the authors focus on modelling the computational complexity of inter-frame prediction operations of the H.264/AVC decoder due to their large share of the overall complexity. Since different inter-prediction modes demand different computational complexities, this was taken into account to estimate the overall decoding complexity in [46]. The number of MVs was used in [47] to estimate the decoding complexity of a macroblock (MB). According to the experiments performed, the decoding computational complexity is directly proportional to the number of MVs used in a MB. In [48], the number of interpolation filter operations was used for the complexity estimation. According to the authors, a MB with fewer interpolation filter operations has lower decoding complexity.

The computational complexity of the inter-frame prediction is modelled in [49] for the H.264/AVC decoder. The motion compensation (MC) complexity is modelled for each inter prediction mode as a linear function of the number of cache misses, the number of interpolation filters and the number of MVs per MB. Then, an H.264/AVC encoder equipped with the developed model is proposed, aiming at estimating the decoding complexity and choosing the mode that best meets a target decoding complexity.

There are also some approaches that deal with the computational complexity of video encoders and decoders based on their basic data structures. In H.264/AVC, these structures are blocks, MBs, frames and GOPs. In [50], the computational complexity of the H.264/AVC encoding process for a frame is estimated based on the computational complexity of encoding the previous frame. A frame is divided into multiple groups of MBs, which are further used by the complexity scaling algorithm, which assigns different complexity budgets to each group. The complexity of the frame is computed by simply adding the complexity of each group of MBs, which in turn is computed by summing up the complexity of each MB.

Some works have also proposed computational complexity models for video encoders that are dependent upon configuration parameters used for complexity scaling. The H.264/AVC encoder complexity was modelled in [51] by first dividing it into three major parts: ME (motion estimation), PRECODING (transforms, quantisation and picture reconstruction) and EC (entropy coding). The ME complexity is given by (Eq. 23), where $C_{SAD}$ is the computational complexity of a SAD operation and $\lambda_{ME}$ is the number of SAD calculations per frame, which is the complexity scaling parameter for the ME module. The PRECODING complexity is given by (Eq. 24), where $C_{NZMB}$ is the PRECODING computational complexity of one nonzero MB and $\lambda_{PRE}$ is the number of nonzero MBs in a frame, which is the complexity scaling parameter for the PRECODING module. Here, a NZMB is a MB with nonzero DCT coefficients after the quantisation. Finally, the EC computational complexity is given by (Eq. 25), and it is proportional to the bitrate $R$. In (Eq. 25), $C_{BIT}$ is the per-bit complexity associated to EC and $S$ is the size of the picture, which is needed because $R$ represents the coding bitrate in bits per pixel. The total computational complexity $C$ of the video encoder, is given by (Eq. 26), where $\lambda_F$ is the video frame rate. The model represents a complexity-scalable architecture for video encoding, for which the computational complexity is scaled by the parameters $\lambda_{ME}$, $\lambda_{PRE}$ and $\lambda_F$.

$$C_{ME} = \lambda_{ME} \cdot C_{SAD} \quad \text{(Eq. 23)}$$

$$C_{PRE} = \lambda_{PRE} \cdot C_{NZMB} \quad \text{(Eq. 24)}$$

$$C_{ENC} = S \cdot R \cdot C_{BIT} \quad \text{(Eq. 25)}$$

$$C(R; \lambda_{ME}, \lambda_{PRE}, \lambda_F) = \lambda_F \cdot (\lambda_{ME} \cdot C_{SAD} + \lambda_{PRE} \cdot C_{NZMB} + S \cdot R \cdot C_{BIT}) \quad \text{(Eq. 26)}$$

Similarly, in [1] the computational complexity of the H.264/AVC encoder is also modelled by dividing it into three modules: ME, PRECODING and EC. Two models are proposed: the first one considers that the RDO process is enabled, while the second considers that the RDO process is off. When RDO is used, the encoder executes the ME, transform, quantisation and entropy coding for each mode tested before choosing the best option. This is represented by equation (Eq. 27), where $C_{RDOon}$ is the encoding complexity of the frame, $C^{i}_{16x16}$ - $C^{i}_{I4x4}$ are the complexities associated to each prediction mode at the $i$th MB, $C^{i}_{M}$ is the complexity of the transform, quantisation and entropy coding for each mode at the $i$th MB, and $C^{i}_{PRE}$ and $C^{i}_{EC}$ are the complexities of the PRECODING and EC modules after the decision mode is performed at the $i$th MB. When RDO is off, the mode decision is only based on the results of the ME process, so that the model in (Eq. 28) does not include the $C^{i}_{M}$ element. In (Eq. 28), $C_{RDOoff}$ represents the total encoding complexity of the frame when RDO is off.

$$C_{RDOon} = \sum_{i=0}^{M-1}(C^{i}_{16x16} + C^{i}_{16x8} + C^{i}_{8x16} + C^{i}_{8x8} + C^{i}_{I16x16} + C^{i}_{I4x4} + C^{i}_{M}) + \\ + \sum_{i=0}^{M-1}(C^{i}_{PRE} + C^{i}_{ENC}) \qquad \text{(Eq. 27)}$$

$$C_{RDOoff} = \sum_{i=0}^{M-1}(C^{i}_{16x16} + C^{i}_{16x8} + C^{i}_{8x16} + C^{i}_{8x8} + C^{i}_{I16x16} + C^{i}_{I4x4}) + \\ + \sum_{i=0}^{M-1}(C^{i}_{PRE} + C^{i}_{ENC}) \qquad \text{(Eq. 28)}$$

In [52], the computational complexity of H.264/AVC encoders is modelled based on the complexity of each prediction mode (*SKIP*, *Inter 16×16*, *Inter 16×8*, *Inter 8×16*, *Inter 8×8*, *Inter 8×4*, *Inter 4×8*, *Inter 4×4*, *P8×8*, *I4×4*, *I16×16*). The model for the full computational complexity of a P frame is calculated as in (Eq. 29), where $W^{RD}_{i}$ indicates the normalised R-D computational cost for mode *i*, $W^{ME}_{i}$ denotes the factor of single-reference ME for the inter modes, $N_{MB}$ is the number of MBs in the frame, $N_{R}$ is the number of reference frames, and $W^{O}$ is the complexity consumed by the remaining encoder modules. The computational complexity of the *Inter 16×16* mode with a single reference frame was used as basis for the normalisations.

$$C^{Full}_{Frame} = N_{MB} \cdot \begin{bmatrix} W^{RD}_{0} + \sum_{i=1}^{7}(W^{RD}_{i} + W^{ME}_{i} \cdot N_{R}) + \\ W^{RD}_{8} + W^{RD}_{9} + W^{RD}_{10} + W^{O} \end{bmatrix} \qquad \text{(Eq. 29)}$$

The computational complexity of the H.264/AVC decoding process is modelled in [53] using an approach where each decoding module (DM) is separately modelled. Entropy decoding complexity is modelled as the product of bit decoding complexity and the number of bits involved, as shown in (Eq. 30), where $C_{vld}$ is the DM complexity, $k_{bit}$ is the average number of cycles required for decoding one bit and $n_{bit}$ is the number of bits for a frame. Side information preparation, including MB sum/clip and DBF strength calculation, is modelled as in (Eq. 31), where $C_{sip}$ is the DM complexity, $k_{MBsip}$ represents the average clock cycles for side information preparation per MB and $n_{MB}$ is the number of MBs in a frame. Inverse transform complexity is modelled in (Eq. 32), where $k_{MBitrans}$ is a constant that describes the complexity of MB de-quantisation and inverse transform and $n_{nzMB}$ is the number of nonzero MBs per picture. Intra-frame prediction is modelled in (Eq. 33), as the product between the average complexity of intra-frame prediction in one intra-coded MB ($k_{intraMB}$) and $n_{IntraMB}$ is the number of intra MBs per frame. The MC computational complexity model is given in (Eq. 34), where $k_{half}$ is the average complexity required to conduct a 6-tap Wiener filtering and $n_{half}$ is the number of filterings to decode a frame. The DBF is modelled as in (Eq. 35), where $k'_{\alpha}$ and $k'_{\beta}$ are the complexities to perform strong and normal filterings, respectively, and $n_{\alpha}$ and $n_{\beta}$ are the numbers of strong and normal filterings, respectively. Finally, the total complexity required to decode a frame is expressed as in (Eq. 36), where $k_{CU(DM)}$ is the complexity to decode a basic coding unit (a MB, for example) in a particular module, $N_{CU(DM)}$ is the number of basic coding units to be decoded by that module.

$$C_{vld} = k_{bit} \cdot n_{bit} \tag{Eq. 30}$$

$$C_{sip} = k_{MBsip} \cdot n_{MB} \tag{Eq. 31}$$

$$C_{itrans} = k_{MBitrans} \cdot n_{nzMB} \tag{Eq. 32}$$

$$C_{Intra} = k_{IntraMB} \cdot n_{IntraMB} \tag{Eq. 33}$$

$$C_{mcp} = k_{half} \cdot n_{half} \tag{Eq. 34}$$

$$C_{dblk} = k'_{\alpha} n_{\alpha} + k'_{\beta} n_{\beta} \tag{Eq. 35}$$

$$C_{frame} = \sum_{DM} C_{DM} = \sum_{DM} k_{CU(DM)} \cdot N_{CU(DM)} \tag{Eq. 36}$$

## 3.1.2 Computational Complexity Modelling for HEVC

As HEVC is still a recent standard, only a few works on the modelling of its computational complexity are currently available in the technical literature [54, 55].

In [54], an analysis based on data mining is used to construct a computational complexity model for HEVC using linear regression. The model uses encoder parameters to determine the time savings provided when using different configurations in comparison to a baseline case. The model is shown in (Eq. 37), where *G*, *H*, *I*, *J*, *K*, *L* and *M* are model parameters, *height* is the number of pixels lines per frame, *BR* is the target bit rate, *LCU* is the dimension of the CTU, *CUD* is the maximum Coding Tree depth allowed, *TUD* is the maximum RQT depth allowed in inter-predicted CUs and *TS* is the time saving obtained when using the current encoding parameters, in comparison to a reference configuration.

$$TS = \frac{H \cdot height + I \cdot BR - J \cdot LCU - K \cdot CUD - L \cdot TUD + M}{G} \qquad \text{(Eq. 37)}$$

In [55], the authors propose a load-balancing algorithm for parallel scheduling in HEVC. A model is used to predict the complexity of each slice and tile, in order to allocate the same amount of work for all cores before the encoding process starts. The computational complexity of each slice or tile is computed by summing up the complexities of all CTUs, which are individually computed as in (Eq. 38), where *CC* is the computational complexity of encoding a CTU, *CEM(s,m)* is the normalised complexity of each prediction mode *m* in a CU of size *s* (measured through offline encodings), and *CHK(s,m)* is the information of whether or not the mode is tested for the CTU. The prediction mode *m* and the CU size *s* belong to sets *S* and *M*, given by (Eq. 39) and (Eq. 40), respectively. If *m* is a valid mode for the current CU size *s*, *CHK(s,m)* is set to 1; otherwise, it is set to 0, as shown in (Eq. 41).

$$CC = \sum_{s \in S} \sum_{m \in M} CEM(s,m) \cdot CHK(s,m) \qquad \text{(Eq. 38)}$$

$$S = \{64 \times 64,\ 32 \times 32,\ 16 \times 16,\ 8 \times 8\} \qquad \text{(Eq. 39)}$$

$$M = \{MERGE,\ INTER,\ INTRA\} \qquad \text{(Eq. 40)}$$

$$CHK(s,m) = \begin{cases} 1, & if\, valid(s,m) \\ 0, & otherwise \end{cases} \qquad \text{(Eq. 41)}$$

## 3.2 Computational Complexity Reduction

Low-complexity algorithms for video encoding and decoding have been proposed in the last years as an attempt to enable the use of compressed video in complexity constrained platforms. As the encoder's computational complexity is much higher than the decoder's, most works focus on its modules and features.

### 3.2.1 Computational Complexity Reduction for H.264/AVC

Most works found in the literature focus on the ME and MD tasks because they are the most complex operations in the encoding process and so provide ampler room for decreasing the encoding computational complexity. As ME and MD are operations common to most video coding standards, most of the techniques presented in the next sub-sections can be applied not only to H.264/AVC, but also to HEVC with some adaptations.

#### 3.2.1.1 Motion Estimation

Several algorithms have been proposed for searching candidate blocks and the algorithmically simplest but most demanding in terms of computation needs is the *Full Search* (FS). In FS, the best match is found by searching all possible candidate blocks in a search area (SA) of the reference frame. However, the computational complexity involved in this process is very high and usually faster approaches are used.

Several *Fast Motion Estimation* (FME) techniques, which have been widely applied to H.264/AVC but were actually designed for use in any video coding standard that includes ME, have been proposed in the literature in the last decade. These techniques may be classified into two categories: (a) those which decrease the number of candidate blocks in the SA and (b) those which decrease the computational complexity required to compare such blocks to the current block (i.e., distortion measure computation, such as SAD or MSE).

Examples of the first category are the *Three-Step Search* (TSS) [56], the *Block-Based Gradient Descent Search* [57], the *New Three-Step Search* [58], the *Diamond Search* [59, 60], the *Hexagon-Based Search* [61], the *One at a Time Search* [23], the *Dual Cross Search* [62], and so on. All of them are sub-optimal algorithms with R-D performance equal to or smaller than that obtained from FS, with the advantage of significantly decreasing the ME computational complexity.

Methods from the second category include techniques such as sub-sampling [63], *Partial Distortion Search* [64], *Normalised Partial Distortion Search* [65, 66] and the

*Successive Elimination Algorithm* [67]. Even though these methods are capable of maintaining good R-D performance, their complexity reduction factor is limited.

There are still other works that reduce the ME computational complexity by applying different techniques. In [68], the authors propose a strategy based on sorting the MVs and coding modes such that the decision process is stopped when the rate required to encode a MV or coding mode exceeds the average of any previous MV or mode tested, thus skipping the evaluation of some cases. In [69], a controller is added to the encoder to extract signal statistics from the motion search and use them to dynamically configure the ME parameters, such as the number of reference frames, tested block modes and SA for each 16×16 block and its sub-partitions. A method for estimating which MBs can be encoded in *SKIP* mode without trying the other modes is proposed in [70]. The authors propose a Bayesian framework using the R-D cost difference between coding and skipping a MB as the single decision feature. Savings in processing time of more than 80% for low-motion sequences are claimed by the authors at a cost of small quality decrease.

#### 3.2.1.2 Mode Decision

The MD process involves different operations from various encoder modules. If RDO is enabled, then every possible coding mode is tried and the R-D performance results obtained using all the modes are compared in order to find and select the best mode. Many works based on heuristics have been proposed to decrease the number of modes to be tested during both intra-frame and inter-frame prediction.

In [71-74], the authors propose fast algorithms to select a subset of intra-frame prediction modes to be compared in the MD stage. The authors in [75] and [76] claim that not all intra modes need to have the R-D performance evaluated, especially those with a high *Sum of Absolute Transformed Differences* (SATD) value, since they tend to result in high encoding bit rates. Therefore, only those modes with an SATD value lower than a threshold are evaluated and the remaining modes are ignored, reducing the volume of computation to be performed.

In [77], the transforms are applied to the intra-frame prediction residue and the transformed coefficients are analysed in order to detect which is the best mode to be used. In [78] the low-frequency transformed coefficients for each MB are analysed in order to decide the best intra-frame block size (4×4 or 16×16).

A two-step algorithm for intra-frame MD was proposed in [79] based on heuristics. The first step consists in performing intra-frame prediction for every mode in both 16×16 and 4×4 prediction block sizes and computing the distortion between the original and predicted blocks. The mode leading to the smallest distortion is used as the best mode for its corresponding block size. In the second step, the MB homogeneity is computed to decide whether a 16×16 or a 4×4 block size should be used. A pruned DCT is applied to the original MB in order to compute a subset of coefficients, which are used to quantify the homogeneity level of the MB, which will then guide the block size choice. Homogeneous areas of the image must be encoded with large blocks, while heterogeneous areas must be encoded using small blocks.

In [80], a hierarchical fast inter-frame MD was proposed based on the evaluation of temporal stationarity, texture homogeneity and block border strength. The method is divided into three steps. In the first one, stationarity is detected by calculating the distortion between a MB and its co-localised MB in the reference frame. If stationarity is detected, the MB is encoded as *SKIP* and the MD process is terminated. Otherwise, the second step takes place and the homogeneity of the MB is computed as in [79] in order to decide if the ME is performed with large or small blocks. In the third and final step, the intensity variation of luminance samples along the edges of possible blocks is calculated in order to detect the better block shape for inter-frame prediction, eliminating the need of testing all the remaining block shapes. This method was integrated with the intra-frame decision heuristics proposed in [79], forming a complete MD scheme for H.264/AVC in [81].

According to [82], only 3% of the modes chosen in P frames are intra prediction modes, on average. This way, the evaluation of inter-frame modes prior to the evaluation of intra-frame modes is proposed in [82] as a way of reducing the necessary computational resources without significant decreasing image quality or bit rate. In [83] a set of heuristics is presented for speeding-up the complete MD process (i.e., both intra and inter-frame predictions). The authors observe that homogeneous and static regions are mostly encoded with inter 16×16 and *SKIP* modes, so that a Sobel Operator is used to detect homogeneity and the SAD calculation is used to detect stationarity. The proposed algorithm reduces the encoding complexity through RDO early termination when one of the specific conditions (homogeneity or stationarity) is true. If neither is true, all the remaining prediction modes are evaluated.

In [84] the inter-frame MD complexity is decreased by analysing the spatial continuity of the motion field, which is generated by ME using 4×4 pixel blocks. A set of experiments led to the conclusion that video regions with high motion continuity present a higher probability of being encoded with inter 16×16, 16×8 and 8×16 modes, while regions with low motion continuity are mostly encoded with inter 8×8 and smaller block sizes. As in [83], this work also uses the Sobel Operator to identify the borders of objects in an image.

Table 3.1 summarises the existing computational resource saving strategies for H.264/AVC. Categories “FME Cand.”, “FME Dist.” and “FME Other” correspond to approaches that (a) decrease the number of candidate blocks in the SA, (b) decrease the computational complexity of distortion calculation, and (c) apply mixed techniques. Categories “Intra MD”, “Inter MD” and “Full MD” correspond to FMD algorithms for intra-frame prediction, inter-frame prediction and both intra/inter-frame predictions respectively. Whenever available in the referenced paper, the amount of computational complexity reduction achieved with each approach is shown in the rightmost column of Table 3.1. These results are generally computed by comparing the algorithm proposed by the authors with the reference software or recommendation model and calculating the encoding time reduction. Exceptions are pointed out in the footnotes of Table 3.1.

It is important to notice that most of the approaches cannot be directly compared with one another in terms of complexity reduction because different encoder versions, encoding configurations, setups and experimental conditions (e.g., processor, memory, etc.) were used. Even so, the table provides useful information to the researcher interested in identifying the best computational resource saving strategies.

Table 3.1: Computational complexity reduction strategies for H.264/AVC.

| Category | Approach | Reference | Complexity reduction |
|---|---|---|---|
| FME Cand. | Three-Step Search | [56] | – |
| | Block-Based Gradient Descent Search | [57] | – |
| | New Three-Step Search | [58] | – |
| | Diamond Search | [59, 60] | – |
| | Hexagon-Based Search | [61] | – |
| | One at a Time Search | [23] | – |
| | Dual Cross Search | [62] | – |
| | Sub-sampling | [63] | – |
| FME Dist. | Partial Distortion Search | [64] | – |
| | Normalised Partial Distortion Search | [65, 66] | 45% - 65% [1] |
| | Successive Elimination Algorithm | [67] | – |
| FME Other | Sorting MVs and modes | [68] | 90% |
| | Dynamic ME configuration | [69] | 75% [1] |
| | Bayesian early-termination | [70] | 80% |
| Intra MD | Selective MD | [71] | 82% |
| | Selective MD | [72] | 42% - 52% |
| | Selective MD | [73] | – |
| | Selective MD | [74] | 60% |
| | Selective SATD-based MD | [75] | 79% |
| | Selective SATD-based MD | [76] | 85% [2] |
| | Transform coefficient analysis | [77] | 20% |
| | Transform coefficient analysis | [78] | 52% [2] |
| | Two-step hierarchical MD | [79] | 99.3% [3] |
| Inter MD | Three-step hierarchical MD | [80] | 99.4% [3] |
| | Motion field analysis | [84] | 50% |
| Full MD | Hierarchical MD | [81] | 99.3% [3] |
| | Inter MD before intra MD | [82] | 30% |
| | Homogeneity and stationarity detection | [83] | 30% |

[1] Time reduction for the ME module only;
[2] Time reduction for the intra-frame MD only;
[3] Reduction in the overall number of tests performed under the RDO scheme.

## 3.2.2 Computational Complexity Reduction for HEVC

Even though the HEVC standard has been recently launched, there are already some works published in the literature presenting methods to reduce the encoder's computational complexity. Most of these works aim at decreasing the computational complexity involved in the definition of the new frame partitioning structures, especially the Coding Trees, PUs and RQTs, and apply different techniques to determine the best configuration without testing all possibilities using an RDO process, as described in the next sub-sections.

### 3.2.2.1 Coding Tree Structure Determination

Fast algorithms for determining the Coding Tree structure are the most commonly found approaches to reduce the computational complexity of HEVC encoders, since the CU is the basic encoding unit and the Coding Tree determination is one of the most complex tasks of the encoding process, as chapter 4 will show in detail.

A fast splitting and pruning method for intra coding is proposed in [85]. It was found that the main contributor to the computational complexity of intra prediction in HM is the calculation of R-D costs for deciding intra prediction modes of CUs at all possible Coding Tree depths using RDO. In order to reduce these computational costs, the method uses a low-complexity R-D cost calculation, which result is tested to decide whether or not the CU splitting and pruning processes must be performed. The statistical parameters used in the Bayes-based choice of the best mode are periodically updated online so as to adapt to the changing characteristics of the video sequence. Experimental results have shown that the method is able to decrease the intra coding computational complexity by 50% with a BD-rate increase of 0.6%. However, the method is only applicable to intra-predicted CUs, which are a minority in the configurations that use inter-frame prediction.

Two fast RDO techniques for HEVC are proposed in [86] aiming at saving computational resources at small R-D performance loss. The first method, the *Top Skip*, avoids checking the R-D cost for large block partitions when they are unlikely to be chosen. A starting CTU depth is selected based on the minimum depth of the co-localised CTU in the reference frame. The second method, the *Early Termination*, avoids checking smaller blocks unlikely to be selected. The algorithm stops the CU splitting process if the best R-D cost is already lower than a given threshold. The threshold is adaptively computed based on the standard deviation of R-D costs relative to the CUs at spatially and temporally neighbouring CTBs. When both methods are integrated in the RDO process, a computational complexity reduction of 40% is achieved for the whole encoding process with an average BD-rate increase of 1.9%.

The methods proposed in [87] and [88] use information obtained from intermediate encoding steps to determine if a CU is split into smaller CUs. In [87], a depth range selection mechanism is proposed. If the best prediction mode found for a determined CU is the *SKIP*, the splitting process is terminated. In [88], the ratio between the R-D costs in the current and upper-depth CUs is calculated and compared to thresholds in order to early terminate the CU-splitting process. The methods [87] and [88] provide a computational complexity reduction of 48% and 38%, respectively, and average BD-rate increases of 1.7% and 1.2%, respectively.

A method based on motion divergence analysis is proposed in [89] to early terminate the splitting process of CUs. Before being encoded, each frame is downsampled and the optical flow of it is estimated based on the frame MVs in order to determine the motion divergence features. Then, for each CU, the motion divergence is evaluated as the variance of the optical flows of the current CU and its sub-CUs. The method yielded a computational complexity reduction of 43% at the cost of a BD-rate increase of 1.9%.

Temporal correlation in neighbouring frames is exploited in [90] to reduce the number of quadtree splitting decisions. Based on the tree depth used in the co-localised CU in the previous frame and its neighbouring CUs, the encoder decides whether or not to split the current CU into sub-CUs. Additionally, an early *SKIP* mode decision at the prediction stage is performed to further reduce the computational complexity. Experimental results show that the method achieves a computational complexity reduction varying from 20% to 33%, depending on the encoder configuration. BD-rate increases varied from 0.1% to 0.47%.

A fast Coding Tree depth decision based on spatial and temporal correlation is proposed in [91]. The authors claim that since successive frames are strongly correlated, especially with the high frame rates lately used in video sequences, the final depth information or split structure of co-localised Coding Trees in neighbouring frames is also highly correlated. The algorithm first determines the depth search range according to the similarity degree between neighbouring CTBs. Three classes of similarity are defined (high, medium, low). Then, once the depth range is settled, the depth levels at which spatial partitioning will be tried can be derived by selecting depths with high probability of occurrence and excluding low probability depths. Experiments have shown that the method is able to reduce the encoding computational complexity in 25% with an increase of 0.16% in bit rate. No results in terms of BD measures were made available.

In [92], a fast Coding Tree depth decision algorithm which operates in both frame level and CU level is proposed for computational complexity reduction of HEVC encoding. The algorithm focuses only on the inter mode early determination, since, according to the authors, mismatches in intra CU sizes would result in too large PSNR drops or bit rate increases. At frame level, the main idea of the method is to skip those depths which are rarely used in the reference frames. The number of CUs encoded at a certain tree depth in a frame is compared to a threshold in order to detect its level of usage. The CU part of the method relies on the fact that motion and texture detail of one particular part of the image tends to stay the same from one frame to another. By checking the spatial and temporal neighbouring CUs of a certain CU, the candidate CU depth can be predetermined. An average decrease of 45% in the ME computational complexity was achieved with this method. Bit rate increases remained under 0.3%. No results using BD measures were made available.

#### 3.2.2.2 Prediction Unit Structure Determination

Some methods to decide or early terminate the decision of the best PU structure have also been proposed in the last few years for HEVC. The next paragraphs summarise the most relevant works in this category up to date.

In [93], the authors propose optimised schemes that conditionally evaluate certain sets of modes according to intermediate encoding decisions. The method maintains square, intra and *SKIP* PU splitting modes unchanged and proposes a conditional evaluation for *Symmetric Motion Partition* (SMP) and *Asymmetric Motion Partition* (AMP) modes, which allegedly correspond to 60% of the computational complexity of the HM encoder (*Main* profile). Overall, a computational complexity reduction of 51% was achieved with the method, at the cost of a BD-rate increase of 1.3%.

In [94], a heuristic method aims at merging *N×N* PUs to form larger ones instead of performing ME for every possible PU partition. The method is applied to decide all PU sizes larger than *N×N*. When certain conditions are met, *N×N* partitions are merged into *2N×N*, *N×2N* or *2N×2N* partitions without performing the ME operations for each one of them. Initially, ME is performed only for the four *N×N* partitions in a CU. If the MVs for the four partitions are identical, they are merged into one single *2N×2N* PU. If the MVs are distinct, the rectangular shapes are tested in a similar manner. Experimental results point out an average computational complexity reduction of 34% with an average bit rate increase of 1.37% and a PSNR drop of 0.08 dB. No results using BD measures were made available.

In [95], a two-stage PU size decision algorithm is proposed to speed up the intra coding process in HEVC. In the first stage, before intra-frame prediction starts, texture complexity of CTUs and its sub-blocks are measured in order to filter out unnecessary

PU sizes. The threshold for filtering PU sizes is calculated dynamically according to the content of the video sequence and to pre-defined coding parameters. The frame texture complexity is calculated by downsampling each 64×64 CTU to a 16×16 block and then computing its variance. In the second stage, which takes place during the intra-frame prediction, the PU sizes of neighbouring 32×32 blocks are analysed in order to skip small PU sizes. The average computational complexity reduction for the intra coding process achieved in this work varies from 28.8% to 44.9%, depending on the video resolution. Average bit rate increases and PSNR decreases stayed under 0.47% and 0.02 dB, respectively. The authors do not present results using BD measures.

An early *SKIP* mode detection scheme is proposed in [96] for complexity reduction of the HEVC encoding process. According to the authors, *SKIP* mode is chosen in about 83% of CUs and detecting its occurrence would allow ignoring all the remaining modes in the RDO process. The proposed method pre-detects *SKIP* mode using the DMV and CBF information of inter *2N×2N* mode. The encoder first searches the best inter *2N×2N* mode (i.e., chooses between competition mode and merging mode) and after selecting the one with the minimum R-D cost, it checks the DMV and the CBF of it. If they are both equal to zero, then the best mode is determined as the *SKIP* mode and the remaining PU modes are not tested. The method reduced computational complexity by about 35% with a BD-rate increase of 0.5%.

Finally, [97] proposes a method in which small intra PUs are combined into larger intra PUs depending on the image characteristics (like texture variance), skipping the evaluation of certain modes. An online QP-based adaptive threshold generation is used to decide whether the smaller neighbouring PUs are to be combined in order to form a larger PU. A complexity reduction of 43.7% was achieved with the method at the cost of an average BD-rate increase of 1.26%.

#### 3.2.2.3 Residual Quadtree Structure Determination

There are also some works which try to decrease the computational complexity demanded in the process of deciding the best RQT structure. However, as the computational complexity reductions achieved when constraining RQTs is very limited, as chapter 4 will show, not many works exploit simplifications of the RQT determination procedure.

In [98] and [99], the computational complexity required to decide the RQT structure is reduced by early terminating the recursive TU splitting based on the number of nonzero transformed coefficients. The method in [98] proposes the concept of quasi-zero-blocks (QZB), which are defined by criteria based on the values of two quantities: the sum of the absolute values of coefficients and the number of nonzero coefficients. According to the authors, subtle differences between nonzero blocks and blocks with very small coefficients cannot be perceived by human eyes, so that QZBs can be used in an early termination scheme to stop splitting of TUs. An encoding time reduction of 22.8% was achieved with the method, at the cost of a PSNR decrease of 0.04 dB. No results using BD measures were reported.

Similarly, in [99] the authors claim that the sizes chosen for the TUs and the number of nonzero coefficients are strongly correlated. Accordingly, the number of nonzero coefficients is used in this method as a threshold to stop further R-D cost evaluation of the RQT. A computational complexity decrease of 61% in the TU processing was achieved in this method with small losses of compression efficiency. No results were reported regarding the effect on overall encoding complexity.

#### 3.2.2.4 Other solutions

Other solutions that reduce computational complexity by combining two or more approaches or by using a strategy different from constraining the decision of frame partitioning structures have also been proposed.

Works [100-102] optimise both the Coding Tree and PU structure decision processes. In [100], a complexity reduction scheme based on a method proposed in this thesis (presented later on, in chapter 5, and published in [11]) introduces a set of conditions that rely on spatial and temporal correlation to terminate early the Coding Tree splitting process. Simultaneously, the PU modes tested for a determined CU are chosen according to the size of neighbouring CUs. The computational complexity reduction achieved in [100] is 43% and the BD-rate increase is around 2.2%. The authors in [101] determine the CU depth range and the PU modes tested according to image characteristics and intermediate encoding results, such as motion homogeneity, R-D costs of neighbouring CUs and the PU mode chosen in the upper Coding Tree depth. Complexity was decreased by 42% at the cost of a BD-rate increase of 1.49%.

In [102], information from intermediate encoding steps is used to avoid exhaustive RDO searches over all possible Coding Tree depths and PU modes. Feature extraction to assist fast decisions is performed and the CU size decision is performed based on a Bayesian decision rule to avoid RDO search on all possible CU sizes and PU splitting modes. Predicting the splitting of a CU is formulated as a two-class classification problem and the features used for the decision include information such as variance of prediction error, SATD between original and predicted pixels, MV magnitude, R-D costs and others. An average complexity reduction of 41.4% was achieved with this method, in comparison to the original HM encoder. Average BD-rate increase is around 1.88%.

In [103], edge information from the current PU is used to reduce the number of candidate intra-frame prediction modes tested. Based on the luminance samples of each 4×4 PU, five edge strengths are calculated according to five linear filters and the orientation of the dominant edge (i.e., the one with the largest strength) is used to choose one among five pre-defined sets of modes to be tested in the intra-frame prediction. In PUs larger than 4×4, the edges are still calculated for sets of 4×4 samples and the edge with the largest number of occurrences among the sets is defined as the dominant edge of the PU. The method resulted in a reduction of 32.08% in the intra-frame prediction complexity and an average BD-rate increase of 1.3%.

Table 3.2 summarises the computational resource saving strategies for HEVC surveyed in this section. Categories “CU size/depth”, “PU mode” and “TU size/depth” correspond to low-complexity methods for deciding the best CU size or Coding Tree structure, the best PU splitting mode, and the best TU size or RQT structure, respectively. Category “Other” corresponds to solutions that include more than one strategy to reduce the encoding computational complexity or that cannot be classified into any of the previous categories. The computational complexity reduction achieved with each approach is shown in the rightmost column of Table 3.2. The results are computed in comparison to the HM software in terms of encoding time reduction. Exceptions are shown as footnotes of Table 3.2.

As mentioned before in regard to Table 3.1, it is also important to notice that most of the approaches presented in Table 3.2 cannot be directly compared with one another due to different encoding conditions. However, the table provides useful information for identifying the best computational resource saving strategies for HEVC published so far. By comparing the results of Table 3.1 and Table 3.2, it is possible to conclude that the research focusing on approaches for HEVC still have room for improvement to identify whether and how complexity reduction levels similar to those of the best performing methods developed for H.264/AVC can be achieved.

Table 3.2: Computational complexity reduction strategies for HEVC.

| **Category** | **Approach** | **Reference** | **Complexity reduction** |
|---|---|---|---|
| CU size/depth | Fast splitting for intra coding | [85] | 50% |
| | Top Skip + Early Termination | [86] | 40% |
| | Early termination | [87] | 48% |
| | Early termination | [88] | 38% |
| | Motion divergence | [89] | 43% |
| | Temporal correlation in neighbouring frames | [90] | 20% – 33% |
| | Spatial/temporal correlation | [91] | 25% |
| | Skip rarely used tree depths | [92] | 45% [1] |
| PU mode | Conditional evaluation | [93] | 51% |
| | Merge smaller PUs into larger PUs | [94] | 34% |
| | Filter out unnecessary intra PU sizes | [95] | 28.8% – 44.9% [2] |
| | Early *SKIP* mode detection | [96] | 35% |
| | Small intra PUs combined into larger PUs | [97] | 44% |
| TU size/depth | QZB-based early termination | [98] | 23% |
| | Nonzero coefficient-based early termination | [99] | 61% [3] |
| Other | Spatial/temporal correlation | [100] | 43% |
| | Image characteristics and intermediate results | [101] | 42% |
| | Feature extraction | [102] | 41.4% |
| | Edge information for intra mode decision | [103] | 32.08% [2] |

[1] Time reduction for the ME module only;
[2] Time reduction for the intra-frame MD only;
[3] Time reduction for the RQT processing only.

## 3.3 Computational Complexity Scaling

To efficiently manage computational complexity, it is not enough to implement low complexity methods. Since these methods usually incur in compression efficiency losses, they must be wisely and gradually employed in order to reduce computational complexity up to a certain desired target, avoiding unnecessary losses in terms of R-D efficiency. This is usually achieved by designing scaling systems in which computational complexity can be adaptively adjusted according to specific conditions, such as the device's battery status, time limitations imposed by the transmission environment and user preferences. Therefore, such systems generally have multiple operation modes, which can be dynamically chosen by sensing environment changes or even by a user's preference.

In the last years, dynamic scaling of computational complexity in video encoding has been a very active research field. The ideal solution for a complexity management system would be to extend the original RDO problem to a third dimension such as Rate-Distortion-Complexity Optimisation (RDCO) or Power-Rate-Distortion Optimisation (PRDO). This solution would find the best encoder configuration leading to the optimal visual quality under predefined rate and computational complexity constraints. However, joint Rate-Distortion-Complexity (R-D-C) analysis is an extremely complex task. Indeed, the lack of analytic models that relate rate, distortion and complexity prevents analytic solutions and the empirical solutions are not practical, as they require trying a huge number of possible combinations of modes and encoding parameters. As exhaustive search is usually infeasible in complexity-constrained environments, several heuristic solutions have been proposed to dynamically adjust the computational complexity of video encoding in order to manage the use of available computational resources.

### 3.3.1 Computational Complexity Scaling for H.264/AVC

Due to its high computational complexity, the ME process is used by many methods to scale the encoding computational complexity of H.264/AVC and other previous standards. A complexity scaling scheme for the MPEG-4 encoder was presented in [104] based on adjusting ME parameters and using the search options of FS, the TSS and the *Spiral Search* algorithms. The remaining parameters are the SA size, the SAD threshold for early termination of the *Spiral Search* algorithm, the use of pixel sub-sampling and the number of bits used to represent a pixel.

In [105], a classification-based method is proposed for the ME process. MBs are classified into different categories according to their importance in the frame and a complexity controlled ME scheme applies different operations to each MB according to its class. Initially, a total computation budget for the video sequence is divided into computation allowances for each frame. Then, the frame budget is divided into three independent sub-budgets, which are assigned to each class of MBs. When performing ME, each frame is classified into one of the three classes and a computation budget is allocated to each MB according to its class. Finally, according to the MB computation budget, the encoder uses more or less computation to estimate the motion of that MB.

Three other adjusting parameters for ME are proposed in [50]: partial cost evaluation for Fractional Motion Estimation (FRME), block size adjustment for FRME, and search range adjustment for Integer Motion Estimation (IME). By combining these parameters, 12 configurations presenting different trade-off between compression efficiency and computational complexity were defined and used. Based on the complexity measure from the previous frames, the complexity scaling algorithm allocates a certain budget for each group of MBs in an image and then for each MB within the group.

A two-stage complexity scaling method is proposed in [1] based on adjusting the ME operation. In the first stage, an encoding time scaling algorithm is applied. It consists of encoding the whole frame only using the inter 16×16 mode and then, based on encoding time information for this mode, estimating the total encoding time, the target encoding time and the parameters to be used in the second stage for complexity scaling. In the second stage, the number of *SKIP* MBs in the frame is used to adjust the encoder computational complexity to a determined target complexity level.

As in other methods aiming to reduce the computational complexity, many complexity scaling methods are based on the adjustment of both MD and ME operations. In [106] a scheme to scale complexity is proposed based on adjusting parameters that affect the aggressiveness of an early stop criterion for ME, the number of prediction modes tested in the MD, and the accuracy of ME steps for inter modes. Before deciding the number of modes to test, they are ordered based on the statistical frequency of the optimal modes for a given type of video, so that the first modes tested are those which most probably yield the best R-D performance. The computational complexity is scaled by adjusting one single parameter, which is mapped to the algorithmic parameters based on a rule tuned by an offline training process that uses several typical video sequences.

The approach of sorting modes was also used in [107] to scale the MD computational complexity. A ranking with the most popular ones, including intra and inter modes, compose a subset which is tested in the RDO process, while the remaining modes are suppressed from the tests. Initially, a few MBs are randomly selected for the frequency distribution analysis. Each mode is then associated with a frequency of occurrence and a computational complexity. Then, based on the target complexity to encode a complete image, the dominant mode set is chosen and used to encode the next frame.

In [108], the computational complexity scaling is divided into two problems: how to allocate the available computational resources to different frames and encoding modules and how to optimally use the allocated computational resources by adjusting the encoding parameters. To solve the first problem, a computation allocation model is proposed to distribute the available resources among the video frames. The second problem is solved by using a complexity-adjustable ME and a complexity-adjustable MD. The ME complexity is adjusted by allowing more or less operations (such as FRME, searching point refinement, etc.) to be executed, while the MD complexity is adjusted by allowing more or less modes to be tested in the RDO process. The list of tested modes is sorted according to their occurrence frequency in the spatial and temporal neighbouring MBs.

The MD complexity was scaled in [52, 109] by an adaptive computational allocation method at the MB level. The computational cost of ME in 16×16 blocks is taken as the basic unit and the computational costs for the remaining modes were obtained through empirical simulations and represented as weighting factors of the basic unit. A target complexity is calculated based on the total computational complexity estimated for a frame, which depends on the number of MBs in the frame and on the computational weighting factors previously defined. The computational budget allocated for each MB is calculated based on the target complexity, on the complexity consumed by the previously encoded MBs in the frame and on the number of MBs already coded. The computational budget for a determined MB is then used to adaptively choose which modes are to be tested and which are not.

A MD early termination method is used in [110] to decrease the encoding process computational complexity. By calculating the difference between the cost of encoding a MB as *SKIP* and an estimated coding cost, the encoder is able to stop the MD evaluation process just after encoding the MB as *SKIP*. A threshold calculated using conditional probability estimates of skipped and not skipped MBs is used in the early termination decision. The authors also propose a complexity scaling method which aims at maintaining a target level of complexity through a feedback algorithm that updates probability models to reduce R-D performance losses.

Other parameters which scale the complexity of other operations besides ME and MD are explored in some works. In [111], an empirical study on the controllability of parameters for complexity scaling on video encoding cloud services is presented. The work shows experimental results in terms of encoding time, bit rate and objective quality when varying the number of B frames, the level of refinement for sub-pixel ME and the operations performed in quantisation.

In [112], the number of reference frames, the method used for sub-pixel ME, the partition sizes allowed for intra-frame and inter-frame prediction and the quantisation approach are used as the parameters to adjust computational complexity. Considering all possible combinations among these four parameters would result in a total of 3360 possible parameter settings, a number so large that makes searching the best configuration infeasible in real time applications. Two fast algorithms are devised for finding the parameter settings which leads to high distortion-complexity performance. The algorithms are based on the *Generalised Breiman, Friedman, Olshen and Stone* (GBFOS) algorithm [113] and use training sequences to find the best parameter settings.

The number of motion search positions and the frame rate were the two parameters used in the method proposed in [114]. By using the *Adaptive Critic Design* technique [115], a class of approximate dynamic programming methods, an online complexity scaling scheme was developed based on neural networks.

A two-level method is proposed in [8]. In the frame-level algorithm, the encoding process is not changed in order to maintain acceptable image quality, but frames are dropped when necessary to decrease the amount of computations. In the per-frame algorithm, computational complexity is scaled for each frame in order to achieve the target coding time. The frame-level algorithm calculates a target encoding time for each frame in the video sequence based on the total delay experienced by the frame in the input buffer. The target time is then used by the per-frame algorithm, which adjusts computational complexity in a frame by increasing or decreasing the number of MBs encoded as *SKIP*, similarly to [110].

A dynamic framework which consists of a set of optimised core components is proposed in [116]. The ME, DCT, quantisation and MD processes can be configured to achieve a desired computation-performance trade-off in the encoder. These modules can all be assembled to form an H.264/AVC encoder with various degrees of computational complexity, which is able to adapt itself according to the available computational resources. Eleven parameters are used to adjust the computational effort of the different modules, such as the number of ME search points and whether or not DCT is applied to the residual MB. As determining the best combination of parameters for each video through exhaustive optimisation is quite computationally demanding, a simpler, sub-optimal greedy optimisation method is used.

RDCO and other similar approaches have also been proposed in several works, which treat the problem as an extension of RDO. In [51] and [117], a Power-Rate-Distortion (PRD) analysis framework was developed in order to build a parametric video encoding architecture which scales the computational complexity of its modules by varying encoding parameters. Based on the R-D behaviour of these parameters and on their associated computational complexity, a PRD model was created and used to determine the best configuration of parameters according to the available power supply level of the device in which the encoding is done and on the target bit rate. The same authors propose in [118] an operational approach for offline PRD analysis and modelling based on a wide set of training data. Based on the models developed, a control database for online resource allocation and energy minimisation is proposed.

Game theoretical analysis is used in [119] to model the power consumption in video encoders. The encoder is divided into modules, which are treated as players competing for the use of a computational resource on a limited budget aiming at maximising its efficiency. In [120], the complexity dimension was added to the RDO strategy. For each particular encoder setup, the total bit rate (R), PSNR (D) and ratio between the time spent to encode a training sequence and the time spent by the full-featured encoder (C) are calculated. The RDC points are then projected into a 2D set of points (lying in the D-C plane, for a given constant bit rate) and a lookup table is built from the points in the convex hull of the set in order to provide optimal starting RDC points. The trellis quantisation, the level of refinement in ME and the number of partitions allowed are the parameters adjusted.

Table 3.3 summarises the strategies for computational resource management proposed for use in H.264/AVC. Categories “ME” and “MD” correspond, respectively, to low-complexity ME and MD methods. Category “FR” corresponds to methods which change the output frame rate (i.e., discarding frames) in order to scale the encoding computational complexity. Categories “RDCO” and “Combined” correspond to Rate-

Distortion-Complexity Optimisation and combined strategies. The computational complexity reductions achieved are generally computed with reference to the H.264/AVC model encoder in terms of encoding time. Exceptions are shown as footnotes of Table 3.3.

Although most of the approaches presented in Table 3.3 cannot be directly compared with one another due to different encoding conditions, the results can still serve as guidelines for identifying the best computational resource management strategies.

Table 3.3: Computational complexity scaling strategies for H.264/AVC.

| Category | Approach | Reference | Complexity reduction |
|---|---|---|---|
| ME | Adjusting ME parameters and algorithms | [104] | up to 40% 1 |
| | MB importance classification | [105] | up to 60% |
| | Adjusting ME parameters | [50] | – |
| | Encoding time estimation | [1] | up to 70% |
| MD | Mode ranking | [107] | up to 70% |
| | SKIP early termination | [110] | up to 50% |
| ME, MD | Adjusting ME parameters and modes | [106] | up to 80% |
| | Frame-level resource allocation | [108] | up to 95% 2 |
| | MB-level resource allocation | [52, 109] | up to 91.2% |
| ME, FR | ME and frame rate (FR) adjustment | [114] | up to 90% |
| MD, FR | Frame-level and MB-level resource allocation | [8] | up to 50% |
| RDCO | Power-Rate-Distortion model | [117] | up to 78.6% 1 |
| | Game theoretical analysis | [119] | up to 95% 1 |
| | ME, MD and quantisation adjustment | [120] | up to 85% |
| Combined | ME, MD, quantisation adjustment | [111] | – |
| | ME, MD, quantisation adjustment | [112] | up to 94.1% |
| | ME, MD, transform and quantisation adjustment | [116] | up to 50% |

1 Power consumption reduction;
2 Uses a computational complexity measure based on the SAD computation cost.

## 3.3.2 Computational Complexity Scaling in HEVC

Even though several solutions for complexity scaling in H.264/AVC have been proposed, very few strategies have been published so far for HEVC. As in 3.2.2, the related works reviewed in this section were published within the last two years, so that they were not available when the research presented in this thesis started.

The authors in [121] present a complexity scaling scheme based on a method proposed in this thesis (explained later in chapter 5 and published in [12]). Their approach allows defining maximum Coding Tree depth values independently for each encoded frame, so that the computational complexity can be adjusted according to a given complexity budget. The initial frames of a video are normally encoded and then a game-theoretic approach is used to choose the depth values for the *N* next frames to be encoded. The maximum complexity reduction achieved by the method is around 40% with an average PSNR loss of 0.027 dB in such case. No results were made available using BD measures.

In [122], a complexity scaling scheme allows adjusting the number of evaluated PU splitting modes for inter-predicted CUs according to a target computational complexity. Through investigations on the MVs correlations, the authors propose a mode mapping method for PU splitting mode selection. Linear programming strategies are used to allocate the computational complexity and adjust the number of candidate modes. The maximum complexity reduction achieved by the method is around 50% with an average BD-rate increase of 5.9% in such case.

Table 3.4 summarises the two computational complexity scaling strategies for HEVC available so far in the literature. The maximum computational complexity reduction achieved with each approach is shown in the rightmost column of the table. Results are computed in comparison to the HM software in terms of encoding time reduction. By comparing Table 3.3 and Table 3.4, it is possible to conclude that the research focusing on approaches for complexity scaling of HEVC is still in its first steps. While several works have been proposed for H.264/AVC, only a couple of strategies are available for HEVC so far.

Table 3.4: Computational complexity scaling strategies for HEVC.

| **Category** | **Approach** | **Reference** | **Complexity reduction** |
|---|---|---|---|
| CU size/depth | Game-theoretic R-D-C optimisation | [121] | up to 40% |
| PU mode | Mode mapping-based mode selection | [122] | up to 50% |

## 3.4 Challenges and Conclusions

As explained in the previous sections, research on computational complexity management for video encoding through reduction and scaling of algorithm complexity made significant advances in recent years. However, the fast evolution of electronic devices in terms of computational power and display technologies and the development of new and more complex video coding standards, like HEVC, introduce new important challenges to be solved.

The challenges come from different sides. The increasing screen resolution of current multimedia-capable electronic devices and cameras allow higher resolution video sequences to be played, recorded and transmitted. Video content with higher resolutions require greater computational efforts to be processed and transmitted, increasing the power consumption in such devices. At the same time, such high-resolution screens require more energy to work, limiting even more the energy available for computational operations. In order to reduce the number of bits and decrease the energy spent on transmission, more efficient compression methods must be used, which in turn increases even more the computational complexity and power consumption. It is easy to perceive that there is no way to avoid or ignore the computational complexity increase incurred by the latest technology advances.

In addition, new video communication paradigms have arisen recently in the form of wireless Visual Sensor Networks (VSN), inter-vehicle communication networks with video transmission support and other ad-hoc networks which permit hop-to-hop video transmission or video diffusion through peer-to-peer (P2P) overlays. Since in many cases the nodes of these heterogeneous networks are mobile devices with limited energy and computation resources, the support of video encoding requires the use of carefully crafted domain-specific computational resource management techniques to ensure longer autonomies without significant encoded video quality degradations.

As discussed in the previous sections, many parameters of current generation encoders can be varied to reduce and scale the use of computation and energy resources of multimedia systems. This large number of parameters makes the analysis of the encoder's R-D-C efficiency a very complex task, but also allows finding better ways of reducing and scaling computational complexity. It is worth noting, however, that the HEVC reference software includes less encoding parameters than H.264/AVC, which on one side simplifies the analysis of the encoder R-D-C surface but on the other side reduces the number of encoding possibilities to be considered when developing a complexity scaling system. This is a challenge to be overcome by identifying in the HEVC standard which tasks should be parameterised in order to allow complexity scaling at a finer grain. Also, as different video sequences with different contents have different R-

D-C characteristics, developing efficient and accurate content-aware models is also another important open issue to be solved.

This chapter has presented a study on the main works published so far, focusing on computational complexity modelling, reduction and scaling for H.264/AVC and HEVC. As the HEVC standard draft has just been completed, most of the presented approaches focus on the computational complexity demanded by the ME and the MD processes of H.264/AVC. Some works aiming at complexity reduction based on fast decisions for the frame partitioning structures have also been exploited lately for HEVC. However, such methods for HEVC are still rare and do not achieve complexity scalability levels as large as the methods designed for H.264/AVC.

When comparing the tables presented in previous sections, it becomes clear that, due to the intrinsic lower computational complexity of H.264/AVC in comparison to HEVC and due to the fact that it has been available for a longer period for researchers, engineers and system designers, the approaches focusing on the former encoder achieve much higher computational complexity reduction levels than those focusing on the latter.

Chapter 4

# Performance and Computational Complexity Assessment of HEVC

This chapter presents an experimental study performed with the research goal of characterising and evaluating the behaviour and performance of the HEVC encoder. A first set of experiments was carried out in order to identify which tools most affect the HEVC encoding process in terms of encoding efficiency and computational complexity. Then, a second set of experiments was defined to analyse the impact of using different frame partitioning structures in both the compression ratio and encoding computational complexity.

The results of this study provided relevant insight for developing the novel complexity reduction and scaling methods presented in the following chapters of this thesis. Part of the experiments and results presented in this chapter were published in [10].

## 4.1 Analysis of HEVC Encoding Tools

As explained in chapter 2, the HEVC encoder includes a large number of tools, each one having a different contribution to the overall encoding efficiency and complexity. The operation of each tool and their functional modes are determined by configuration parameters that may be set to several different values during the encoding process.

This section presents an analysis on the impact in both the R-D efficiency and the encoding computational complexity when enabling and disabling each tool, as well as employing different parameter values whenever possible. By doing so, it is possible to identify which tools present the best trade-off between compression efficiency and computational complexity and thus should have enabling priority in a complexity-constrained system.

### 4.1.1 Experimental Setup and Methodology

The methodology defined for the experimental study presented in this section comprised two main steps. Firstly, the coding tools that present stronger impact in both coding efficiency and computational complexity were identified. To that end, the individual contribution of each and every encoding tool to such performance indicators was experimentally evaluated. Secondly, those tools identified in the first step and ordered according to encoding gain per complexity increase were selected for further analysis, which consisted in evaluating the impact of these tools when enabled in a cumulative sequence (i.e. first enabling tool *A*, then tools *A* and *B*, and so on).

To conduct the experiments, 12 video sequences that differ broadly from one another in terms of frame rate, bit depth, motion and texture characteristics as well as spatial resolution were used. The 12 selected sequences are the *BlowingBubbles*, *RaceHorses1*, *BasketballDrillText*, *PartyScene*, *BQMall*, *SlideShow*, *vidyo1*, *vidyo4*, *ParkScene*, *BasketballDrive*, *NebutaFestival*, and *Traffic*, which are all detailed in the Appendix B of this thesis.

The reference software used was the HM – version 7.0 (HM7) [123], which was compiled using *Microsoft Visual Studio C++ Compiler* under the *Release* compilation mode (i.e., allowing compiling optimisations). All tests were performed in a single core of a clustered computer based on *Intel® Xeon® E5520* (2.27 GHz) processors running the *Windows Server 2008 HPC* operating system. The computational complexity was measured in terms of processing time, reported by the *Intel® VTune™ Amplifier XE 2011* software profiler [124]. The *Low Delay* temporal configuration was used in all the tests.

### 4.1.2 Identification of Relevant Parameters

As revealed by experiments carried out in the scope of this chapter, as well as in other works conducted by the JCT-VC group, the impact of different HEVC tools on encoding efficiency and computational complexity is highly variable [125]. Since testing all possible combinations of coding tools and functional modes for all video sequences would require an inordinate amount of time to the point of being unfeasible and produce a huge amount of data, a preliminary exploration was first conducted to identify the configuration parameters that would be more important to this study. These exploratory experiments were performed by varying one parameter at a time, in the multidimensional encoder configuration parameter set, such that the impact of enabling each tool could be separately analysed. Starting with a baseline encoder configuration, each tool was enabled (and then reset to the default value), one after the other, and the resulting image quality, bit rate and encoding computational complexity were recorded for comparison with the reference baseline configuration. In every case, including the baseline configuration, the encoding structure was set to support CUs of up to 64×64 pixels, Coding Tree depths of up to 4 levels (i.e., minimum CU size is 8×8) and TUs varying from 4×4 to 32×32 pixels.

Table 4.1 shows all test cases corresponding to 17 encoding configurations. The baseline encoder configuration is defined as *TEST 1* while the other 16 configurations correspond to *TEST 2* through *TEST 17*. The table lists the parameter values used in active coding tools, while *D* and *E* represent a disabled or enabled tool/functional mode, respectively. Although a larger number of tests were performed using a broader spectrum of configuration parameters, only the 16 most representative ones in terms of PSNR, bit rate and computational complexity were selected and included in Table 4.1. Every test was performed using five different QPs: 22, 27, 32, 37, and 42.

Fig. 4.1 shows the computational complexity results obtained from encoding the 12 video sequences using the 17 test encoding configurations listed in Table 4.1. In Fig. 4.1, the computational complexity values were normalised with respect to *TEST 1* (reference configuration). For all cases, except *TEST 3*, one can notice a close similarity between the trends of lines in Fig. 4.1, which means that complexity varies fairly likewise for all video sequences when a specific tool is enabled. *TEST 3*, which evaluates the effect of increasing the ME search range from 64 to 128, shows different complexity values for each sequence most likely due to their very different motion characteristics. It is quite evident that video sequences with large motion activity, such as *RaceHorses1* and *BasketballDrive*, result in higher computational complexities than others with little or slow motion, such as *vidyo1*, *vidyo4* and *Traffic*. Encoding efficiency results, measured in terms of bit rate (also normalised with respect to *TEST 1*) and luminance PSNR (Y-PSNR) variation (using *TEST 1* as reference), are shown in Fig. 4.2 and Fig. 4.3, respectively, for the same 12 video sequences and 17 test configurations.

Fig. 4.1-Fig. 4.3 indicate that some configurations do not influence significantly any of the three performance metrics. For example, choosing configuration *TEST 15* has a very small impact on the computational complexity and on bit rate savings in comparison to *TEST 1*, leading to a slight decrease in Y-PSNR for most video sequences. Based on these results, those coding tools that were shown to have the largest impact on performance and complexity were selected as the basis for the second step of the computational complexity analysis process, which is described in the next section.

### 4.1.3 Relevant Encoding Configurations

Most studies on complexity analysis of video encoders focus on testing each feature independently, by comparing the performance of a baseline configuration against the same baseline configuration with only one tool enabled at a time. However, current video coding algorithms are characterised by high levels of inter-dependency between coding tools, which means that any additional encoding gain obtained by enabling a particular coding option may be dependent on the enabled/disabled status of other coding tools. Since this is the case of HEVC, the complexity analysis performed in this section is based on a sequence of predefined encoder configurations, where the coding tools are enabled in a cumulative order along such sequence. The sequence of relevant encoding configurations was constructed in two steps, as follows.

In the first step, 13 configurations from Table 4.1 were identified as those having significant impact on Y-PSNR, bit rate and overall computational complexity in comparison to the baseline test case (*TEST 1*). Here, significant impact was defined as Y-PSNR variations of at least 0.1 dB, bit rate changes of at least 1.5% and computational complexity increases of at least 5%. These 13 configurations correspond to seven coding tools – ME, DBF, SAO, ALF, *Internal Bit Depth* (IBD), *Linear Mode* (LM) *Intra Prediction* and *Non-Square Transforms* (NSQT) –, which were then used in the second step to create a sequence of 15 relevant encoding configurations.

Table 4.1: Encoder configurations for identifying relevant parameters.

| **Tool** | **Test Case (*TEST*)** | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| *Inter 4×4* | D | **E** | D | D | D | D | D | D | D | D | D | D | D | D | D | D | D |
| *Search Range* | 64 | 64 | **128** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| *Bi-prediction Refinement* [a] | 4 | 4 | 4 | **8** | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| *Hadamard ME* | D | D | D | D | **E** | D | D | D | D | D | D | D | D | D | D | D | D |
| *Fast Encoding* | E | E | E | E | E | **D** | E | E | E | E | E | E | E | E | E | E | E |
| *Fast Merge Decision* | E | E | E | E | E | E | **D** | E | E | E | E | E | E | E | E | E | E |
| *Deblocking Filter* | D | D | D | D | D | D | D | **E** | D | D | D | D | D | D | D | D | D |
| *Internal Bit Depth* | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | **10** | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| *Sample Adaptive Offset* | D | D | D | D | D | D | D | D | D | **E** | D | D | D | D | D | D | D |
| *Adaptive Loop Filter* | D | D | D | D | D | D | D | D | D | D | **E** | D | D | D | D | D | D |
| *Linear Mode Intra Prediction* | D | D | D | D | D | D | D | D | D | D | D | **E** | D | D | D | D | D |
| *Non-Square Transforms* | D | D | D | D | D | D | D | D | D | D | D | D | **E** | D | D | D | D |
| *Asymmetric Motion Partitions* | D | D | D | D | D | D | D | D | D | D | D | D | D | **E** | D | D | D |
| *Transform Skipping* | E | E | E | E | E | E | E | E | E | E | E | E | E | E | **D** | **D** | E |
| *Fast Transform Skipping* | E | E | E | E | E | E | E | E | E | E | E | E | E | E | E | **D** | E |
| *PCM Mode* [b] | D | D | D | D | D | D | D | D | D | D | D | D | D | D | D | D | **E** |

[a] Search range increase for ME when bi-prediction is used; [b] Pulse Code Modulation (PCM) mode allows a PU to be encoded with no prediction, no transform, and no entropy coding.

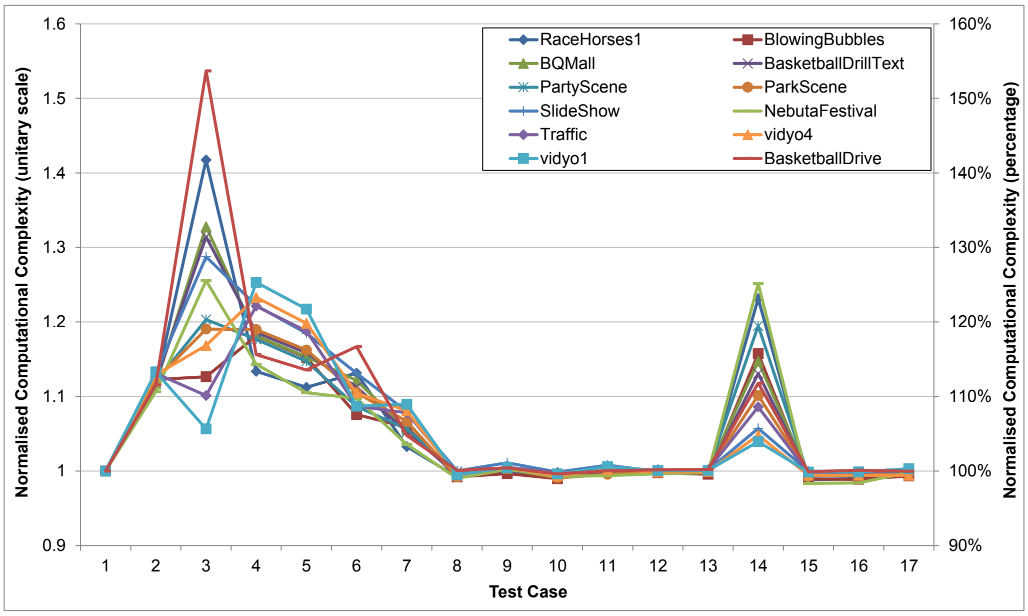


Fig. 4.1: Normalised computational complexity for encoding each video sequence under all 17 configurations (QP 32).

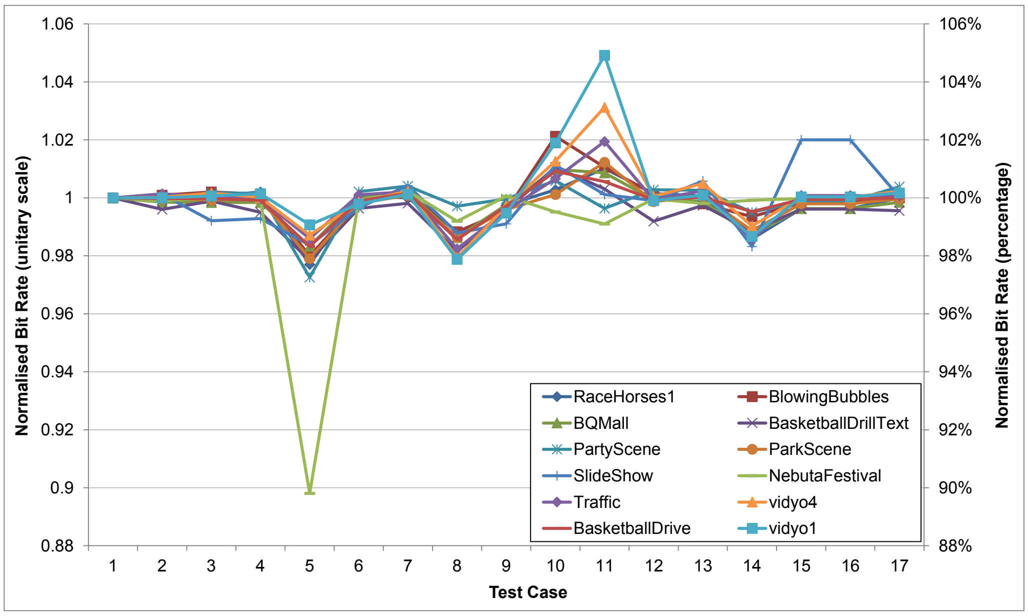


Fig. 4.2: Normalised bit rate for each video sequence encoded under all 17 configurations (QP 32).

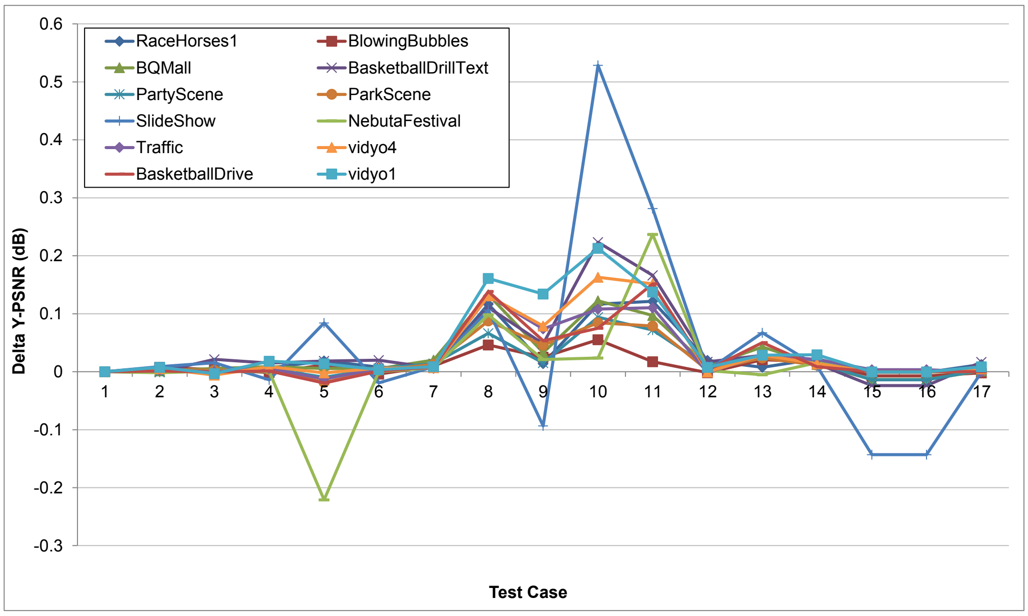


Fig. 4.3: Delta Y-PSNR for each video sequence encoded under all 17 configurations (QP 32).

In the second step, the 15 configurations (*CFG 1* to *CFG 15*) presented in Table 4.2 were created by defining the parameter values for each of the seven tools mentioned above, in a cumulative sequence. An additional configuration (*CFG 16*) was defined by choosing optimal tools and parameters, as shall be explained in detail in the next section. The baseline configuration (*CFG 1*) is the same as *TEST 1* in Table 4.1. Configurations 2, 4, 5, 6, 7, 8, 9, and 10 correspond to different functional modes of ME/MC; configurations 3, 11 and 12 vary filtering operations, enabling the DBF, SAO and ALF, respectively; *CFG 13* increases the IBD from 8 to 10 bits; *CFG 14* enables *LM Intra Prediction*; and *CFG 15* enables NSQT. As the tools were enabled in a cumulative order, *CFG 1* represents the simplest encoder configuration, while *CFG 15* is the most complex one, with all tools enabled.

The enabling order of the tools, presented in Table 4.2, was defined based on the ratio between the bit rate reduction and the increase in encoding computational complexity associated with each tool. The ratio between the Y-PSNR decrease and the complexity increase was also considered as a tiebreak whenever the bit rate-complexity ratio was too similar between two tools. Those tools which presented the highest relative coding efficiency-complexity gains were enabled before the others, since they should have higher priority of activation in complexity-constrained video coding systems.

The set of 16 configurations presented in Table 4.2 was used to encode all the test sequences listed in section 4.1.1. For each simulation, bit rate, Y-PSNR and complexity results were recorded for use in the performance and complexity trade-off analysis presented in the next section, which evaluates the activation of each HEVC tool/functional mode.

### 4.1.4 Encoding Performance and Complexity Trade-off Analysis

This section presents the performance results in terms of Y-PSNR, bit rate and computational complexity for the 16 test cases listed in Table 4.2. The results are summarised in Fig. 4.4-Fig. 4.8 and in Table 4.3.

The computational complexity results for all video sequences under all encoding configurations are plotted in Fig. 4.4, normalised with respect to those of *CFG 1*, as done in Fig. 4.1. In Fig. 4.4 all video sequences exhibit a similar monotonic increase of the encoding complexity from *CFG 1* to *CFG 15*. However, from *CFG 10* to *CFG 15* the slope is very small and the normalised computational complexity is approximately constant. The largest computational complexity increases are observed in the transitions from *CFG 4* to *CFG 9*. As shown in Table 4.2, these configurations correspond to different functional modes of ME according to the specified parameters. The results in Fig. 4.4 show that the choice of more accurate ME modes is accountable for most of the computational complexity increases observed in the results.

It is also noticeable in Fig. 4.4 that from *CFG 5* and especially from *CFG 6* onwards, the curves pertaining to the different video sequences start to be farther apart from each other. This happens due to the fact that these two configurations increase the ME search range from 64 to 96 and from 96 to 128, respectively. As explained in section 4.1.3 in regard to *TEST 3* in Fig. 4.1, the video sequences are quite distinct in terms of motion activity and for this reason they result in different encoder behaviour when the SR used in the ME is increased. The spreading of the curves is even more pronounced in the case of the configurations with rank order above *CFG 6* due to the cumulative effect of the tool activations. This is observed in the last configuration, *CFG 15*, for which the computational complexity is up to 3.2 times larger than that of the baseline configuration. Even though Fig. 4.4 presents results for QP 32, other QP values were also tested and showed similar behaviour.

Table 4.2: Encoder configurations used for complexity and performance analysis.

| Tool | Configuration Case (*CFG*) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| *Hadamard ME* | D | **E** | E | E | E | E | E | E | E | E | E | E | E | E | E | **E** |
| *Deblocking Filter* | D | D | **E** | E | E | E | E | E | E | E | E | E | E | E | E | **E** |
| *Asymmetric Motion Partitions* | D | D | D | **E** | E | E | E | E | E | E | E | E | E | E | E | **E** |
| *Search Range* | 64 | 64 | 64 | 64 | **96** | **128** | 128 | 128 | 128 | 128 | 128 | 128 | 128 | 128 | 128 | **64** |
| *Bi-prediction Refinement* | 4 | 4 | 4 | 4 | 4 | 4 | **8** | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | **4** |
| *Inter 4x4* | D | D | D | D | D | D | D | **E** | E | E | E | E | E | E | E | **D** |
| *Fast Encoding* | E | E | E | E | E | E | E | E | **D** | D | D | D | D | D | D | **E** |
| *Fast Merge Decision* | E | E | E | E | E | E | E | E | E | **D** | D | D | D | D | D | **E** |
| *Sample Adaptive Offset* | D | D | D | D | D | D | D | D | D | D | **E** | E | E | E | E | **E** |
| *Adaptive Loop Filter* | D | D | D | D | D | D | D | D | D | D | D | **E** | E | E | E | **D** |
| *Internal Bit Depth* | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | **10** | 10 | 10 | **8** |
| *Linear Mode Intra Prediction* | D | D | D | D | D | D | D | D | D | D | D | D | D | **E** | E | **D** |
| *Non-Square Transforms* | D | D | D | D | D | D | D | D | D | D | D | D | D | D | **E** | **D** |

Table 4.3 presents absolute complexity values for the least and most complex HM configurations (*CFG 1* and *CFG 15*, respectively) encoded with QPs 22, 27, 32, 37, and 42. As a reference for comparison, the complexity obtained by encoding the same video sequences with an H.264/AVC High Profile encoder (JM software, version 18.3) is also presented ("H.264 HP" lines). The table presents results for the sequences *RaceHorses1*, *BasketballDrillText*, *SlideShow*, *ParkScene*, and *NebutaFestival*. The rightmost column shows the average computational complexity increase of *CFG 1* and *CFG 15* over the H.264/AVC HP encoder. It is noteworthy that even the least complex configuration (*CFG 1*) in HEVC is still more complex than H.264/AVC HP (in a range from 9.1% up to 103.6%). Moreover, *CFG 15* is at least 1.6 times more complex than H.264/AVC HP, reaching an average computational complexity increase of up to 502.2% for *NebutaFestival*, which is the most complex video sequence among those listed in the table (high spatial resolution, colourful detailed texture and continuous motion activity).

Fig. 4.6 shows the increase of Y-PSNR for each encoding configuration in comparison to the reference. It is possible to notice that the image quality is roughly maintained from *CFG 4* to *CFG 10* and from *CFG 13* to *CFG 15*. Most of the image quality gains are obtained in *CFG 3*, *CFG 11* and *CFG 12*, which enable DBF, SAO and ALF, respectively. These results lead to the conclusion that the three filters have significant impact on the objective image quality in HEVC. When the three filters are enabled (*CFG 12*), the Y-PSNR is increased by up to 0.94 dB (*SlideShow* sequence). Nevertheless, despite the impact on the image quality, it is important to remark that the activation of ALF resulted in a very large bit rate increase, as presented in Fig. 4.5 (see *CFG 12*).

Further results for a wider range of QP values (22, 27, 32, and 37) are presented in Fig. 4.7 and Fig. 4.8 in terms of BD-rate and BD-PSNR, respectively. Six video sequences were used in the tests and the results were computed by taking *CFG 1* as the reference case. Both figures reveal large differences between *H.264/AVC HP* (non-connected points on the right side of the figures) and the remaining HM encoding configurations in terms of encoding efficiency. All HM configurations result in a significant decrease in the BD-rate value when more encoding tools are activated, as shown in Fig. 4.7. The *H.264/AVC HP* configuration, however, presents a BD-rate increase that varies from 23.5% to 43.3%, depending on the video sequence, in comparison to HM *CFG 1*. Similarly, BD-PSNR in Fig. 4.8 is improved when more tools are added to the baseline HM configuration. On the other hand, the *H.264/AVC HP* configuration presents BD-PSNR degradations in a range from 1.3 dB to 2.1 dB in comparison to *CFG 1*.

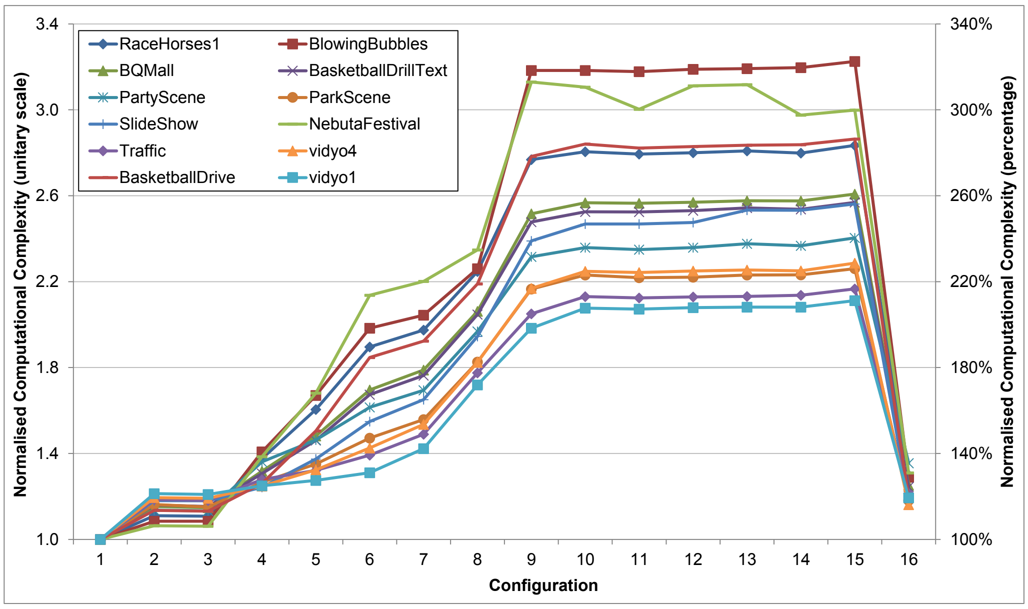


Fig. 4.4: Normalised computational complexity for encoding each video sequence under all 16 configurations (QP 32).

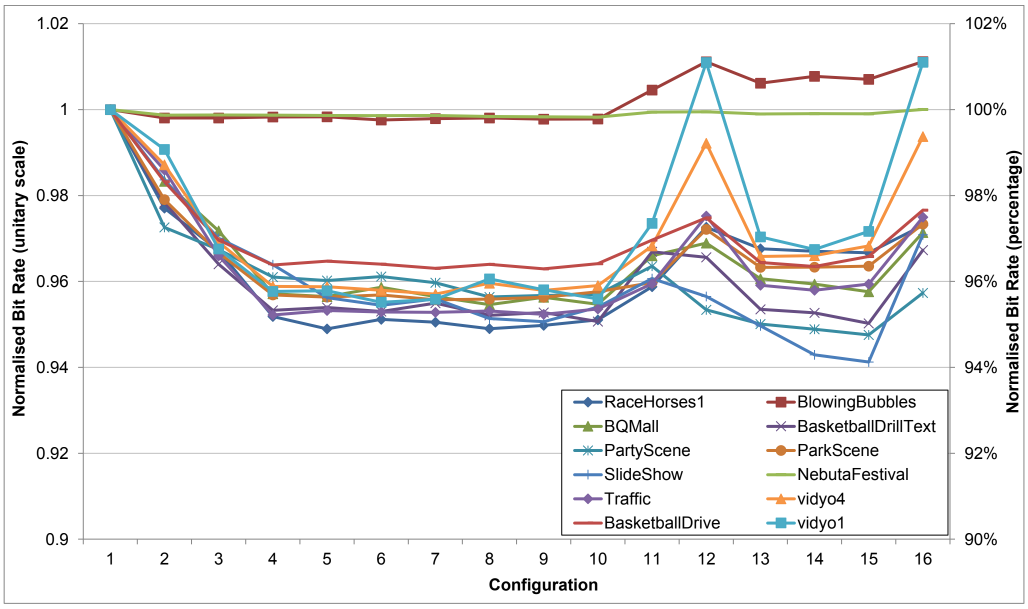


Fig. 4.5: Normalised bit rate for each video sequence encoded under all 16 configurations (QP 32).

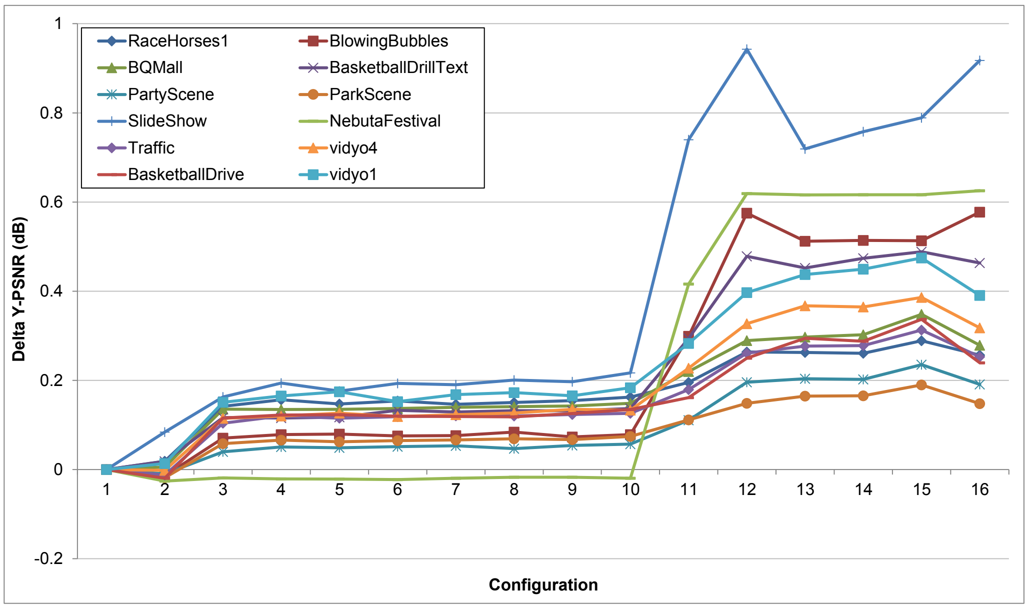


Fig. 4.6: Delta Y-PSNR for each video sequence encoded under all 16 configurations (QP 32).

A relevant conclusion of this study is that the Hadamard ME, the AMP and the filters (except for ALF, which causes a large bit rate increase) should be first enabled in a complexity-constrained encoder, since they increase significantly the encoding efficiency at the cost of a low complexity increase, as previously shown. In fact, most of the gains in the BD-PSNR and BD-rate curves (Fig. 4.7 and Fig. 4.8) are accounted for by the tools enabled in *CFG 1*-*CFG 3* and from *CFG 11* onwards.

The conclusions of this study should be taken into account when selecting the tools and functional modes to define an efficient HEVC encoder configuration. The presented results show that a careful choice of tools is necessary to achieve high encoding performance under low computational complexity levels.

To exemplify how the proposed complexity versus performance analysis can be used for tuning an HEVC encoder, an optimised configuration was created. This configuration, which appears in Table 4.2 labelled as *CFG 16*, does not use any tool or functional mode that is not advantageous from the point of view of complexity versus performance. More specifically, in *CFG 16* only the three first parameters listed in Table 4.2 are enabled, which are those that presented the highest relative coding efficiency-complexity gains in the configurations of Table 4.1 (i.e., the ratio between the bit rate reduction and the increase in encoding computational). These parameters are the Hadamard ME, the DBF and the AMP, respectively. Besides them, the SAO filter was also enabled in *CFG 16* because, although it was not classified at the top of the ranking, it yielded significant increases in terms of image quality (Y-PSNR), as shown in Fig. 4.6 and Fig. 4.8 (*CFG 11*). The remaining ME functional modes, the IBDI, the *LM Intra Prediction* and the use of NSQT were all either disabled or enabled in their lowest complexity functional mode, as shown in the last column of Table 4.2.

Table 4.3: Computational complexities (in seconds) for HEVC and H.264/AVC configurations under QPs 22, 27, 32, 37, and 42

| **Video (resolution)** | **Encoder (Conf.)** | **QP 22** | **QP 27** | **QP 32** | **QP 37** | **QP 42** | **Average Incr. (%)** |
|---|---|---|---|---|---|---|---|
| *RaceHorses1* (416x240) | HEVC (CFG 1) | 839 | 740 | 645 | 576 | 1,306 | 18.6 |
| | HEVC (CFG 15) | 2,293 | 1,969 | 1,828 | 1,599 | 447 | 224.5 |
| | H.264 HP | 693 | 603 | 534 | 484 | 1,628 | – |
| *BasketballDrill Text* (832x480) | HEVC (CFG 1) | 2,493 | 2,226 | 2,023 | 1,870 | 4,085 | 9.1 |
| | HEVC (CFG 15) | 6,366 | 5,768 | 5,197 | 4,724 | 1,656 | 178.4 |
| | H.264 HP | 2,197 | 1,904 | 1,845 | 1,742 | 3,541 | – |
| *SlideShow* (1280x720) | HEVC (CFG 1) | 4,264 | 4,088 | 3,940 | 3,823 | 8,663 | 46.7 |
| | HEVC (CFG 15) | 11,551 | 10,657 | 10,093 | 9,581 | 2,567 | 276.7 |
| | H.264 HP | 2,857 | 2,718 | 2,639 | 2,610 | 8,684 | – |
| *ParkScene* (1920x1080) | HEVC (CFG 1) | 12,778 | 11,199 | 10,299 | 9,676 | 19,134 | 15.1 |
| | HEVC (CFG 15) | 29,250 | 25,798 | 23,283 | 21,463 | 9,293 | 160 |
| | H.264 HP | 10,310 | 8,743 | 8,385 | 9,040 | 38,295 | – |
| *NebutaFestival* (2560x1600) | HEVC (CFG 1) | 59,418 | 57,811 | 18,065 | 49,361 | 111,126 | 103.6 |
| | HEVC (CFG 15) | 190,443 | 182,055 | 39,128 | 142,355 | 19,154 | 502.2 |
| | H.264 HP | 27,332 | 24,147 | 21,132 | 17,903 | 1,306 | – |

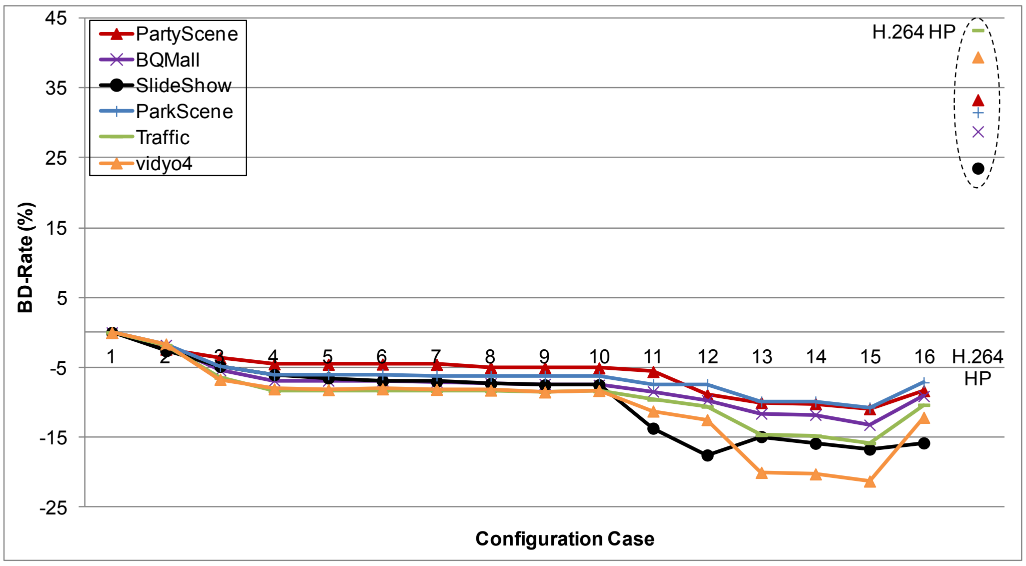


Fig. 4.7: BD-rate values for each configuration using *CFG 1* as reference.

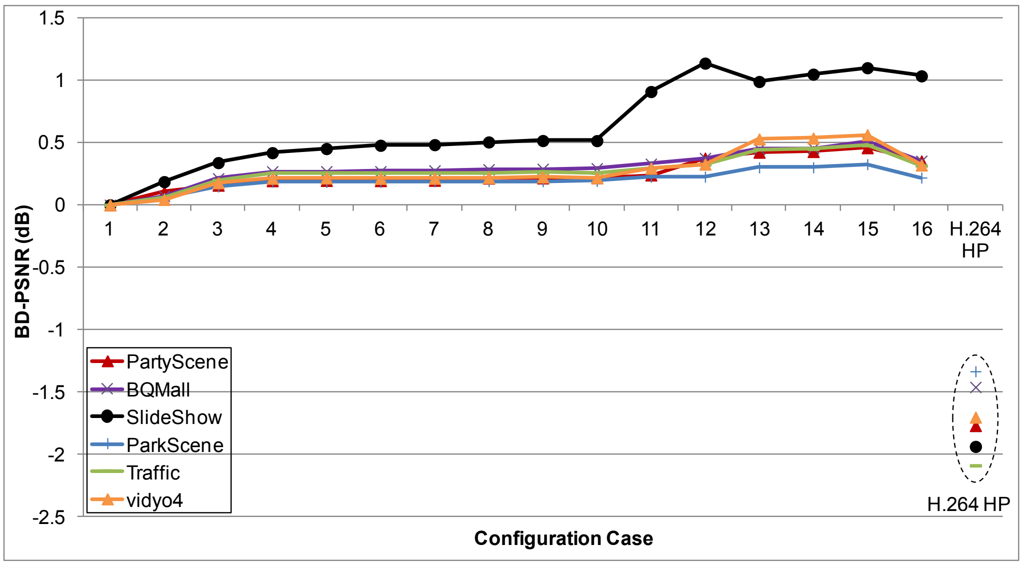


Fig. 4.8: BD-PSNR values for each encoding configuration using *CFG 1* as reference.

The results in Fig. 4.4-Fig. 4.8 show that, even though the encoding efficiency of *CFG 16* is similar to that achieved by high-complexity configurations, its computational complexity is much smaller. For instance, *CFG 16* provides roughly the same coding efficiency as *CFG 12* but its complexity is up to 2.5 times smaller (see Fig. 4.4). This optimisation exercise shows that a wise selection of coding parameters can be used to define a low complexity configuration which is capable of achieving roughly the same coding efficiency as a more complex one. In other words, higher levels of compression efficiency are not necessarily obtained using the most complex coding configurations.

### 4.1.5 Performance and Complexity as a Function of QP

This section presents an analysis of the encoding performance and computational complexity for different bit rates. In order to decouple the results from the effects of rate control algorithms, a set of fixed QP values (22, 27, 32, 37) was used in each experiment.

Fig. 4.9 shows the R-D encoding efficiency of the HM encoder under different configurations for the *BQMall*, *vidyo4*, *ParkScene* and *Traffic* video sequences. The H.264/AVC HP encoder was also included in this evaluation to provide an anchor for comparison. Even though all HM configurations presented in section 4.1.3 were analysed, only four of them (1, 12, 15 and 16) are presented in the charts of Fig. 4.9 for clarity. The results confirm that the encoding efficiency of HM configurations is much higher than the encoding efficiency of H.264/AVC HP, since the bit rates are reduced by approximately 50% while maintaining roughly the same Y-PSNR. As *CFG 1* and *CFG 15* are the configurations which present the worst and the best R-D efficiency results, respectively, the performance curves for the remaining HM configurations fall between those of *CFG 1* and *CFG 15*. For this reason, they are omitted from the charts in Fig. 4.9 for clarity. The curves corresponding to *CFG 12* and the optimised configuration *CFG 16* are overlapped in all charts of Fig. 4.9, confirming that their R-D efficiency is approximately the same for all tested QPs.

Fig. 4.10 shows the encoding time as a function of QP for the same video sequences and the same configurations presented in Fig. 4.9. Fig. 4.10 shows that, although *CFG 16* achieves R-D efficiency close to that of *CFG 12* in Fig. 4.9, its computational complexity is much more similar to that of the baseline configuration (*CFG 1*) for all QPs and all video sequences analysed. In fact, the computational complexity of *CFG 16* is closer to that observed in the H.264/AVC HP encoder than to the observed in *CFG 12*, even though its R-D efficiency is almost as high as in *CFG 12*. Therefore, the previous conclusions can be extended for a wide range of bit rates, meaning that coding complexity and efficiency are not necessarily correlated. Thus, in complexity-constrained encoder implementations it is worthwhile to take these findings into account.

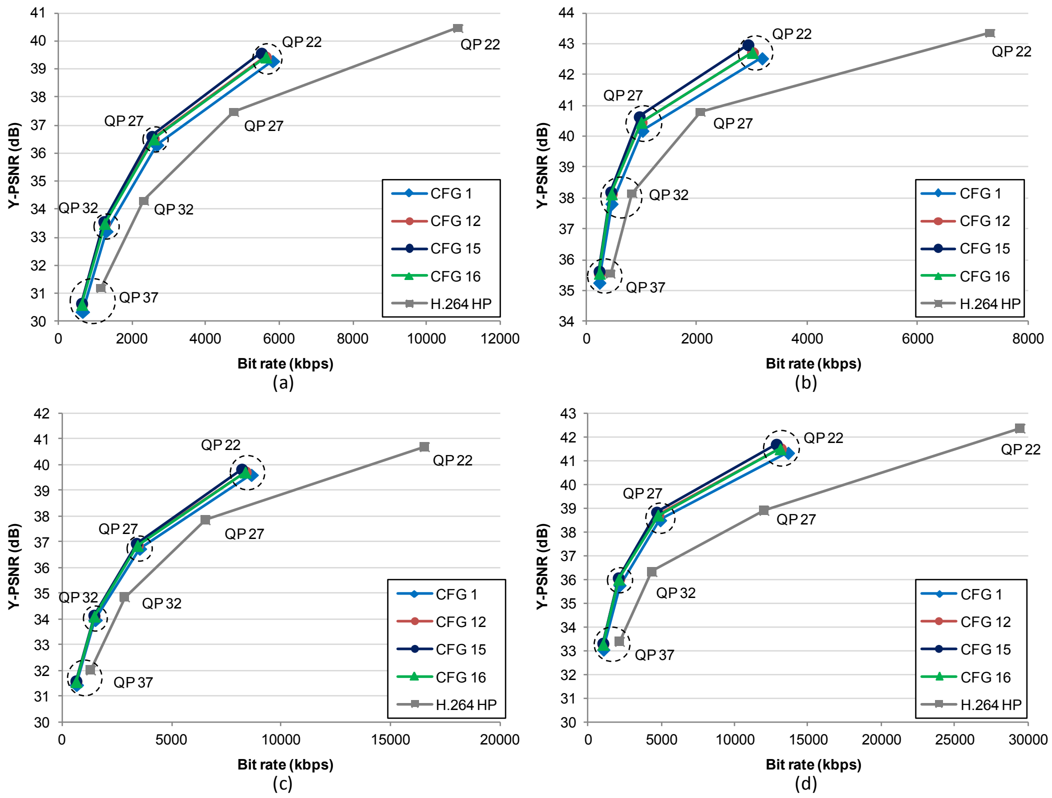


Fig. 4.9: R-D efficiency of HEVC with *CFG 1*, *CFG 12*, *CFG 15*, and *CFG 16* and H.264 HP for the (a) *BQMall*, (b) *vidyo4*, (c) *ParkScene* and (d) *Traffic* videos (QPs 22, 27, 32, 37).

Concerning the effect of QP on the encoding complexity, it was further observed from Fig. 4.10 that the overall complexity decreases slightly when the QP increases. This effect is more prominent in the cases with larger complexity values, which allow such difference to be more easily noticed (*CFG 12* and *CFG 15*). This effect might result from the smaller amount of processed data in the case of higher QP (e.g., more zero coefficients).

## 4.1.6 Experiment Conclusions

The trade-off between computational complexity and encoding performance of the HEVC encoder was evaluated in the previous sections using a broad range of encoding configuration cases over a wide variety of video contents. The experimental study analysis shows that maximum HEVC complexity can be reduced at practically no coding efficiency cost, if the coding tools are wisely chosen, combined and configured.

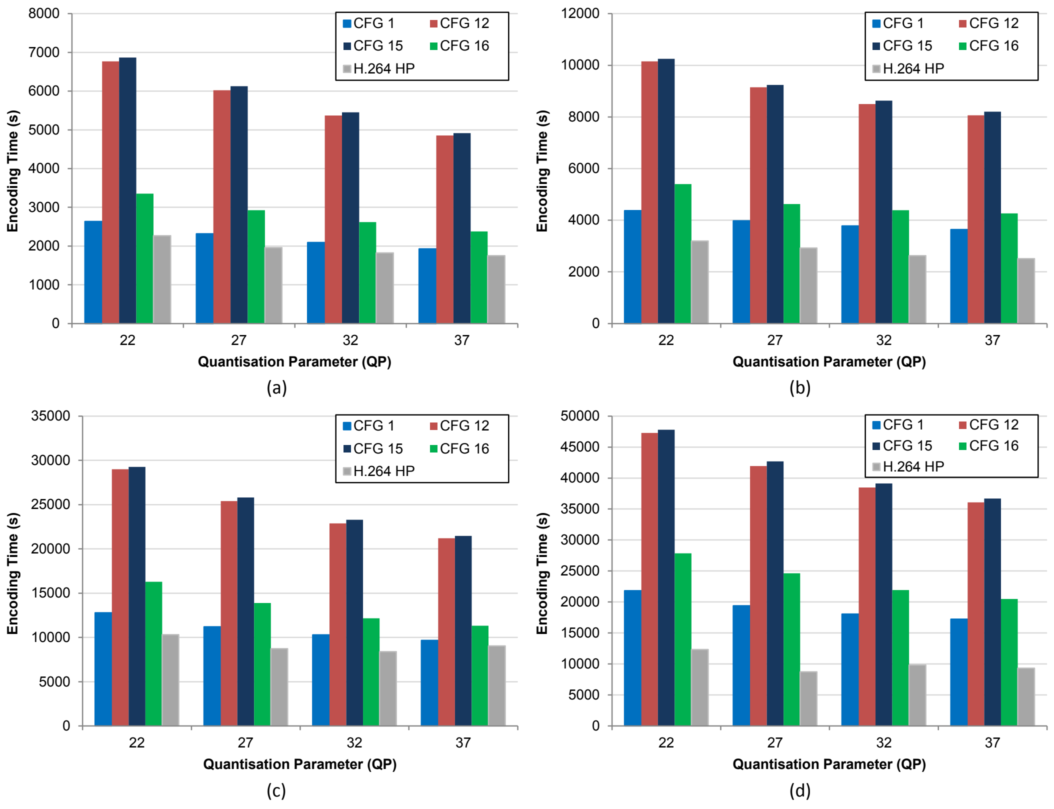


Fig. 4.10: Encoding time of HEVC with *CFG 1*, *CFG 12*, *CFG 15*, and *CFG 16* and H.264 HP for the (a) *BQMall*, (b) *vidyo4*, (c) *ParkScene* and (d) *Traffic* videos (QPs 22, 27, 32, 37).

The experimental results show that when the number of tools and functional modes increase in a cumulative progression, the computational complexity grows in a similar way, even though the encoding performance does not increase at the same pace. Therefore, it is advisable to enable first those tools which provide most gains for the least cost. Such a strategy for enabling tools and choosing encoding parameter values lead to a good trade-off between computational complexity and encoding efficiency, making it possible to achieve high encoding performance while still reducing the computational complexity in comparison to the case in which all tools are enabled.

The results demonstrate that a good trade-off between coding efficiency and computational complexity can be achieved by enabling the Hadamard ME, the AMP, and the use of filters (DBF and SAO), instead of enhancing the performance of other computationally demanding but not so efficient tools. Our conclusions about the usefulness of some tools such as *Inter 4×4*, *LM Intra Prediction*, ALF, and NSQT were confirmed later when JCT-VC removed them from HM versions later than 7. Our experiments performed with HM7 had shown that these tools should be disabled (see our proposed *CFG 16* in Table 4.2), since they did not offer significant encoding efficiency when considering their associated computational complexity increases.

The study presented in this section provided an important basis to devise the R-D-C optimised control system presented in chapter 7 of this thesis. As it will explained later on, an R-D-C analysis was performed on a set of encoding configurations, which varied the value of those parameters identified in the previous sections as those that most affect the computational complexity of an HEVC encoder. As shown in Fig. 4.4, the largest computational complexity increases are observed in the transition from *CFG 1* to *CFG 2* and from *CFG 4* to *CFG 9*, which correspond to the *Hadamard ME*, the AMP, the *Search Range*, the *Bi-prediction Refinement*, the *Inter 4×4* and the *Fast Encoding* parameters. For this reason, some of them were selected for the analysis that led to the development of an encoding time control system presented later on in chapter 7.

## 4.2 Analysis of HEVC Frame Partitioning Structures

As presented in chapter 3, complexity-aware solutions for HEVC have focused mainly on the decision process of frame partitioning structures such as CUs, PUs and TUs. However, a systematic analysis on the impact of constraining the decision of such structures in both the R-D efficiency and the encoding computational complexity is still missing. The analysis presented in this section was important for identifying research directions and lead the studies presented in this thesis to focus mainly on those partitioning structures that would most probably yield the largest complexity reductions and the best trade-offs between encoding efficiency and computational complexity when constrained.

### 4.2.1 Experimental Setup and Methodology

The experiments were performed using a setup similar to that described in section 4.1, using the same 12 test video sequences presented in section 4.1.1 and the HM encoder – version 13 (HM13) [30]. As before, the *Microsoft Visual Studio C++ Compiler* was used to compile the encoder with the *Release* mode and the tests were run on *Intel® Xeon® E5520* (2.27 GHz) processors running the *Windows Server 2008 HPC* operating system. The computational complexity was measured in terms of processing time, reported by the *Intel® VTune™ Amplifier XE 2011* software profiler [124]. The experimental study was carried out by changing the configuration of the frame partitioning structures, one parameter at a time, such that the impact of each one could be independently analysed. Every test was performed using five different QPs: 22, 27, 32, 37, and 42.

### 4.2.2 Frame Partitioning Configurations Tested

As explained in section 2.6.1.2, the main frame partitioning structures of HEVC are CUs, PUs and TUs, which size and format can vary broadly following a quadtree-structured partitioning scheme. There are three frame partitioning parameters in the HEVC encoder that can control directly these structures: *Max CU Depth*, *Max TU Depth* and *AMP*, which define, respectively, the maximum quadtree depth allowed for a CU in each CTU, the maximum quadtree depth allowed for a TU in each RQT, and the possibility of using *Asymmetric Motion Partitions* in the PU format decision.

Starting with a baseline encoder configuration using the *Random Access*[1] temporal configuration, new configurations were created by modifying the value of each partitioning parameter, one at a time in a non-cumulative way (i.e., a parameter was changed in one configuration and then set to its original value in the following configuration). The resulting image quality, bit rate and encoding computational complexity were recorded for comparison with the reference baseline configuration.

The seven configurations tested are presented in Table 4.4. The baseline encoder configuration is defined as *PAR 1*, while the other six configurations correspond to *PAR 2* through *PAR 7*. In *PAR 1*, the maximum CU depth allowed is 4 (*Max CU Depth*), the maximum TU depth allowed is 3 (*Max TU Depth*) and the use of AMP is enabled. From *PAR 2* to *PAR 4*, the only parameter changed is *Max CU Depth*. In *PAR 5* and *PAR 6*, the effect of changing only the *Max TU Depth* parameter is tested. Finally, in *PAR 7* the activation of AMP is evaluated. The values *D* and *E* represent the disabled and enabled states of AMP, respectively.

[1] During the development of the research work presented in this thesis, it was noticed that most works published in the literature presented results for the *Random Access* temporal configuration. For this reason, aiming at providing comparable results with these related works, at some point of this work the *Low Delay* configuration was not used anymore and the *Random Access* configuration was used for all tests.

All test sequences listed in section 4.1.1 were encoded with the seven configurations presented in Table 4.4. For each simulation, bit rate, Y-PSNR and complexity results were recorded in order to allow the performance and complexity trade-off analysis presented in the next section.

Table 4.4: Frame partitioning structure configurations tested in the experiments.

| **Parameter** | **Frame Partitioning Configuration (PAR)** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| *Max CU Depth* | 4 | **3** | **2** | **1** | 4 | 4 | 4 |
| *Max TU Depth* | 3 | 3 | 3 | 3 | **2** | **1** | 3 |
| *AMP* | E | E | E | E | E | E | **D** |

## 4.2.3 Encoding Performance and Complexity Trade-off Analysis

This section presents the performance results in terms of PSNR, bit rate and computational complexity for the seven test cases listed in Table 4.4. The results are summarised in Table 4.7 and Fig. 4.11-Fig. 4.15. Even though Fig. 4.11-Fig. 4.13 present results for QP 32, other QPs were also tested and have shown similar behaviour. General results for QPs 22, 27, 32 and 37 are shown in Fig. 4.14-Fig. 4.15.

Results in terms of computational complexity for all video sequences encoded under all tested configurations are plotted in Fig. 4.11, normalised with respect to *PAR 1*. In Fig. 4.11, all video sequences exhibit a similar variation in the encoding complexity from *PAR 1* to *PAR 7.* The largest complexity decreases are noticed from *PAR 1* to *PAR 4*, which represent the cases when the value of *Max CU Depth* was modified. In *PAR 5* and *PAR 6* the computational complexity has also decreased in comparison to *PAR 1*, even though in a much smaller extent. Finally, with *PAR 7* the encoding computational complexity experiences a small decrease varying from 1% to 19% in comparison to *PAR 1*. It is also noticeable that, differently from the first six configurations, the curves in *PAR 7* are more distant from each other. This happens due to the fact that this configuration affects directly the ME operation and as the video sequences are quite distinct in terms of motion activity, they result in different encoder behaviour when AMP is disabled. A similar effect has been previously noticed in Fig. 4.4.

Normalised bit rate results are presented in Fig. 4.12. It is noticeable that *PAR 2*, *PAR 3* and *PAR 4*, which are also those configurations which resulted in the largest decreases in computational complexity, are also responsible for the largest bit rate increases. However, even though the computational complexity is affected in a similar way for all video sequences in such cases, the compression efficiency varies differently from one video to another, depending on its characteristics. This happens because when *Max CU Depth* is decreased the number of tested coding tree possibilities decreases exponentially in all video sequences, but the effect on the compression efficiency is much more noticeable in low-resolution video sequences, which use smaller CU sizes in the encoding process, than in high-resolution video sequences, which are generally encoded with larger CU sizes. This is visible in Fig. 4.12, which shows that the largest bit rate increases appear for the *SlideShow* (1280×720), *RaceHorses1* (416×240), and *BQMall* (832×480) sequences, while the smallest increases appear for *NebutaFestival* (2560×1600), *ParkScene* (1920×1080) and *Traffic* (2560×1600) sequences. The bit rate remains practically the same in *PAR 5*, *PAR 6* and *PAR 7*, in comparison to *PAR 1*.

Fig. 4.13 shows the image quality variation in terms of Y-PSNR for each configuration in comparison to *PAR 1*. It is possible to notice that the quality is maintained almost unchanged from *PAR 5* to *PAR 7*. Most of the image quality drops are noticed in *PAR 2*, *PAR 3* and *PAR 4*, respectively, due to the same reasons explained for Fig. 4.12.

Further results in terms of BD-rate and BD-PSNR are presented in Fig. 4.14 and Fig. 4.15, respectively. The BD values were computed using QPs 22, 27, 32, and 37. Six video sequences were used in the tests and the results were computed by taking *PAR 1* as the reference case. Confirming the results discussed previously for QP 32, the configurations *PAR 2*, *PAR 3* and *PAR 4* are those which reveal the largest decreases in compression efficiency in comparison to *PAR 1*, while *PAR 5*, *PAR 6* and *PAR 7* maintained BD-rate and BD-PSNR results close to zero. In fact, when *Max CU Depth* is set to 1 (*PAR 4*), the BD-rate increases reach values above 24% and up to 52% in the worst case, which shows that the indiscriminate decrease of *Max CU Depth* is an extremely inefficient way of reducing the computational complexity of HEVC encoders.

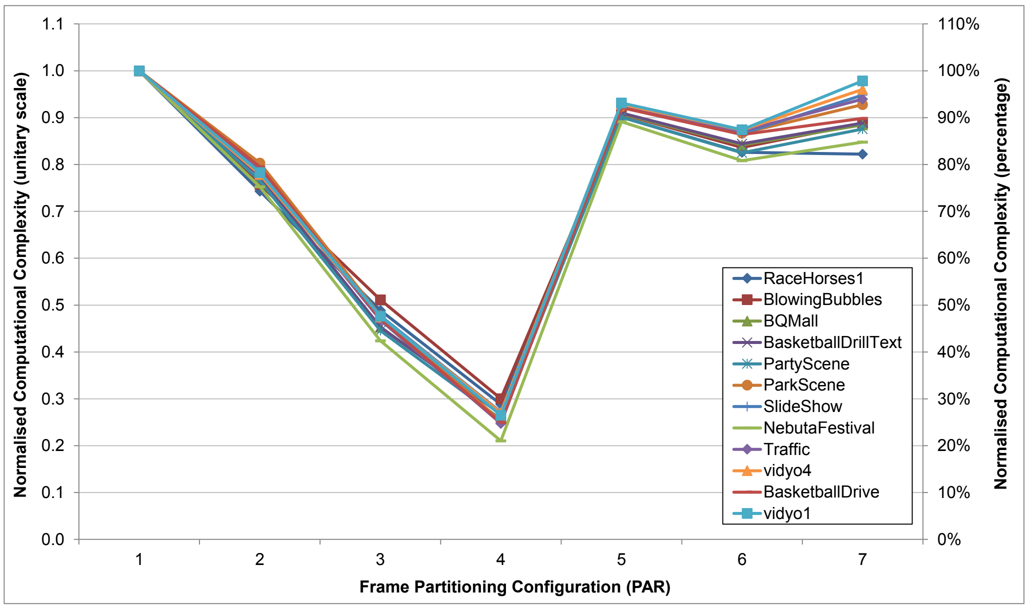


Fig. 4.11: Normalised computational complexity for encoding each video sequence under the seven frame partitioning configurations (QP 32).

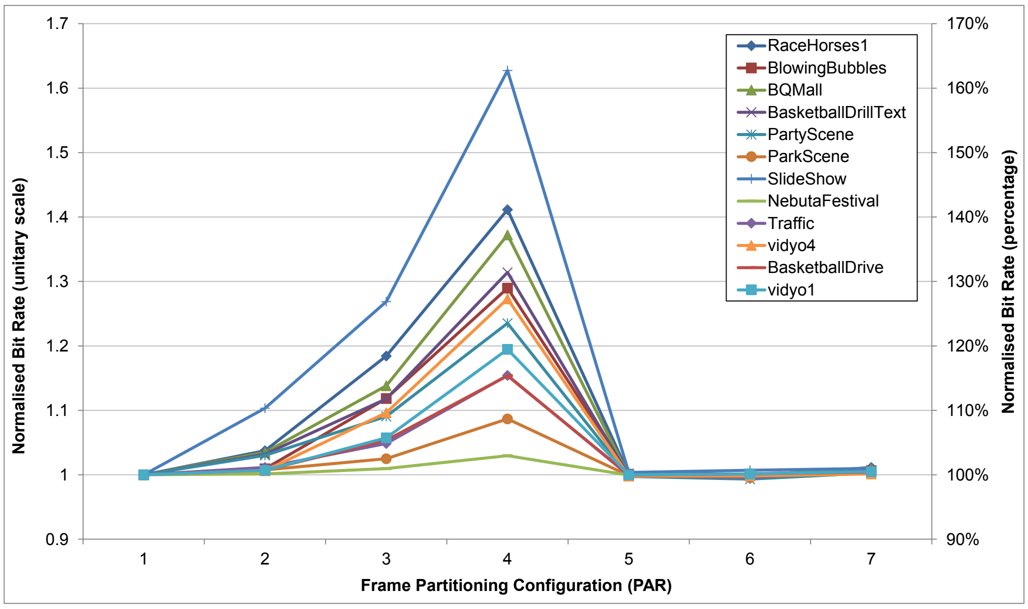


Fig. 4.12: Normalised bit rate for each video sequence encoded under the seven frame partitioning configurations (QP 32).

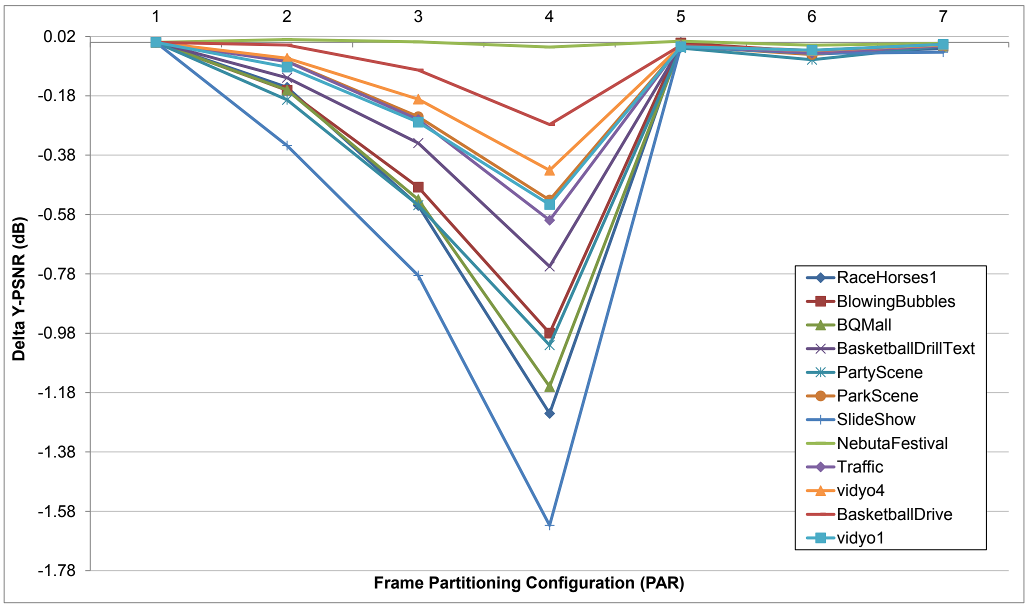


Fig. 4.13: Delta Y-PSNR for each video sequence encoded under the seven frame partitioning configurations (QP 32).

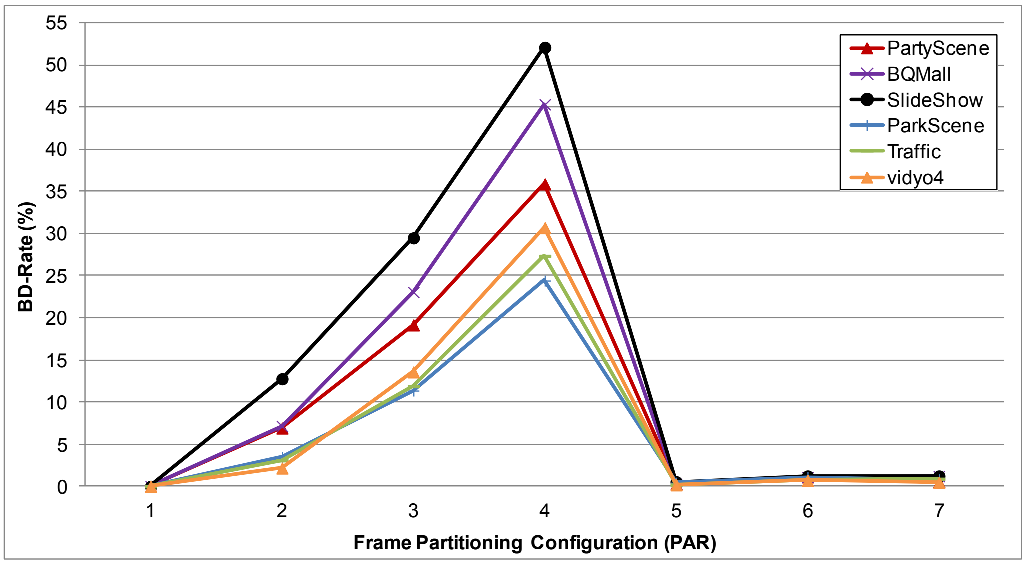


Fig. 4.14: BD-rate values for each configuration using *PAR 1* as reference.

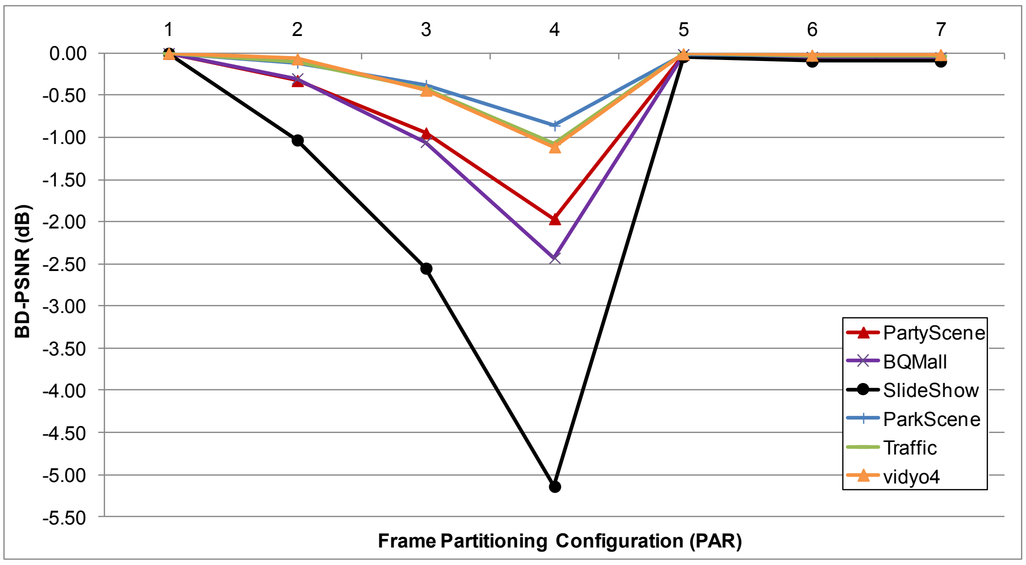


Fig. 4.15: BD-PSNR values for each configuration using *PAR 1* as reference.

Table 4.5 and Table 4.6 present results in terms of BD-rate increase and average computational complexity decrease per configuration tested, while Table 4.7 shows the ratio between the two values presented in Table 4.5 and Table 4.6. On the one hand, according to Table 4.7 the best trade-off between computational complexity reduction and compression efficiency would be achieved by first using *PAR 5*, *PAR 6* and *PAR 7* and, as last resource, *PAR 2*, *PAR 3* and *PAR 4*. On the other hand, choosing *PAR 5*, *PAR 6* or *PAR 7* would allow a maximum computational complexity decrease of only 15.3%, which is not sufficient when the large computational complexity of HEVC is taken into account.

Table 4.5: BD-rate increase (%) per configuration.

| **Video Sequence** | **Frame Partitioning Configuration (PAR)** | | | | | |
|---|---|---|---|---|---|---|
| | **2** | **3** | **4** | **5** | **6** | **7** |
| *PartyScene* | 6.9 | 19.1 | 35.9 | 0.3 | 1.1 | 0.5 |
| *BQMall* | 7.2 | 23.0 | 45.3 | 0.3 | 1.1 | 1.2 |
| *SlideShow* | 12.8 | 29.5 | 52.1 | 0.5 | 1.2 | 1.2 |
| *ParkScene* | 3.5 | 11.4 | 24.4 | 0.4 | 1.1 | 0.8 |
| *Traffic* | 3.1 | 11.9 | 27.3 | 0.2 | 0.7 | 0.9 |
| *vidyo4* | 2.2 | 13.6 | 30.7 | 0.2 | 0.7 | 0.5 |
| **Average** | **5.9** | **18.1** | **36.0** | **0.3** | **1.0** | **0.9** |

Table 4.6: Average computational complexity reduction (%) per configuration.

| **Video Sequence** | **Frame Partitioning Configuration (PAR)** | | | | | |
|---|---|---|---|---|---|---|
| | **2** | **3** | **4** | **5** | **6** | **7** |
| *PartyScene* | 23.4 | 54.7 | 74.30 | 9.9 | 17.6 | 9.7 |
| *BQMall* | 23.2 | 55.1 | 73.4 | 10.0 | 17.1 | 11.8 |
| *SlideShow* | 22.4 | 52.6 | 73.1 | 7.8 | 13.8 | 5.2 |
| *ParkScene* | 20.1 | 52.5 | 75.0 | 8.5 | 15.0 | 8.5 |
| *Traffic* | 22.1 | 53.7 | 75.6 | 8.3 | 14.7 | 7.4 |
| *vidyo4* | 22.2 | 52.7 | 73.1 | 7.6 | 13.6 | 5.1 |
| **Average** | **22.2** | **53.5** | **74.1** | **8.7** | **15.3** | **8.0** |

Table 4.7: Ratio between BD-rate increase (%) and computational complexity reduction.

| **Video Sequence** | **Frame Partitioning Configuration (PAR)** | | | | | |
|---|---|---|---|---|---|---|
| | **2** | **3** | **4** | **5** | **6** | **7** |
| *PartyScene* | 29.5 | 35.0 | 48.3 | 2.8 | 6.3 | 5.5 |
| *BQMall* | 30.9 | 41.8 | 61.7 | 2.9 | 6.4 | 10.2 |
| *SlideShow* | 57.1 | 56.1 | 71.3 | 6.9 | 8.9 | 23.7 |
| *ParkScene* | 17.2 | 21.6 | 32.6 | 5.1 | 7.4 | 9.7 |
| *Traffic* | 14.0 | 22.2 | 36.2 | 2.4 | 5.0 | 12.1 |
| *vidyo4* | 9.8 | 25.7 | 42.0 | 2.2 | 5.2 | 9.3 |
| **Average** | **26.4** | **33.7** | **48.7** | **3.7** | **6.5** | **11.8** |

### 4.2.4 Experiment Conclusions

The previous section presented an analysis on the trade-off between computational complexity and encoding performance of HEVC when different constraints are applied to the decision of the frame partitioning structures. Seven configurations and 12 video sequences were used in the tests. The analysis shows that the encoding computational complexity can be thoroughly reduced by managing the frame partitioning structures, but some configurations incur in much larger costs in terms of compression efficiency than others.

By changing the maximum RQT depth allowed and the possibility of using AMP (i.e., *PAR 5*, *PAR 6*, *PAR 7*), the HEVC encoding complexity can be reduced at very small compression efficiency costs. The costs of modifying such structures varied between 3.7 and 11.8, while the costs of decreasing complexity by changing the maximum CU depth (i.e., *PAR 2*, *PAR 3*, *PAR 4*) varied between 26.4 and 48.7, as presented in Table 4.7.

However, as previously noticed, the computational complexity decrease achieved when changing the first structures mentioned varied between 8.0% and 15.3%, which is still modest when considering the enormous computational complexity of HEVC (up to 502% more complex than H.264/AVC HP, as we have shown in Table 4.3).

In order to achieve larger complexity reductions levels minimising the compression efficiency loss, the encoder should be able to choose wisely their parameters according to the video characteristics, so that a positive trade-off between computational complexity and compression efficiency is achieved. Instead of simply removing indiscriminately the possibility of using certain frame partitioning structures in the whole video sequence, as done in the experiments described in the previous sections, the encoder must be able to decide the constrained structure at a finer scale, such as in a per-frame or per-CU frequency, in response to the time and space varying characteristics of the video signal.

The next chapters of this thesis propose methods for efficiently reducing and scaling the computational complexity of the HEVC encoder that possess these desirable adaptation properties.

# Chapter 5

# Computational Complexity Scaling Using Adaptive Frame Partitioning Structures

This chapter presents a set of methods which were developed with the goal of dynamically scaling the computational complexity required by HEVC in the encoding process. All of them rely on constraining the frame partitioning structures introduced by the standard, namely the CUs and the PUs, in order to adjust the number of R-D evaluations performed in the optimisation process and consequently the encoding computational complexity[2].

By using the proposed algorithms, the encoder computational complexity can be downscaled by up to 50% with negligible R-D performance losses and down to 20% of the unconstrained complexity with larger losses[3]. The levels of performance degradation observed when applying these complexity scaling methods are acceptable in many applications and in power constrained devices where some sort of encoding complexity reduction methods have to be applied. Part of the methods and results presented in this chapter were published in [11-18].

[2] The experiments presented in this chapter were performed using HM versions from 2 to 9. However, since the results for each proposed algorithm are compared against the unmodified HEVC encoder of the same version, the differences between encoder versions can be disregarded. Besides, the frame partitioning structures did not suffer significant modifications during the standardisation process.

[3] As in chapter 4, computational complexity was measured in terms of processing time by the *Intel® VTune™ Amplifier XE 2011* software profiler [124], which was executed on the HM software compiled for a 64-bit architecture running on a 2.27 GHz processor.

## 5.1 Introduction and Motivation

The study carried out in chapter 4 has shown that the computational complexity of HEVC is a direct consequence of the frame partitioning structures that lead to nested encoding loops, such that the encoding of CUs at deeper Coding Tree depths is a process invoked during the encoding of CUs at lower depths. For each CU in each possible Coding Tree configuration, all possible PU divisions and all possible RQT formations are tested in an RDO process, which considers every encoding possibility and compares all of them in terms of R-D efficiency. As the R-D cost for each encoding possibility is computed only after intra/inter-prediction, direct and inverse transform and quantisation, entropy coding and filtering operations, the complexity of defining the best combination of CU, PU and TU structures is the bottleneck of the HEVC encoder.

Chapter 4 has also shown that CUs are the most suitable frame partitioning structures to be adjusted when large amounts of computational complexity reductions are sought, even though with potentially large costs in terms of compression efficiency reduction. Fig. 5.1 shows the distribution of average computational complexity of CU encoding as a function of tree depth for HM – version 4 (HM4). Coding Tree depths are presented in the y-axis and the computational complexity is presented in the x-axis. The figure reflects explicitly the nested nature of the HEVC encoding process, with CUs at higher tree depths encoded inside CUs at smaller tree depths. For each CU depth (y-axis), the computational complexity is divided into three components: (a) the complexity of performing inter-frame prediction for CUs at the current depth (in grey); (b) the complexity of coding the same image area as CUs at a higher depth (in white); and (c) the complexity of other operations (in black).

Due to this nested encoding structure, the percentage of computational complexity for encoding CUs at the first tree depth indicated as 100% in Fig. 5.1, is divided into the complexity of coding CUs at the second tree depth (84.5%, in white), the complexity of inter prediction for 64×64 CUs (*IP 64×64*, 14.6%, in grey) and the complexity of other small operations (0.9 %, in black). Similarly, the complexity of encoding CUs in the second tree depth includes the complexity of encoding CUs at the third tree depth (in white), the complexity of inter prediction for 32×32 CUs (*IP 32×32*, in grey) and the complexity of other various operations (in black). The same structure applies to the third depth. Finally, for CU depth = 4, the complexity includes only the inter prediction for 8×8 CUs (*IP 8×8*, in grey) and the complexity associated with other small operations (in black), since no further tree depths are allowed.

Notice that although the smallest CU depths show the largest total percentages of computational complexity in Fig. 5.1, the highest CU depths are the actual responsible for most of the encoding computational complexity (*IP 8×8*). For example, although compressing CUs in the third depth (composed by 16×16 CUs) represents 68.3% of the overall computational complexity, in fact only 17.4% is the real complexity associated to this CU size, whereas almost all the remaining complexity (49.5%) is dedicated to inter prediction for 8×8 CUs (*IP 8×8*) and other operations in the fourth depth.

The behaviour of CU coding along the temporal domain was also investigated. Based on the concept of temporal stationarity (also called *stillness*), defined as the tendency of a video sequence to comprise large image areas with either no or low motion between frames, an experimental study was carried out in order to characterise coding tree depth variations in co-localised areas of neighbouring frames. Fig. 5.2 represents the maximum Coding Tree depth variation used in random co-localised 64×64 areas for 50 frames of the (a) *BQTerrace*, (b) *BasketballDrive* and (c) *Cactus* video sequences, which were encoded with QPs 27, 32 and 37 in HM4, respectively. The figure shows that the maximum Coding Tree depth does not change very often in the video sequences, which means that once a depth is used in a determined area of the video, the same depth tends to be used in co-localised areas of temporally adjacent frames before it changes to a different value. Based on this observation, the first methods proposed in this chapter were developed assuming that maintaining the maximum Coding Tree depth for a relatively long period and skipping all the R-D tests at deeper tree levels would incur in a small decrease in terms of R-D performance.

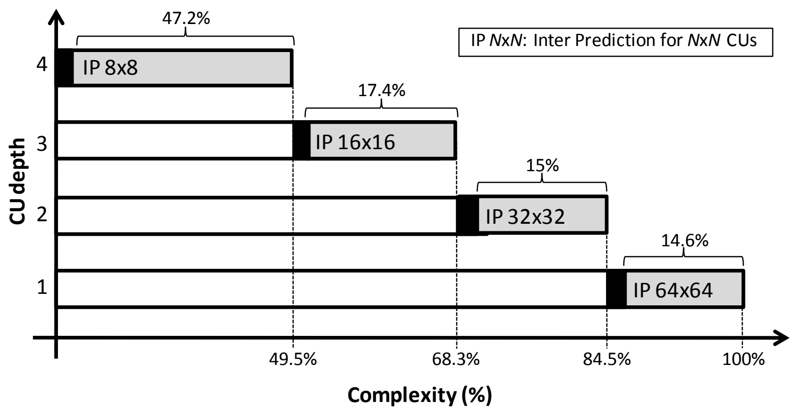
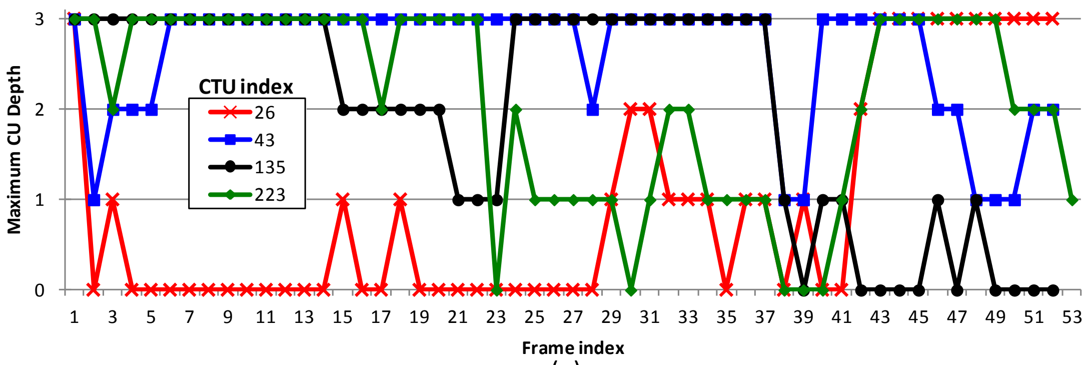
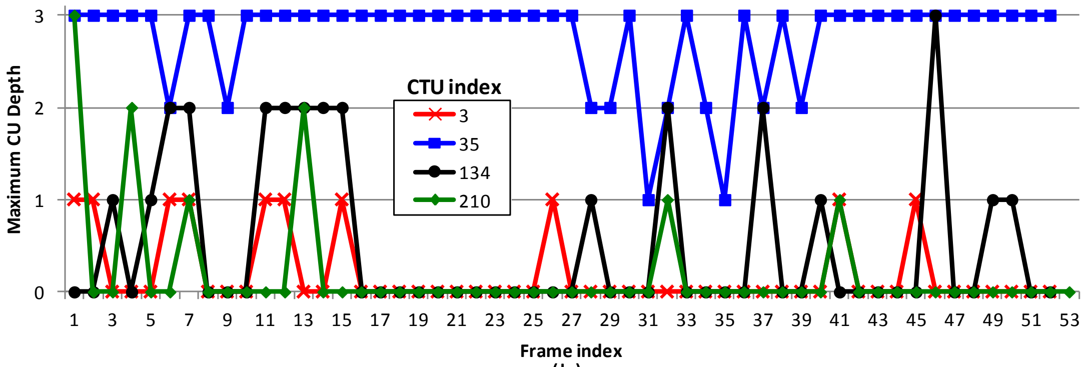


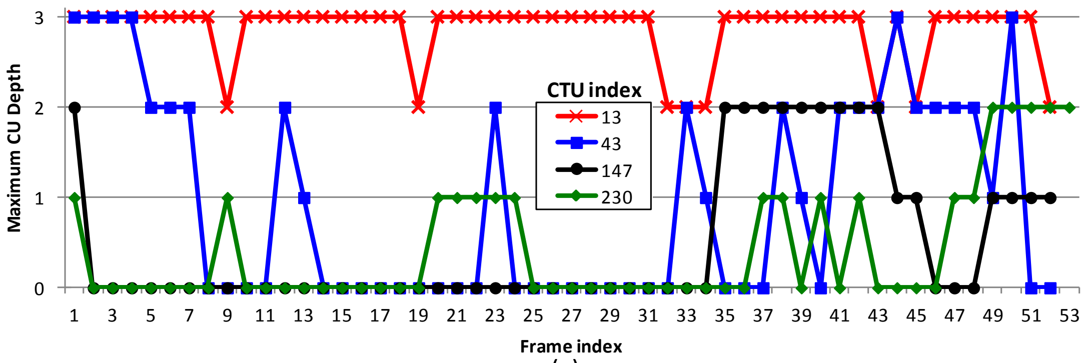
Fig. 5.1: Computational complexity of encoding CUs in each Coding Tree depth.



(a)



(b)



(c)

Fig. 5.2: Maximum Coding Tree depth of random co-localised 64×64 areas for sequences (a) *BQTerrace* (QP 27), (b) *BasketballDrive* (QP 32), and (c) *Cactus* (QP 37).

## 5.2 Fixed Depth Complexity Scaling

The Fixed Depth Complexity Scaling (FDCS) method [13] is proposed here as the simplest way of adjusting the maximum Coding Tree depth used in a frame according to the system's computational limitations. Initially, the encoding process is performed normally using the maximum Coding Tree depth possible for all CTUs in a frame. Then, whenever the computational complexity increases beyond an upper bound, the maximum depth allowed for CTUs in the next frame is decremented in one unit. Oppositely, when the complexity decreases to a value under the upper bound, the maximum depth allowed is incremented by one unit. The process is repeated if necessary until the minimum or the maximum depth allowed by the standard is reached or the target computational complexity is achieved.

### 5.2.1 Algorithm Overview

In this section, the method is explained in more detail with the help of the high-level diagram presented in Fig. 5.3. Initially, the video sequence is partitioned into temporal video segments composed of *N* consecutive frames. Then, the first three frames at the beginning of each video segment are encoded using the maximum possible Coding Tree depth. These are called unconstrained frames (*Fu*). The time spent to encode each *Fu* is used to estimate the maximum overall computational complexity for the video segment, as shown in (Eq. 42), where $CE^{Max}$ is the estimated maximum complexity, $CS^{Fi}$ is the computational complexity spent to encode the $i^{th}$ frame and *N* is the number of frames in the video segment to be encoded. The average complexity of the first three frames was used in the calculation of $CE^{Max}$ in order to take into account possible computational complexity variations caused by the use of a different number of reference frames in each image in a GOP.

$$CE^{Max} = \frac{N}{3}\sum_{i=1}^{3} CS^{Fi} \tag{Eq. 42}$$

The value of $CE^{Max}$ is then used to set the maximum target complexity $CE^{T}$ available to encode the whole video segment, as defined in (Eq. 43), where $\alpha^{T} \in [0\%, 100\%]$ is the complexity reduction ratio defined either as a user parameter or computed from system parameters (e.g., remaining battery life). In this case, 100% represents the maximum possible encoding complexity when no reduction is imposed.

$$CE^{T} = \alpha^{T} \cdot CE^{Max} \tag{Eq. 43}$$

After defining the target complexity $CE^{T}$, the algorithm keeps encoding *Fu* frames and maintains a record of the computational expenditure while encoding the video segment. The record is used to estimate the complexity for the whole segment ($CP^{N}$), which is calculated after encoding each frame. The $CP^{N}$ calculation assumes that the complexity of encoding the next frames in the segment is similar to the complexity of the most recently encoded frame. $CP^N$ is computed as shown in (Eq. 44), where $CS^{Fi}$ is the computational complexity spent to encode the $i$th frame of the segment, $CS^{F_{NE}}$ is the complexity spent to encode the last frame, and $NE$ is the number of frames already encoded in the video segment.

$$CP^N = \sum_{i=1}^{NE} CS^{Fi} + CS^{F_{NE}} \cdot (N - NE) \qquad \text{(Eq. 44)}$$

While encoding the video segment, if $CP^N$ is lower than the limit imposed by $CE^T$, the next frame is encoded as *Fu*. However, when $CP^N$ increases beyond the limit imposed by $CE^T$, the maximum depth allowed in the next frame (*MD*, in Fig. 5.3) is decremented in one unit (if the minimum depth has not been achieved yet). In such case, the frame is called a constrained frame (*Fc*). When $CP^N$ decreases to a value under the limit imposed by $CE^T$, the maximum depth allowed in the next frame is incremented by one unit with possible saturation at the maximum depth allowable by the standard. The maximum depth value is maintained if $CP^N$ is equal (or very close) to $CE^T$. To avoid adjustments caused by small computational complexity differences between adjacent frames (which occur naturally in video sequences due to their intrinsic changing characteristics), a difference of 2% is accepted between $CP^N$ and $CE^T$ without triggering a change in the *MD* value, as shown in Fig. 5.3. This value was defined after experimental observations.

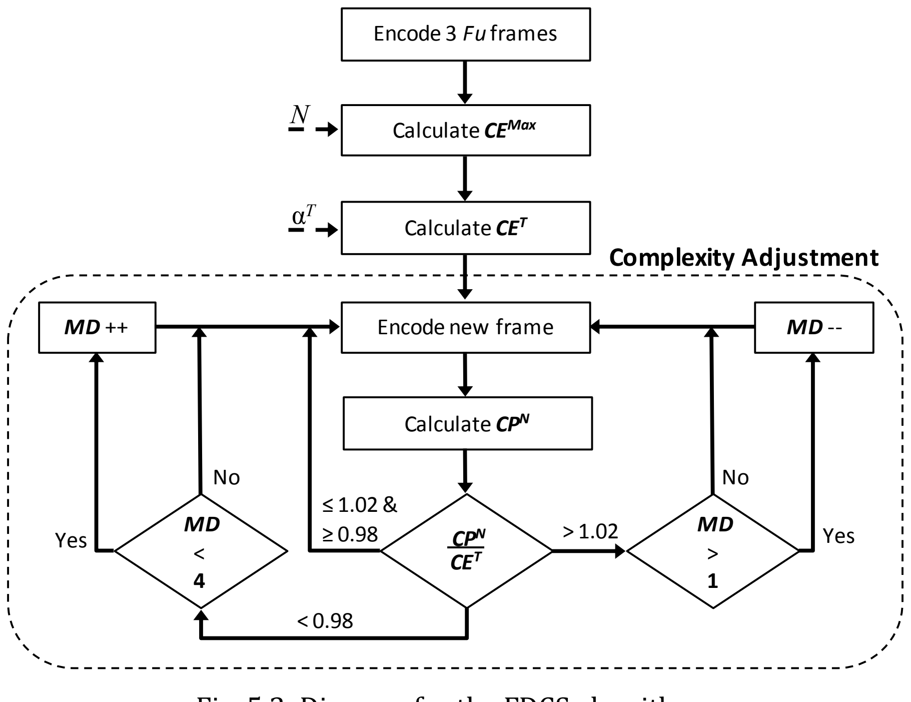


Fig. 5.3: Diagram for the FDCS algorithm.

## 5.2.2 Results for FDCS and Performance Evaluation

The FDCS method was evaluated by measuring the encoding computational complexity reduction accuracy under specific targets and its influence on the R-D performance of the HM encoder (HM4). Three video sequences comprising 500 frames (*BasketballDrive*, *BQTerrace*, *Cactus*) and a concatenation of two of them (*BasketballDrive* and *Cactus*, from now on referenced as *BasketballCactus*) were used in the experiments. The *Low Delay P* temporal configuration was used in all tests. The encoder performance was evaluated under five complexities reduction ratios (from 60% to 100% with 10% steps) and four different QP values (27, 32, 37, and 42).

The R-D efficiency as a function of the target computational complexity is shown in Fig. 5.4. Average values for bit rate (kbps) and luminance PSNR (dB) for the four test sequences are presented. Table 5.1 shows the average performance results in terms of running complexity, BD-rate and BD-PSNR for the five target complexities tested. The relative bit rate increases and the Y-PSNR reductions are calculated with reference to the maximum complexity (100%) results, which correspond to the case in which no complexity scaling is applied.

As expected, the bit rate increases when small target complexities are used. In the worst case ($\alpha^T$ set to 60%), a BD-rate increase of 12.02% was noticed. This happens due to the fact that limiting the coding tree depth to decrease the computational complexity leads to a smaller number of small-sized CUs, which results in prediction residues with more spatial structure and larger magnitudes, which are more difficult to encode, leading to higher bit rates. As small blocks are not allowed in such cases, a coarser prediction is performed, which also results in a lower image quality. The running complexity results show that the method is quite accurate and can maintain the actual complexity near the target values, with maximum variations around 3%.

The results show that FDCS is capable of adjusting the number of tested Coding Tree possibilities by increasing or decreasing the maximum Coding Tree depth used in all treeblocks belonging to a frame. Nevertheless, using a unique, fixed maximum Coding Tree depth for all treeblocks in the frame harms the R-D efficiency in some cases, especially when the target complexity is set to the lowest value tested ($\alpha^T$=60%). In such case, the actual computational complexity reduction achieved is around 37% and the BD-rate increase is 12.02%. Aiming at finding solutions to improve the R-D efficiency of this simple method, the approaches presented in the next sections are proposed.

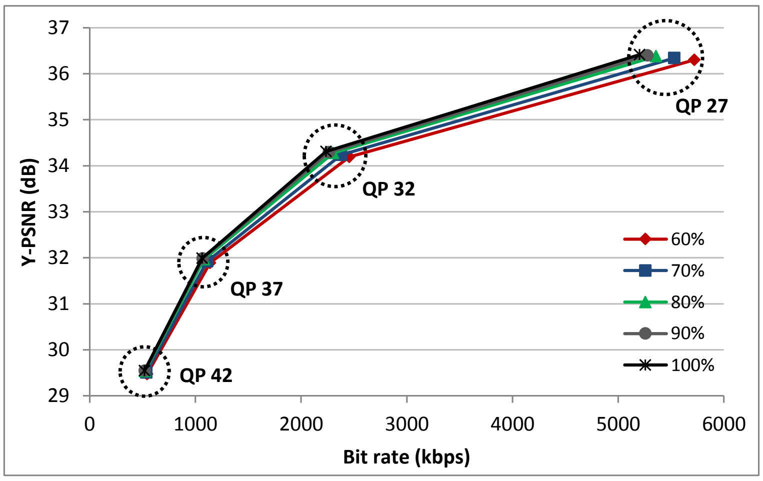


Fig. 5.4: Average R-D performance of the FDCS method.

Table 5.1: Average results obtained for the FDCS algorithm.

| **Target Complexity ($\alpha^T$)** | **Running Complexity** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|
| 100% | 100% | — | — |
| 90% | 89% | +1.24 | -0.03 |
| 80% | 77% | +3.15 | -0.09 |
| 70% | 69% | +7.59 | -0.21 |
| 60% | 63% | +12.02 | -0.33 |

## 5.3 Variable Depth Complexity Scaling

The Variable Depth Complexity Scaling (VDCS) method [11-13] is based on dynamic constraining of the maximum Coding Tree depth according to the depth used in previously encoded frames. This method is motivated by the results presented in Fig. 5.2 of section 5.1, which shows the evolution of the maximum Coding Tree depth used in random co-localised treeblocks of neighbouring frames. Based on those observations and on the strategy presented in section 5.2, a Variable Depth Complexity Scaling method was developed.

As in the FDCS method, two types of frames are used in VDCS: unconstrained (*Fu*) and constrained frames (*Fc*). *Fu* frames are also defined as those encoded through the usual process in which all possible Coding Tree structures are tested. However, differently from FDCS, in VDCS the *Fc* frames are encoded with maximum Coding Tree depths bound to those used in the most recently encoded *Fu* frame, taking advantage of the temporal stationarity characteristic of video sequences.

### 5.3.1 Algorithm Overview

As the computational complexity required to encode an *Fc* frame is smaller than that of an *Fu* frame, the number of consecutive *Fc* frames (called here as *Nc*) is dynamically adjusted as a function of the target computational complexity[4], as illustrated in the example of Fig. 5.5. Naturally, the larger the value of *Nc*, the smaller is the computational complexity required to encode the video sequence.

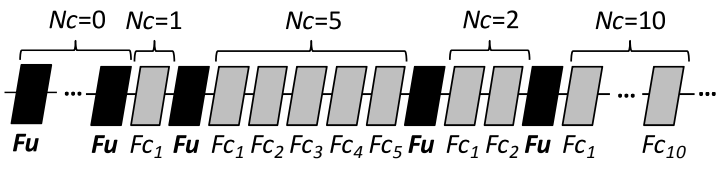


Fig. 5.5: Example of operation of the VDCS strategy.

The high-level diagram of the VDCS algorithm is presented in Fig. 5.6. The number of frames which compose the video segment (*N*) and the complexity reduction ratio ($0\% < \alpha^T < 100\%$) are the two input parameters of the algorithm. Just as in the FDCS method, in the starting phase, $CE^{Max}$ and $CE^T$ are computed according to (Eq. 42) and (Eq. 43), respectively. While $CP^N$ is within the limit imposed by $CE^T$, all frames are encoded as *Fu* (i.e., $Nc = 0$). Whenever $CP^N$ increases beyond $CE^T$, the complexity adjusting phase is activated and frames start being encoded as *Fu* or *Fc*, according to the algorithm's decisions (dashed line box in Fig. 5.6).

The first step in the complexity adjustment phase of the algorithm is the calculation of a new *Nc* value, which is updated using a proportional control loop. The value of *Nc* depends on the normalised difference $\beta$ between $CP^N$ and $CE^T$, calculated as in (Eq. 45). Fig. 5.7 shows how *Nc* is adjusted as a non-linear function of $\beta$. The larger the normalised difference between $CP^N$ and $CE^T$ (horizontal axis), the larger is the decrease or increase of *Nc* (vertical axis). If the *Nc* saturation value is reached, the algorithm keeps it constant until a computational complexity increase is allowed.

[4] The VDCS method presented in [11] adjusts the *Nc* value through unitary increments and decrements, differently from the algorithm version presented in this section and published in [12, 13].

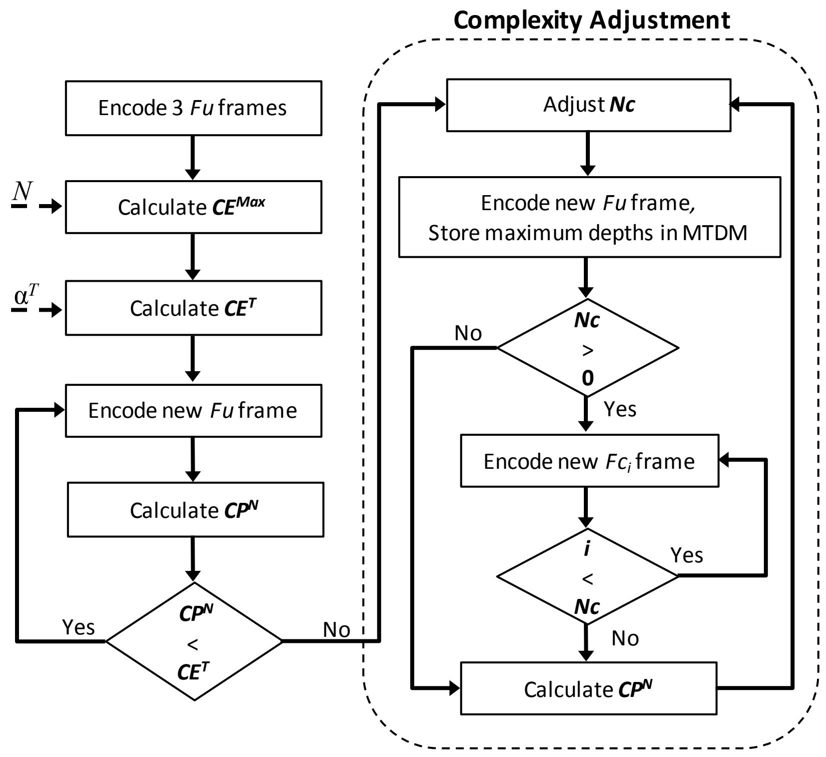


Fig. 5.6: Diagram for the VDCS algorithm.

After the new *Nc* value is defined, a new *Fu* frame is encoded and the maximum Coding Tree depth used for each treeblock of that frame is stored for future use in a matrix called Maximum Tree Depth Map (MTDM). Then, the next *Nc* constrained frames *Fc* are encoded with the RDO process limited to the values saved in the MTDM, which means that the maximum depths used are upper-bounded by the depths used in the co-localised treeblocks of the most recent *Fu* frame. $CP^N$ is then once again calculated and *Nc* is re-adjusted, if necessary. The $CP^N$ value is computed differently from FDCR, according to (Eq. 46), where $CS^{Fi}$ is the computational complexity spent to encode the $i$th frame of the segment, $CS^{Fu}$ is the complexity used to encode the last *Fu* frame, $CS^{Fcj}$ is the computational complexity used in the $j$th frame of the last group of constrained frames and *NE* is the number of frames already encoded in the video segment.

$$\beta = \frac{CE^T - CP^N}{CE^T} \qquad \text{(Eq. 45)}$$

$$CP^{N} = \sum_{i=1}^{NE} CS^{Fi} + \frac{CS^{Fu} + \sum_{j=1}^{Nc} CS^{Fc_j}}{Nc+1} \cdot (N - NE) \qquad \text{(Eq. 46)}$$

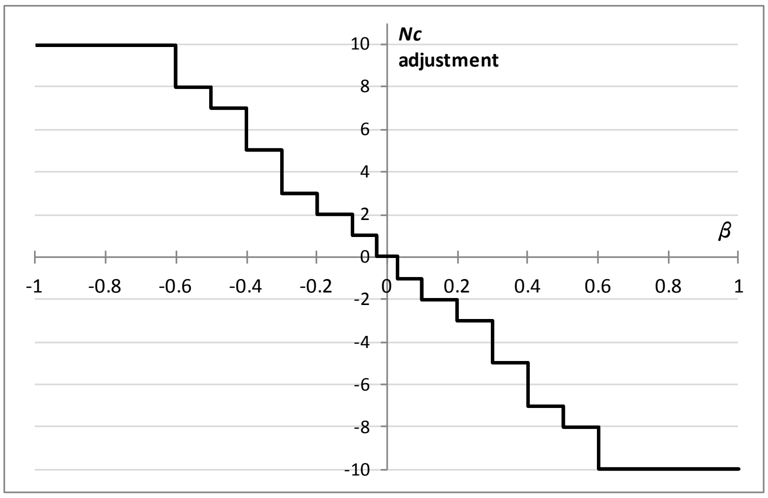


Fig. 5.7: Adjustment of *Nc* value according to *β*.

### 5.3.2 Results for VDCS and Performance Evaluation

The VDCS method was evaluated using the same experimental setup as that of FDCS, based on HM4, three video sequences comprising 500 frames (*BasketballDrive*, *BQTerrace*, *Cactus*) plus a concatenation of two of them (*BasketballCactus*), the *Low Delay P* temporal configuration, five target complexities (from 60% to 100%) and four different QPs (27, 32, 37, and 42). An upper limit (i.e., a saturation value) of 10 frames was used for *Nc* in all experiments. Even though this value can be increased or decreased depending on the system/user requirements, it was found experimentally that larger maximum values for *Nc* would lead to higher unacceptable RD-efficiency losses.

To illustrate the operation of the algorithm, Fig. 5.8 shows the evolution of the *Nc* and $CP^{N}$ values along the encoding of the *BasketballDrive* sequence with QP 32. Fig. 5.8(a) shows that large *Nc* values are used with small target complexities, reaching the saturation value (*Nc* = 10) when the target is set to 60%. With a target complexity of 90%, the maximum *Nc* value used is 3. It is possible to notice that variations on the $CP^{N}$ value shown in Fig. 5.8(b) are followed by increases or decreases in the *Nc* value in Fig.

5.8(a). For example, for a target complexity of 60%, the increase of $CP^N$ from frames 20 to 60 caused successive increments on the $Nc$ value for the same frames. Oppositely, in frame 180 the $CP^N$ value decreased to a value under the target (dashed line in the chart), which means that more computational resources could be employed in the encoding process. This reflected in a decrease of the $Nc$ value from 10 to 8 in frame 180.

Fig. 5.9 shows the relationship between the target complexities defined at the beginning of the encoding process and the actual complexities measured while encoding the video sequences. The dashed line in the graph represents the ideal behaviour of a complexity scaling method. As running complexity results for each tested sequence are around this ideal case, the proposed method is considered quite accurate and capable of scaling computational complexity.

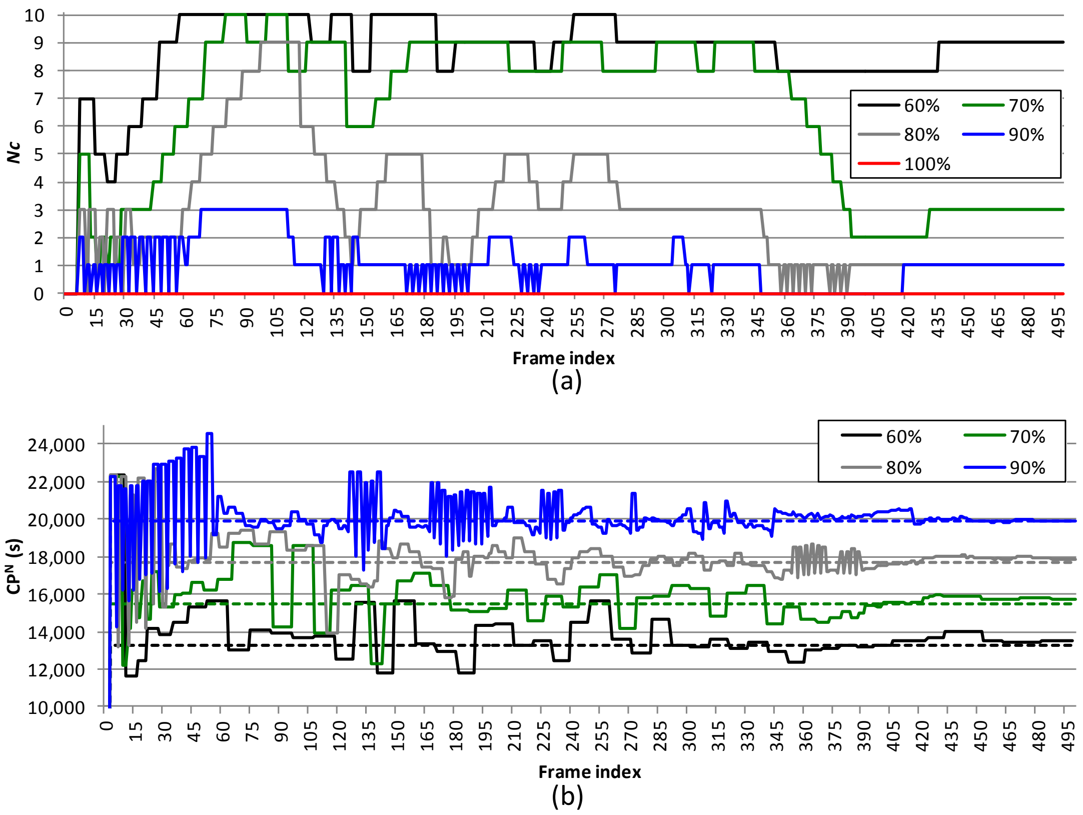


Fig. 5.8: *Nc* and $CP^N$ variations when encoding sequence *BasketballDrive*, QP32.

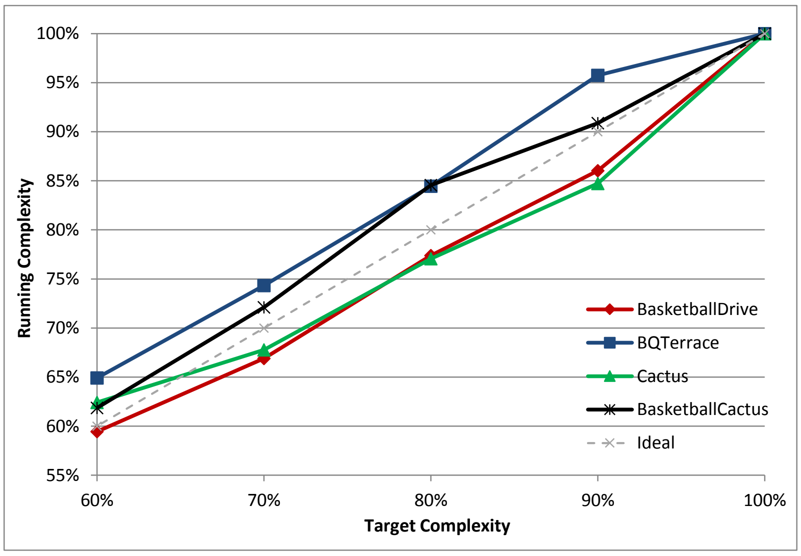


Fig. 5.9: Accuracy of the VDCS complexity scaling method.

The encoder R-D efficiency as a function of the computational complexity under different bit rates is presented in Fig. 5.10. As expected, the best results are obtained when no complexity scaling is applied (i.e., $\alpha^T$ = 100%). However, when compared to the curves obtained for FDCS (see Fig. 5.4), the results for VDCS are represented by Y-PSNR versus bit rate curves that are much closer to each other, which means that the complexity scaling algorithm does not affect the compression efficiency significantly, or at least not as much as FDCS, even when small target complexities are set (e.g., $\alpha^T$ = 60%).

Table 5.2 shows average performance results for the VDCS algorithm. The table shows, for each target complexity tested, the average running complexity, the BD-rate and the BD-PSNR values in comparison to the maximum complexity case (100%). The results show that VDCS is twice more efficient than FDCS, since it achieves similar computational complexity decreases (up to 38%) at the cost of almost half the BD-rate increase (up to +6.29%).

Even though to a smaller extent than FDCS, the use of VDCS also incurs in R-D efficiency losses, which also happens mostly in the low target complexities. This is because in such cases the complexity scaling algorithm uses large *Nc* values, which leads to the use of maximum Coding Tree depth values that may not match very well the actual image characteristics due to temporal changes accumulated since the last *Fu* frame encoded. To solve this problem a new method is presented in the next section, which considers the average motion of each treeblock in the decision of the maximum Coding Tree depth.

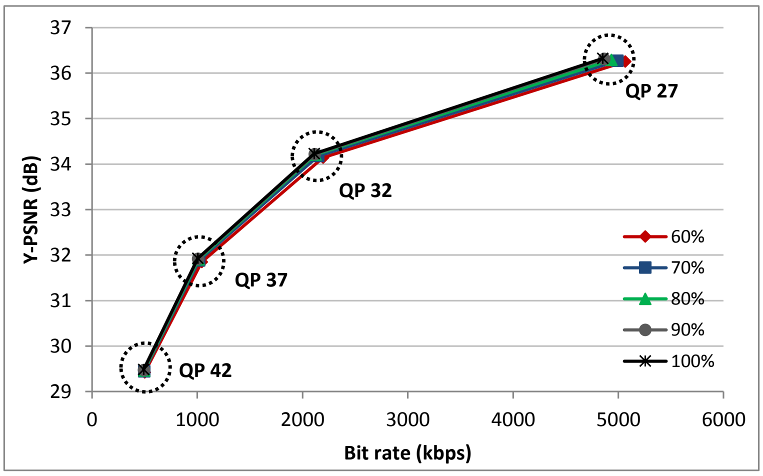


Fig. 5.10: Average R-D performance of the VDCS method.

Table 5.2: Average results obtained for the VDCS algorithm.

| **Target Complexity ($\alpha^T$)** | **Running Complexity** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|
| 100% | 100% | – | – |
| 90% | 90% | +0.58 | -0.02 |
| 80% | 80% | +1.55 | -0.04 |
| 70% | 71% | +3.10 | -0.09 |
| 60% | 62% | +6.29 | -0.18 |

## 5.4 Motion-Compensated Tree Depth Limitation

Just as FDCS and VDCS, the Motion-Compensated Tree Depth Limitation (MCTDL) method [14] is based on dynamic constraining the maximum Coding Tree depth allowed in order to adjust the encoding computational complexity. As previously explained, in VDCS the maximum Coding Tree depths of *Fc* frames are constrained according to the MTDM saved while encoding the last *Fu* frame. However, in videos sequences with fast motion segments this method may decrease the encoding performance, partly due to the fact that in such cases image areas may move away from the position where they were in the last *Fu* frame before another *Fu* frame is encoded and the MTDM is updated. As a result, the *Fc* frames occurring between two consecutive

*Fu* frames are encoded using a MTDM that may not be well matched to the current frame content, with the mismatch becoming worse towards the end of the group of *Fc* frames. This problem is more severe in cases where the target complexity is small and thus *Nc* large.

To solve this problem, MCTDL adds a new step to the complexity adjustment phase of VDCS, updating the MTDM after encoding each *Fc* frame according to the average motion of each treeblock, effectively motion-compensating the MTDM. This compensation of the motion effect is performed using motion information from the previous frame in order to predict the most probable displacement from frame $k-1$ to frame $k$ for the image region corresponding to each CTU.

## 5.4.1 Algorithm Overview

As in VDCS, the maximum depths saved when encoding an *Fu* frame are stored in an $n \times m$ matrix (referred from now on as MTDM[$n$][$m$]), where $n$ is the number of CTUs in the horizontal dimension and $m$ is the number of CTUs in the vertical dimension of a frame. Simultaneously, a weighted average motion vector (MV) for each CTU is computed and stored in another $n \times m$ matrix (referred from now on as MV[$n$][$m$]), where the weights are proportional to the size of each PU inside the CTU, as detailed later in this section. When the first *Fc* frame which follows an *Fu* frame is encoded, the values stored in MTDM[$n$][$m$] are motion compensated according to the MVs stored in MV[$n$][$m$]. Each treeblock in *Fc* is then encoded with this motion compensated map of maximum depths. While encoding each *Fc* frame, new average MVs for each CTU are computed and stored in MV[$n$][$m$] in order to update the MTDM to be used in the next *Fc* frame.

To fully understand the maximum depth map construction procedure, let us explain as an example how the maximum depth to be applied to a certain $\mathrm{CTU}^{k-1}_{(o,p)}$ belonging to a frame $F_{k-1}$ is derived. Consider that this CTU average MV is $\mathrm{MV}^{k-1}_{(o,p)} = (x, y)$, the reference frame is $F_r$, the coordinates $(o,p)$ are the CTU line and column in frame $F_{k-1}$, and the coordinates $(x, y)$ are the CTU line and column displacement from frame $F_r$ to frame $F_{k-1}$. These elements are all shown in Fig. 5.11 for better comprehension. Now, let us assume that the motion speed of the group of pixels constituting the CTU is approximately constant. The displacement of the pixels belonging to the example CTU, from frame $F_{k-2}$ to frame $F_{k-1}$ can be computed by dividing $MV^{k-1}{}_{(o,p)}$ components $m$ and $n$ by the number of frames between frame $F_{k-1}$ and $F_r$. Still assuming constant motion speed, we can predict where the pixels of $CTU^{k-1}{}_{(o,p)}$ may be located in frame $F_k$. The solid line arrow in Fig. 5.11 shows the estimated motion displacement from frame $F_{k-1}$ to frame $F_{k-2}$ and the pointed line arrow shows the predicted motion displacement from frame $F_{k-1}$ to frame $F_k$. The maximum Coding Tree depth for each $CTU^{k}{}_{i,j}$ in frame $F_k$ is then copied from the corresponding position in $MTDM^{k-1}$ and stored into a motion compensated MTDM for frame $F_k$, named $MTDM^k$.

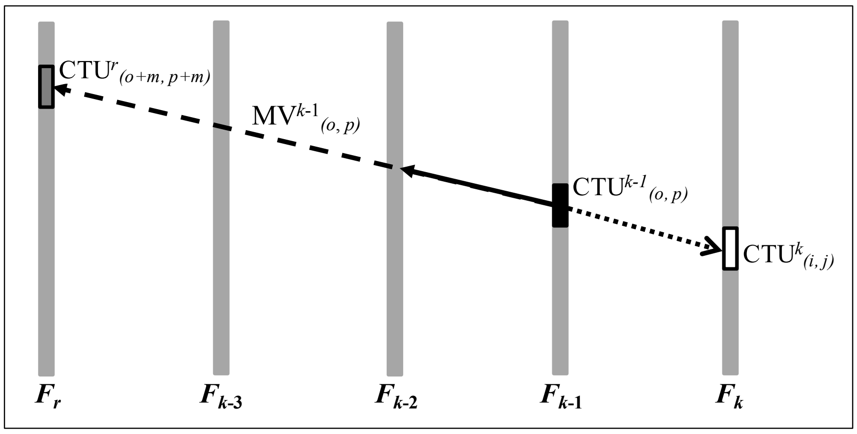


Fig. 5.11: Example of MTDM motion compensation.

Fig. 5.12 shows the high-level diagram of the MCTDL algorithm. The operation of this algorithm is very similar to that of VDCS, which was explained in details in section 5.3.1. The starting phase is exactly the same as in VDCS, with the computations of $CE^{Max}$, $CE^{T}$ and $CP^{N}$ performed accordingly to (Eq. 42), (Eq. 43) and (Eq. 46), respectively. While $CP^{N}$ is smaller than $CE^{T}$, all frames are encoded as *Fu*. When this condition ceases to hold, the complexity adjustment phase is activated (portion inside the dashed line in Fig. 5.12) and frames start being encoded as *Fu* or *Fc*, according to the algorithm's decisions.

The new operations introduced in MCTDL executed during the complexity adjusting phase are underlined in the diagram of Fig. 5.12. Initially, the calculation of *Nc* is done as in VDCS, according to the normalised difference between $CP^{N}$ and $CE^{T}$, as in (Eq. 45). After that, a new *Fu* frame is encoded and the maximum Coding Tree depths are saved in the MTDM. Simultaneously, the encoder records the average weighted MVs belonging to each CTU, so that the MTDM can be motion compensated in the next step.

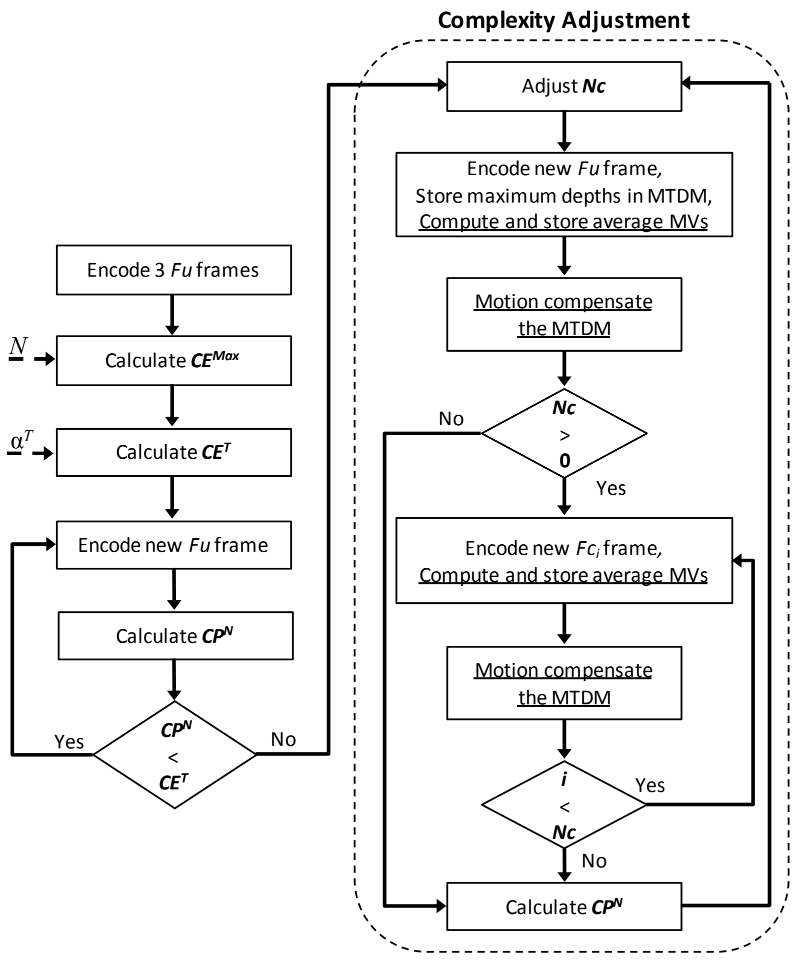

Complexity Adjustment
Encode 3 Fu frames
N
Calculate CE^Max
α^T
Calculate CE^T
Encode new Fu frame
Calculate CP^N
CP^N < CE^T
Yes
No
Adjust Nc
Encode new Fu frame, Store maximum depths in MTDM, Compute and store average MVs
Motion compensate the MTDM
Nc > 0
No
Yes
Encode new Fc_i frame, Compute and store average MVs
Motion compensate the MTDM
i < Nc
Yes
No
Calculate CP^N

Fig. 5.12: Diagram for the MCTDL algorithm.

The average MV is computed as a weighted average of all MVs belonging to the PUs in the CTU. The weights applied to each MV depend on the PU size relatively to the CTU size, as shown in Table 5.3. For example, a 64×64 PU has a weight equals to 1, since the number of samples in the PU and in a 64×64 CTU is the same. A 32×32 PU has a weight equals to 1024 / 4096 = 0.25, where 1024 is the number of samples in the PU and 4096 is the number of samples in the 64×64 CTU.

If *Nc* is zero (i.e., no complexity reduction is necessary), the algorithm keeps encoding *Fu* frames, but still computes a new $CP^N$ and a new *Nc* after each *Fu* frame is encoded in order to detect whenever complexity reduction is required. In this case, *Nc* becomes larger than zero and *Fc* frames start being encoded with the maximum depth used in the RDO process limited to the values saved in the MTDM, which are reordered according to the average MVs for each CTU after encoding each *Fc* frame. By doing that, the encoder keeps an updated estimation of the location where is the CTU corresponding to a determined value in the MTDM.

Differently from the VDCS method, in which the MTDM data becomes out-of-date when large *Nc* values are used, in this new method the encoder is able apply a Coding Tree depth limitation that fits better the characteristics of each image region. This is illustrated in the example presented in Fig. 5.13, which shows a fragment of the 58th and 59th frames of the *BasketballDrive* video sequence. The maximum Coding Tree depth is shown for each CTU in the fragments (i.e., the information recorded in the MTDM) and the average MVs calculated as explained before are shown in Fig. 5.13(b) for each inter-predicted CTU. It is possible to notice in the figure that motion compensating the MTDM according to the average MV yields a good estimation of the maximum Coding Tree depth for the next frame.

Table 5.3: Weights for average MV calculation.

| **PU size** | **Number of Samples** | **Weight** |
|---|---|---|
| 64×64 | 4096 | 1 |
| 64×32 | 2048 | 0.5 |
| 32×64 | 2048 | 0.5 |
| 64×16 | 1024 | 0.25 |
| 16×64 | 1024 | 0.25 |
| 32×32 | 1024 | 0.25 |
| 32×16 | 512 | 0.125 |
| 16×32 | 512 | 0.125 |
| 32×8 | 256 | 0.0625 |
| 8×32 | 256 | 0.0625 |
| 16×16 | 256 | 0.0625 |
| 16×8 | 128 | 0.03125 |
| 8×16 | 128 | 0.03125 |
| 16×4 | 64 | 0.015625 |
| 4×16 | 64 | 0.015625 |
| 8×8 | 64 | 0.015625 |
| 8×4 | 32 | 0.0078125 |
| 4×8 | 32 | 0.0078125 |

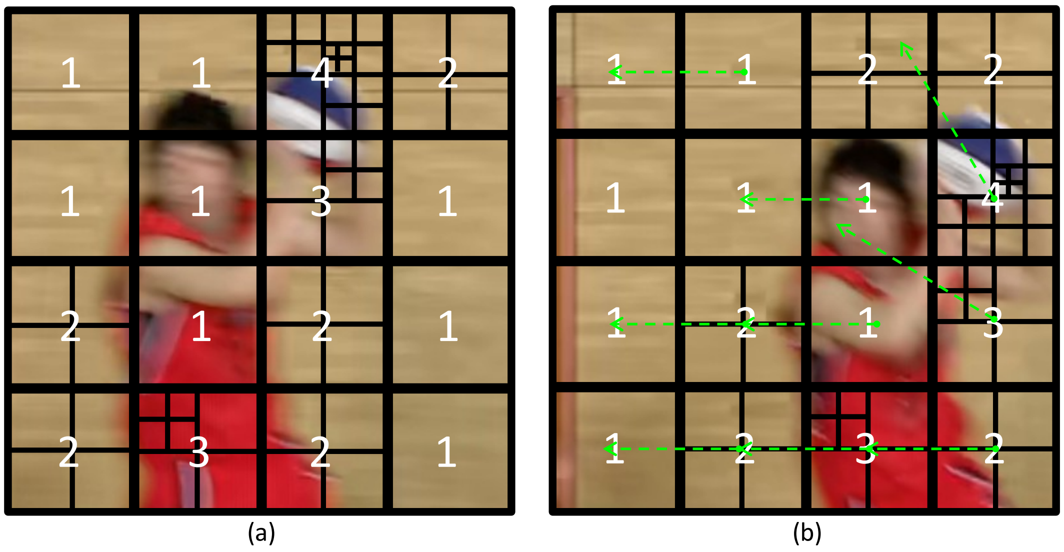


Fig. 5.13: MTDM fragment and average MVs for the (a) 58th and (b) 59th frames of the *BasketballDrive* sequence (QP 32).

## 5.4.2 Results for MCTDL and Performance Evaluation

The same setup used in the evaluation of FDCS and VDCS was used for performance assessment of the MCTDL method.

Fig. 5.14 shows a graph with the actual complexities obtained after encoding the video sequences as a function of the target complexities. The figure presents results for QP 32, but other values were also tested and show similar behaviour. The dashed line represents the target complexity and the solid lines represent each video tested. As each video sequence presents results that are close to the ideal, it is possible to conclude that the proposed method is accurate and capable of scaling computational complexity.

Table 5.4 presents average results for MCTDL in terms of running complexity, BD-rate and BD-PSNR differences for all target complexities tested in comparison to the case in which no complexity scaling is applied, i.e., when target complexity is 100%. An increase in the bit rate was observed in all cases, but especially when the target complexity ($\alpha^T$) is set to 60%. However, in comparison to VDCS and FDCS, the MCTDL method yields better R-D results for all target complexities. In the lowest target complexity case ($\alpha^T = 60\%$), an average complexity reduction of 39% is achieved at the cost of a BD-rate increase of 5.42%.

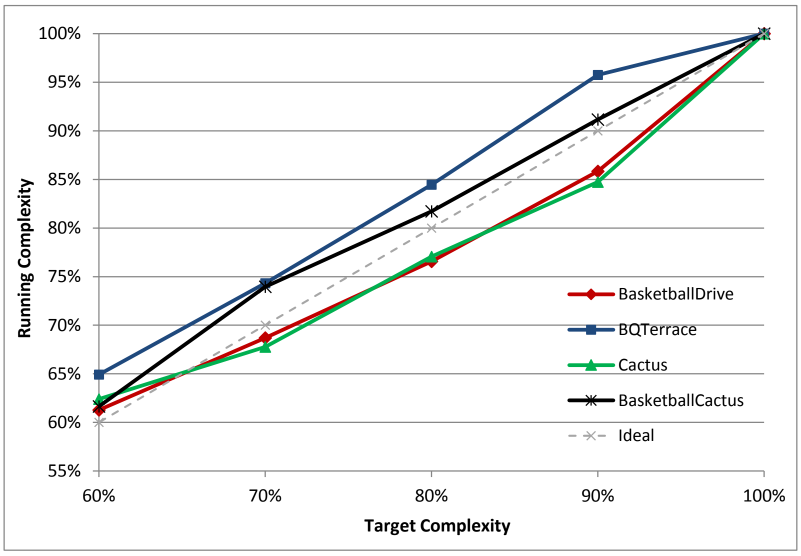


Fig. 5.14: Accuracy of the MCTDL complexity scaling method.

Table 5.4: Average results obtained for the MCTDL algorithm.

| **Target Complexity ($\alpha^T$)** | **Running Complexity** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|
| 100% | 100% | – | – |
| 90% | 90% | +0.54 | -0.02 |
| 80% | 80% | +1.46 | -0.05 |
| 70% | 71% | +3.22 | -0.10 |
| 60% | 61% | +5.42 | -0.16 |

## 5.5 Coding Tree Depth Estimation

Despite providing a fair complexity scaling without significantly decreasing compression efficiency, the MCTDL method (as well as VDCS) uses only the temporal correlation between neighbouring frames in order to determine the maximum Coding Tree depth tested for each CTU. If spatial correlation was also used combined with temporal data, the R-D performance achieved when complexity scaling is enabled could be potentially improved.

This section presents a Coding Tree Depth Estimation (CTDE) method [15, 16], which is based on the method presented in 5.4 and allows estimating the best maximum Coding Tree depth for a CTU based on both spatial and temporal correlations observed in neighbouring CTUs located in the same and previous frames. As explained before, the computational complexity scaling approaches used in the previously presented methods rely on the fact that the maximum Coding Tree depth tends to be constant in co-localised areas of adjacent frames, as experimentally verified. For the CTDE method, a set of experiments were conducted in order to analyse how frequently a certain CTU is encoded with the same or smaller maximum Coding Tree depth than its spatially neighbouring CTUs (top, left and top-left CTUs). The results of such experiments are presented in Table 5.5 and show that in most cases the CTUs surrounding a given CTU are encoded with maximum depths that are equal to or exceed the maximum depth of that CTU. As all Coding Tree depths from 0 to *n* are tested through RDO when a depth *n* is selected as maximum, no R-D efficiency losses would be observed if a depth greater than the one that should be used for encoding that CTU was selected as maximum.

Table 5.5: Maximum depths used in neighbouring CTUs.

| **Spatially Neighbouring CTU** | **Greater or equal depth (%)** | **Smaller depth (%)** |
|---|---|---|
| Top | 83.47 | 16.53 |
| Left | 82.24 | 17.76 |
| Top-left | 91.84 | 8.16 |

Based on these observations and considering the complexity scaling scheme of VDCS and MCTDL, in CTDE the maximum Coding Tree depth allowed for each CTU is decided taking into consideration the type of frame (*Fc* or *Fu*), as well as the maximum depths used in the temporally and spatially neighbouring CTUs.

### 5.5.1 Algorithm Overview

Let $CTU^{k}_{i,j}$ be a Coding Tree block located at position *i*, *j* of a frame with index *k*. If *k* is an *Fu* frame, the CTU is encoded with no complexity limitation, which means that the maximum Coding Tree depth possible is allowed. If *k* is an *Fc* frame, the maximum Coding Tree depth allowed is defined to be the largest of the maximum Coding Tree depths used at the:

i. Left side neighbouring CTU – i.e., $CTU^{k}_{(i,j-1)}$;
ii. Top neighbouring CTU – i.e., $CTU^{k}_{(i-1,j)}$;
iii. Top-left neighbouring CTU – i.e., $CTU^{k}_{(i-1,j-1)}$;
iv. Co-localised CTU from the previous frame – i.e., $CTU^{k-1}_{(i,j)}$;
v. Motion-compensated CTU from previous frame – i.e., $CTU^{k-1}_{(o,p)}$.

Except for the last value listed above, which requires some processing to be obtained, all values are straightforward to obtain by simply storing Coding Tree depths used in each CTU. Two MTDMs are used to store values corresponding to CTUs in the current and previous frames ($k$ and $k$−1 in display order): MTDM$^{k}$ and MTDM$^{k-1}$, respectively. A third MTDM is used to store values of the motion-compensated CTUs from the previous frame: CMTDM$^{k}$.

The CTDE method follows a complexity scaling scheme very similar to MCTDL, so that the high-level diagram previously presented in Fig. 5.12 is slightly modified for CTDE, as shown in Fig. 5.15. In Fig. 5.15, the steps which are incorporated into CTDE or modified from MCTDL are underlined. While a frame is encoded (both *Fu* and *Fc*) in the complexity adjustment phase, the maximum Coding Tree depths are stored in MTDM$^{k}$ and the average MVs are computed and saved. Then, after encoding the whole frame, CMTDM$^{k}$ is created by motion-compensating the values in MTDM$^{k}$ (as explained in 5.4), which is finally copied to MTDM$^{k-1}$.

The main difference between the CTDE and MCTDL consists in the way the maximum Coding Tree depth allowed for each CTU is computed. As this process is not depicted in Fig. 5.15, since it is a part of the instruction to encode a new frame ("Encode new *Fu* frame" and "Encode new $Fc_i$ frame" lines in Fig. 5.15), it is detailed in the pseudo-code presented in Fig. 5.16, which is executed for each frame (both *Fu* and *Fc*).

In an *Fu* frame, all Coding Tree depths are allowed, so that the encoder sets the variable *max_depth_allowed* to the maximum Coding Tree depth possible (lines 04-05, in Fig. 5.16), which varies from 1 to 4, according to the encoder configuration used. In an *Fc* frame, the *max_depth_allowed* value is decided for each CTU by taking the maximum value among MTDM$^{k}_{(i,j-1)}$, MTDM$^{k}_{(i-1,j)}$, MTDM$^{k}_{(i-1,j-1)}$, MTDM$^{k-1}_{(i,j)}$, and CMTDM$^{k}_{(i,j)}$ (lines 06-08), which correspond to the five depths listed in the first paragraphs of this section.

To allow further computational complexity reduction, in cases with very small target complexities the CTDE method allows decreasing the maximum Coding Tree depth by one additional unit if the maximum value for *Nc* is already achieved but complexity is still above the target. This is shown in lines 09-10 in Fig. 5.16. If *Nc* reaches its maximum value (*MAX_Nc*), the computational complexity can be further reduced by decreasing *max_depth_allowed* in one unit. Even though *MAX_Nc* can be theoretically set to any value, increasing it too much leads to large long term R-D efficiency losses. To avoid this problem, and following the results of empirical tests, the value of *MAX_Nc* used in the experiments presented in the next section was set to half the video frame rate.

The CTU is finally encoded according to *max_depth_allowed* (line 11) and the maximum Coding Tree depth used to encode it after testing all tree possibilities is stored in MTDM$^k$ (line 12) for use in the decision of maximum depths for the next CTUs.

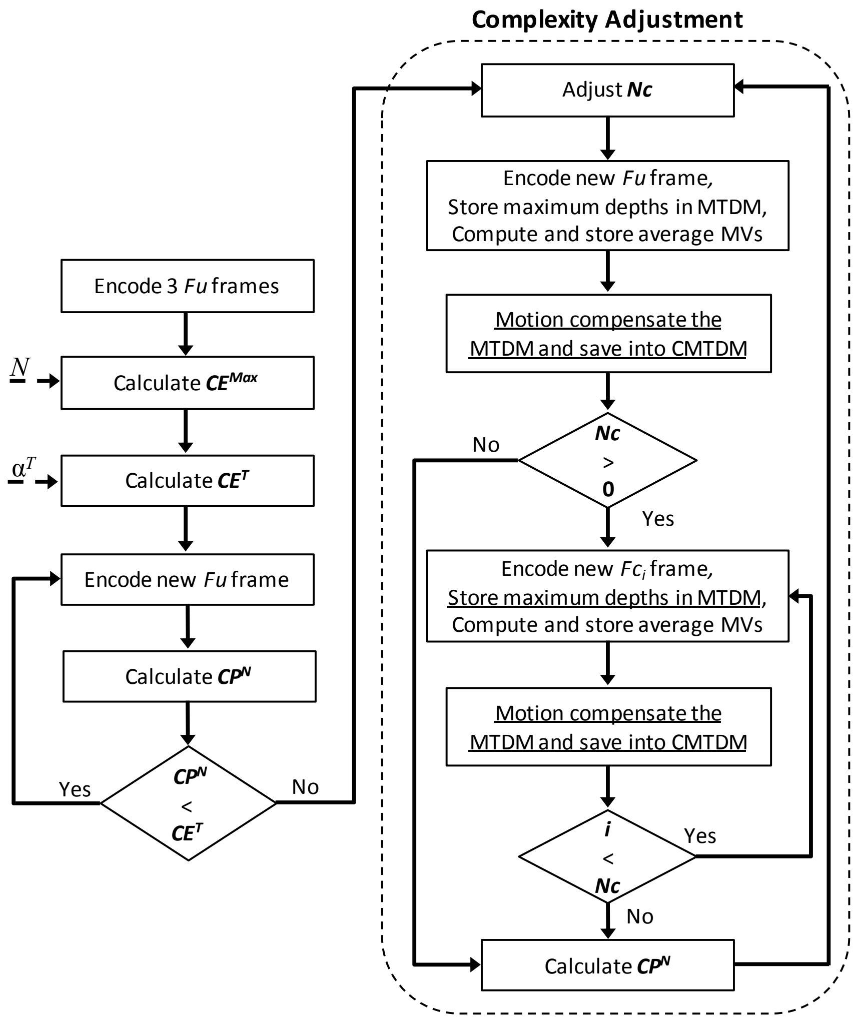

Complexity Adjustment
Adjust Nc
Encode new Fu frame, Store maximum depths in MTDM, Compute and store average MVs
Motion compensate the MTDM and save into CMTDM
Nc > 0
No
Yes
Encode new Fc_i frame, Store maximum depths in MTDM, Compute and store average MVs
Motion compensate the MTDM and save into CMTDM
i < Nc
Yes
No
Calculate CP^N
Encode 3 Fu frames
N
Calculate CE^Max
α^T
Calculate CE^T
Encode new Fu frame
Calculate CP^N
CP^N < CE^T
Yes
No

Fig. 5.15: Diagram for the CTDE algorithm.

```
for each CTU in a frame $k$
  $i$ ← CTU line position
  $j$ ← CTU column position
  if $k$ is an $Fu$ frame
    max_depth_allowed ← maximum depth possible
  else
    max_depth_allowed ← max($MTDM^{k}_{(i-1,j)}$, $MTDM^{k}_{(i,j-1)}$,
                            $MTDM^{k}_{(i-1,j-1)}$, $MTDM^{k-1}_{(i,j)}$, $CMTDM^{k}_{(i,j)}$)
    if $Nc$ = $MAX\_Nc$
      max_depth_allowed ← max_depth_allowed - 1
  encode CTU limiting depth to max_depth_allowed
  $MTDM^{k}_{i,j}$ ← maximum depth used to encode current CTU
end for
```

Fig. 5.16: Pseudo-code for the maximum Coding Tree depth decision in CTDE.

### 5.5.2 Results for CTDE and Performance Evaluation

The CTDE method was evaluated by having its complexity scaling accuracy and R-D efficiency measured under specific target complexities. The algorithm was implemented in the HM encoder – version 8.2 (HM8.2) – and evaluated with six video sequences (*BQMall*, *BQTerrace*, *Cactus*, *vidyo1*, *BasketballDrive*, *Traffic*), all of which are detailed in Appendix B. The encoder performance was evaluated under five target complexities: 60%, 70%, 80%, 90%, and 100%. The *Low Delay P* temporal configuration [43] was used in all tests (see Appendix B for details).

Fig. 5.17 shows a chart with the average complexities obtained after encoding the video sequences as a function of the target complexities. The dashed line in the graph represents the target complexity, while the other lines represent the actual running complexity for various test video sequences. As it can be seen, the actual running complexity for each tested sequence is close to the target, thus showing that CTDE is accurate and capable of scaling computational complexity to within a tight interval around the desired value. In the worst case ($\alpha^{T}$ = 60%), the difference between target and actual running computational complexity was around 3%.

Concerning the video encoding performance, Table 5.6 presents average results when the CTDE method is used. Besides the coding performance indicators variations (i.e., BD-rate increases and BD-PSNR decreases), the table also shows average results for running complexity, considering all videos and QPs tested. An increase in the bit rate was observed in all sequences coded at low complexity points, especially when the target complexity is set to 60%. However, in comparison to all methods presented so far in this thesis, CTDE produced the best R-D efficiency results for all target complexities, with encoding efficiency closer to the values obtained when no complexity scaling is applied. In the worst case ($\alpha^T$ = 60%), the BD-rate increase was 4.74% and the decrease in BD-PSNR was 0.13 dB.

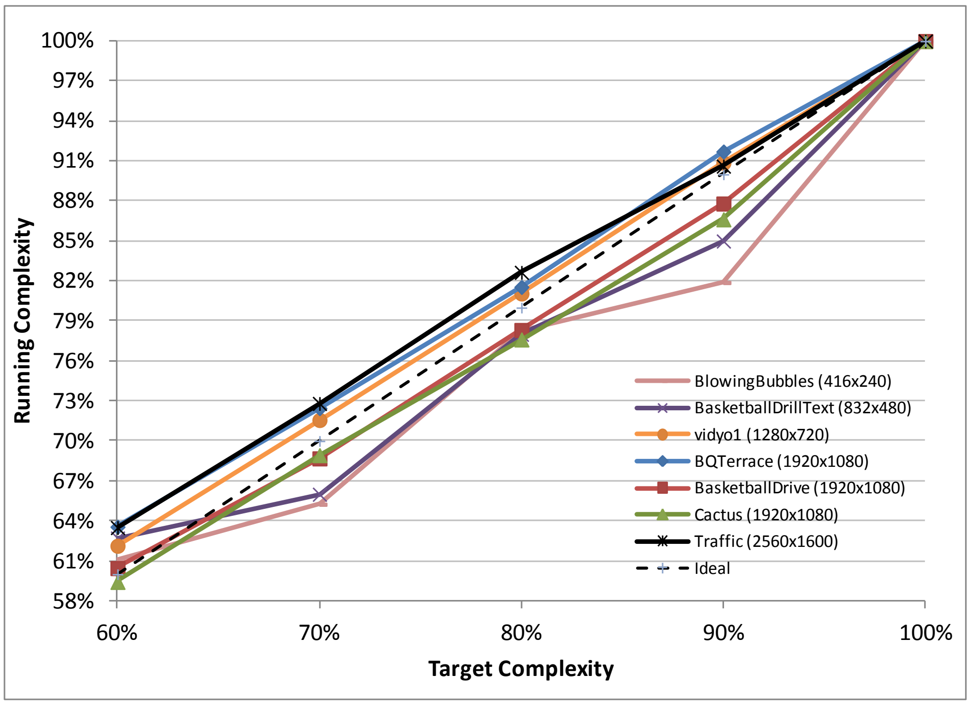


Fig. 5.17: Accuracy of the CTDE complexity scaling method.

Table 5.6: Average results obtained for the CTDE algorithm.

| Target Complexity ($\alpha^T$) | Running Complexity | BD-rate (%) | BD-PSNR (dB) |
|---|---|---|---|
| 100% | 100% | – | – |
| 90% | 89% | +1.05 | −0.03 |
| 80% | 80% | +1.62 | −0.04 |
| 70% | 72% | +3.84 | −0.10 |
| 60% | 63% | +4.74 | −0.13 |

## 5.6 Constrained Coding Units and Prediction Units

As the experiments presented in chapter 4 have shown, CUs are the frame partitioning structures which allow the largest reductions in computational complexity when constrained partition definition procedures are used. However, the same experiments have also exposed that constraining CUs in order to decrease computational complexity incurs in the largest costs in terms of R-D efficiency loss in comparison to PUs and TUs (see Table 4.6 and Table 4.7).

According to the experiments presented in chapter 4, the frame partitioning structure that yields the second largest reductions in computational complexity when constrained are PUs (see Table 4.6). They are also the structures which presented the second smallest ratios between BD-rate and computational complexity decrease (see Table 4.7), which means that constraining PUs is more advisable than constraining CUs if maintaining the R-D efficiency is a crucial preoccupation in the system implementation.

Based on that analysis and on the methods for computational complexity scaling previously presented, a method based on Constrained Coding Units and Prediction Units (CCUPU) [17, 18] is proposed in this section. The goal of this method is to allow adjusting the HEVC encoding structures in order to achieve a dynamic scaling of the computational complexity beyond levels achieved by the previous approaches.

### 5.6.1 Rate-Distortion-Complexity Relationship in CUs and PUs

A more extensive analysis of the R-D-C efficiency of different CU and PU configurations was performed in order to assist the development of the CCUPU method. This analysis was necessary mainly because the constraining of PUs performed in the experiments of chapter 4 was based on simply enabling or disabling the use of AMP, which is the only parameter regarding the configuration of PUs available in the HM configuration. For the experiments described in this section, a configuration in which PU sizes smaller than *2N×2N* are disabled was created through modifications of the HM code. This *2N×2N*-only configuration was used in the experiments and compared with the unmodified HM encoder. A subset of 10 video sequences (*BlowingBubbles*, *RaceHorses1*, *BasketballDrillText*, *PartyScene*, *SlideShow*, *vidyo1*, *ParkScene*, *BasketballDrive*, *NebutaFestival*, and *Traffic*) was selected for tests from the CTC document. Their characteristics are detailed in Appendix B.

The 10 video sequences were encoded using four different QPs (27, 32, 37, and 42) and the *Low Delay* P temporal encoder configuration while varying two parameters: the maximum Coding Tree depth allowed for the CTUs and the possibility of splitting a CU into PUs smaller than *2N×2N* for inter-frame prediction. Table 5.7 presents the eight configurations tested. Table 5.8 shows comparative results in terms of BD-rate and computational complexity for the different configurations presented in Table 5.7. In the first three rows of Table 5.8, those configurations in which only the maximum Coding Tree depth varies (*Cfg. 1* to *Cfg. 4*) are compared, while the remaining four rows compare results when only the possibility of using PUs smaller than *2N×2N* varies.

Table 5.7: CU and PU configurations tested.

| **Configuration** | **Max. Coding Tree Depth** | **PU smaller than *2N×2N*** |
|---|---|---|
| *Cfg. 1* | 4 | Yes |
| *Cfg. 2* | 3 | |
| *Cfg. 3* | 2 | |
| *Cfg. 4* | 1 | |
| *Cfg. 5* | 4 | No |
| *Cfg. 6* | 3 | |
| *Cfg. 7* | 2 | |
| *Cfg. 8* | 1 | |

Table 5.8: Comparison between encoding configurations.

| **Comparison** | **BD-BR (%)** | **ΔComp (%)** | **BD-BR/ΔComp ($R_{i,j}$)** | **Avg. BD-BR (%)** | **Avg. ΔComp (%)** |
|---|---|---|---|---|---|
| *Cfg. 2* vs *Cfg. 1* | +1.10 | -28.4 | 0.0373 | +3.33 | -41.3 |
| *Cfg. 3* vs *Cfg. 2* | +2.05 | -45.5 | 0.0451 | | |
| *Cfg. 4* vs *Cfg. 3* | +6.84 | -49.9 | 0.1371 | | |
| *Cfg. 5* vs *Cfg. 1* | +1.13 | -44.1 | | +0.93 | -43.5 |
| *Cfg. 6* vs *Cfg. 2* | +1.09 | -49.3 | | | |
| *Cfg. 7* vs *Cfg. 3* | +0.68 | -43.4 | | | |
| *Cfg. 8* vs *Cfg. 4* | +0.81 | -37.1 | | | |

The results in Table 5.8 show that when the maximum Coding Tree depth allowed is reduced by one unit, an average increase of 3.33% in the BD-rate and a computational complexity decrease of 41.3% are observed. On the other hand, turning off the possibility of using PUs smaller than *2N×2N* in inter-frame prediction results in an average BD-rate increase of 0.93% and a computational complexity decrease of 43.5%, which seems much more profitable from the point of view of R-D-C trade-offs.

R-D costs and computational complexity were jointly analysed by computing the ratio $R_{i,j}$ between the BD-rate increase and the decrease in computational complexity ΔComp when the maximum Coding Tree depth is reduced from *i* to *j* (without changing the possibility of using PUs smaller than *2N×2N*). By calculating the ratio, it is possible to compare the configurations in order to detect which one incurs in the largest impact on compression efficiency per computational complexity saving. The results are presented in the fourth column of Table 5.8 and show that the ratio $R_{i,j}$ is smaller when decreasing Coding Tree depths of higher values (from *Cfg. 1* to *Cfg. 2* and from *Cfg. 2* to *Cfg. 3*), even though ΔComp is moderate in such cases. On the other hand, when decreasing the Coding Tree depth of lower values (from *Cfg. 3* to *Cfg. 4*), the ratio $R_{i,j}$ increases significantly (0.1371), but ΔComp also increases. These results show that, after constraining the PU size, the best option for decreasing computational complexity is to restrict the use of large Coding Tree depths.

## 5.6.2 Encoding Constrained CTUs

The CCUPU method scales the computational complexity by using a two-level constraining scheme. As suggested by the experiments presented in the previous section, a better R-D efficiency is achieved by the encoder if limiting PU shapes is performed prior to limiting Coding Tree depths. Thus, in the first constraining level the number of CUs that are split into PUs smaller than *2N×2N* for inter prediction is adjusted to scale the computational complexity. Then, if the first constraining level is not enough to achieve the target complexity, the number of CTUs that can be encoded at any Coding Tree depth is also adjusted. The amount of CTUs constrained in the first and second levels is controlled by two parameters: $N_c^1$ and $N_c^2$, respectively.

CTUs constrained with the first parameter ($N_c^1$) are called PU-constrained CTUs, while CTUs constrained with the second parameter ($N_c^2$) are called CU-constrained

CTUs. Notice, however, that CU-constrained CTUs are also PU-constrained CTUs, since both parameters are used for complexity scalability in these cases.

When the first parameter is used for complexity constraining, the encoder disables PU shapes smaller than *2N×2N* in those CTUs that most probably yield low R-D costs according to information of their co-localised CTUs in the previous frame, as explained in the next section. Once the number of PU-constrained CTUs reaches the number of CTUs in a frame, the second constraining parameter is changed to further reduce computational complexity. In this case, the method adjusts the number of CTUs in which the maximum Coding Tree depth is decided by taking into account previous encoding decisions, based on the fact that spatial and temporal neighbouring CTUs tend to present the same or similar maximum Coding Tree depths, as discussed in the previous methods presented in this chapter.

For the CCUPU method, these characteristics were further exploited by analysing additional spatial neighbouring CTUs of a certain CTU. Besides the top, left and top-left CTUs, which were already analysed in Table 5.5, statistics for the top-right CTU and the co-localised CTU in the previous frame are shown in Table 5.9. The results show that in most cases the CTUs surrounding the CTU to be encoded and its co-localised CTU in the previous frame are encoded with maximum depths that are equal to or exceed the maximum depth of the CTU under consideration.

Table 5.9: Maximum depths used in neighbouring CTUs.

| **Spatially Neighbouring CTU** | **Greater or equal depth (%)** | **Smaller depth (%)** |
|---|---|---|
| Co-localised | 93.14 | 6.86 |
| Top | 83.47 | 16.53 |
| Left | 82.24 | 17.76 |
| Diagonal top-left | 91.84 | 8.16 |
| Diagonal top-right | 86.33 | 13.67 |

As in the case of unconstrained CTUs, the proposed method also uses RDO to decide the best Coding Tree configuration in CU-constrained CTUs. However, instead of using a fixed maximum depth for every CTU, this value varies from one CTU to another according to spatio-temporal correlation. In CTUs with smaller maximum depths, the number of Coding Tree possibilities is smaller and thus the computational cost of finding the best tree configuration is reduced.

Based on this analysis, the maximum Coding Tree depth *max_depth* used to encode a CU-constrained $CTU^{k}_{(i,j)}$ located at position (*i, j*) of frame *k* is defined as the largest of the maximum coding tree depths used at the:

i. Left side neighbouring CTU – i.e., $CTU^{k}_{(i,j-1)}$;
ii. Top neighbouring CTU – i.e., $CTU^{k}_{(i-1,j)}$;
iii. Top-left neighbouring CTU – i.e., $CTU^{k}_{(i-1,j-1)}$;
iv. Top-right neighbouring CTU – i.e., $CTU^{k}_{(i-1,j+1)}$;
v. Co-localised CTU from the previous frame – i.e., $CTU^{k-1}_{(i,j)}$;

The five values above are obtained directly from stored information of CTUs already encoded in the current and previous frame and saved in the MTDM structures used in VDCS, MCTDL and CTDE methods. Since all tree depths, from 1 to *max_depth*, are evaluated when the RDO process is active, no loss of R-D efficiency is observed if a depth greater than the one that should be used for encoding the current CTU is selected as maximum.

## 5.6.3 Algorithm Overview

The pseudo-code for the CCUPU method is presented in Fig. 5.18. The method starts by calculating the target time $T_t^{GOP}$ for a GOP, which is needed throughout the encoding process to adjust the number of constrained CTUs at each level of the constraining scheme. This adjustment is performed according to the difference between the time used to encode the previous GOP and the target encoding time for a GOP. For real-time encoding and typical GOP sizes (e.g., 16 frames or smaller) and frame rates (e.g., 50 frames per second), adjustments at GOP granularity are at sub-second scale.

The target time $T_t^{GOP}$ for a GOP is calculated following (Eq. 47), where *nFR* is the number of frames in a GOP and $T_t$ is the target time to encode a frame, both of which are inputs to the complexity scaling algorithm. The parameter $T_t$ is specified by the user or given by the encoding system according to the current load level of the processing elements (e.g., percentage of CPU in use) or the device battery status, and it represents the average maximum time per frame that the encoder is expected to use. As the GOP encoding time depends on the number of frames composing it, $T_t$ is used as an input for further calculation of $T_t^{GOP}$. The calculation of $T_t^{GOP}$ is shown in line 02 in the pseudo-code for the complexity scalability method presented in Fig. 5.18.

$$T_t^{GOP} = T_t \cdot nFR \qquad \text{(Eq. 47)}$$

At least one GOP in a sequence must be encoded without any restriction in order to allow the encoder to detect if complexity adjustment is needed or not. The first GOP is encoded with only unconstrained CTUs (i.e., the full RDO process is applied to all CTUs in its frames), as shown in lines 03-08 in Fig. 5.18, and the time spent to encode each frame is saved. Based on the encoding time of the first GOP, the method starts adjusting (if necessary) the computational complexity in order to achieve the target processing time. If the time spent to encode the frames of the previous GOP is larger than the target, then the number of complexity-constrained CTUs in each frame is increased in the next GOP. Otherwise, it is decreased. This process is repeated for every GOP.

The ratio $\alpha^{GOP}$ between the encoding time of the previous GOP and the target time is computed as shown in (Eq. 48), where $T_e^{GOP}$ is the actual time elapsed when encoding the previous GOP. The value of $\alpha^{GOP}$ is used as a parameter in the proportional control loop shown in (Eq. 49), which adjusts the number $N_c^k$ of constrained CTUs per frame in the next GOP. In $N_c^k$, $k$ represents which of the two complexity constraining levels is used. The method starts scaling computational complexity by adjusting the number of CTUs which allow PUs smaller than $2N\times2N$. This number is represented as $N_c^1$ in the pseudo-code of Fig. 5.18. Once $N_c^1$ reaches its limit (i.e., the number of CTUs in a frame), the second parameter in the scheme is used, which is the number of CTUs with constrained maximum coding tree depth, represented as $N_c^2$. This step is shown in lines 09-14 of Fig. 5.18.

Constrained CTUs are distributed according to the R-D costs of co-localised CTUs in the previous frame as follows. After encoding each CTU, the R-D costs of their CUs are summed up to obtain the CTU R-D cost. When every CTU is finally encoded, the CTU R-D costs are sorted in ascending order (line 17 of Fig. 5.18). CTUs at the beginning of the sorted list are those with lowest R-D cost and are thus less probable to yield high bit rates or distortions if constrained encoding is applied to them in the next frame. On the contrary, CTUs at the end of the list are those with highest bit rates and/or distortions, requiring an unconstrained encoding process to improve R-D performance. Therefore, in the proposed method, the $N_c^k$ CTBs with smallest R-D cost in the sorted list are encoded as constrained, while the remaining CTUs are encoded as unconstrained. This process is described in lines 18-23 of Fig. 5.18.

$$\alpha^{GOP} = \frac{T_e^{GOP}}{T_t^{GOP}} \qquad \text{(Eq. 48)}$$

$$N_c^k \leftarrow \alpha^{GOP} \cdot N_c^k \qquad \text{(Eq. 49)}$$

```
01  start
02  calculate $T_t^{GOP}$
03  start a new GOP
04  for each $j$ from 0 to $nFR$
05    for each $i$ from 0 to $nCTU$
06      mark CTU $i$ as unconstrained
07      encode CTU $i$
08    if last frame go to line 01
09  calculate $T_e^{GOP}$
10  calculate $\alpha^{GOP}$
11  if $N_c^1 < nCTU$
12    calculate new $N_c^1$
13  else
14    calculate new $N_c^2$
15  start a new GOP
16  for each $j$ from 0 to $nFR$
17    sort CTUs in ascending order of R-D cost
18    for each $i$ from 0 to $nCTU$
19      if $i < N_c^k$
20        mark CTU $i$ as constrained
21      else
22        mark CTU $i$ as unconstrained
23      encode CTU $i$
24    if last frame go to line 01
25  go to line 09
```

Fig. 5.18: Pseudo-code for the CCUPU method.

### 5.6.4 Results for CCUPU and Performance Evaluation

The performance of the CCUPU method was evaluated by measuring the trade-off between processing time and R-D efficiency. The same video sequences listed in section 5.6.1 were used in the tests, as well as the *Low Delay P* temporal encoder configuration, HM – version 9.2 (HM9.2), and QPs 27, 32, 37, and 42. For simulation purposes, the target processing times are defined as a percentage of the fully unconstrained case, i.e., when complexity scalability is not used. The tested target times were calculated corresponding to 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90% of the average processing time per frame when no complexity constraint is used for a sequence.

Complexity scalability accuracy was measured by comparing the target processing times to the actual times spent for encoding the video sequences. Table 5.10 shows average results for the sequences with different spatial resolutions, encoded under the different target complexities. In Table 5.10, *TT* corresponds to the target time per frame and *ET* is the average elapsed time per frame, both of which are presented as percentages of the time when no complexity scaling is applied. Both BD-BR (which stands for the BD-rate) and BD-PSNR values were calculated using the original HM encoder with no complexity constraining as reference. Average results considering the five resolutions tested are presented in Table 5.11.

A comparison between the values at the *TT* and *ET* columns in Table 5.10 and Table 5.11 leads to the conclusion that the algorithm is capable of scaling computational complexity quite accurately, since the target and elapsed time are very close in most cases. This is clearer in Fig. 5.19, which presents charts with *TT* values on the horizontal axis and *ET* results on the vertical axis for all the video sequences with four different QPs and nine target times. These figures show that for almost all the 360 tests executed in these experiments, the proposed method was able to scale processing times with high accuracy, i.e., with small differences between *TT* and *ET*. The only *TT* value which was not achieved for all video sequences was the 10% case, setting the lower limit of the achievable computational complexity to values between 10% and 20% of the unconstrained case. For this reason, the discussion presented in the next paragraphs will focus on the results obtained when *TT* is set between 20% and 90%.

Table 5.10: Average complexity and R-D results for CCUPU considering five different spatial resolutions.

| TT (%) | 416×240 | | | 832×480 | | | 1280×720 | | | 1920×1080 | | | 2560×1600 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ET (%) | BD-BR (%) | BD-PSNR (dB) | ET (%) | BD-BR (%) | BD-PSNR (dB) | ET (%) | BD-BR (%) | BD-PSNR (dB) | ET (%) | BD-BR (%) | BD-PSNR (dB) | ET (%) | BD-BR (%) | BD-PSNR (dB) |
| 90 | 96 | 0.04 | 0.00 | 95 | 0.05 | 0.00 | 92 | 0.01 | 0.00 | 91 | 0.02 | 0.00 | 91 | 0.04 | 0.00 |
| 80 | 86 | 0.40 | -0.02 | 82 | 0.21 | -0.01 | 82 | 0.06 | 0.00 | 81 | 0.10 | 0.00 | 82 | 0.41 | -0.02 |
| 70 | 73 | 0.95 | -0.04 | 71 | 0.47 | -0.02 | 71 | 0.34 | -0.03 | 71 | 0.24 | -0.01 | 71 | 0.27 | -0.01 |
| 60 | 62 | 1.49 | -0.06 | 61 | 0.79 | -0.04 | 59 | 0.81 | -0.07 | 61 | 0.45 | -0.01 | 61 | 0.44 | -0.02 |
| 50 | 53 | 2.12 | -0.09 | 51 | 1.57 | -0.07 | 50 | 1.28 | -0.10 | 51 | 0.79 | -0.03 | 51 | 0.64 | -0.02 |
| 40 | 45 | 6.83 | -0.27 | 41 | 7.48 | -0.33 | 42 | 2.00 | -0.15 | 41 | 1.79 | -0.06 | 42 | 1.82 | -0.06 |
| 30 | 35 | 19.9 | -0.75 | 31 | 19.2 | -0.81 | 32 | 7.47 | -0.53 | 31 | 6.00 | -0.19 | 32 | 5.22 | -0.18 |
| 20 | 27 | 33.5 | -1.23 | 22 | 32.6 | -1.30 | 23 | 22.5 | -1.41 | 21 | 14.5 | -0.43 | 22 | 10.1 | -0.35 |
| 10 | 24 | 40.7 | -1.47 | 15 | 51.8 | -1.92 | 14 | 55.9 | -2.94 | 13 | 31.4 | -0.86 | 14 | 17.6 | -0.61 |

According to the results presented in Table 5.10 and Fig. 5.19, the only cases which do not accurately scale computational complexity are the 416×240 video sequences. More specifically, the method was not able to scale computational complexity for the *RaceHorses1* sequence under 43% for QP 27, 28% for QP 32, and 22% for QP 37, as shown in Fig. 5.19 (a-c). This happens most likely because this sequence is the one with smallest spatial and temporal resolution among all the video sequences tested, resulting on an encoding process which is much faster in comparison to the other video sequences. As the encoding time is already small for the *RaceHorses1* sequence, the proposed method is not able to scale it down to values around 15%, as in the other sequences. As Table 5.10 shows, the average *ET* results for 416×240 sequences do not scale under 24% and only when *TT* is set to values above 40% the method yields appropriate *ET* results. We can conclude from this analysis that the proposed method is more accurate for scaling the complexity of high-resolution video, which is in fact the target of the HEVC standard and the case where it is more important to reduce and scale encoding complexity due to its larger absolute value.

Concerning the effect of the complexity scaling on encoding efficiency, Table 5.11 shows that the R-D performance tends to decrease significantly only when small target times are set. This happens because small target times incur in the use of a larger number of constrained CTUs, which are not optimally encoded through full RDO. On the one hand, for most target times these variations are negligible, more specifically from the 90% to the 50% case, which presented a BD-PSNR varying between 0 and 0.06 dB and a BD-rate increase between 0.03% and 1.28%. On the other hand, when the target time is set to values from 40% to 20%, the computational complexity savings cause a BD-PSNR decrease between 0.18 and 0.94 dB and a BD-rate increase between 3.98% and 22.64%.

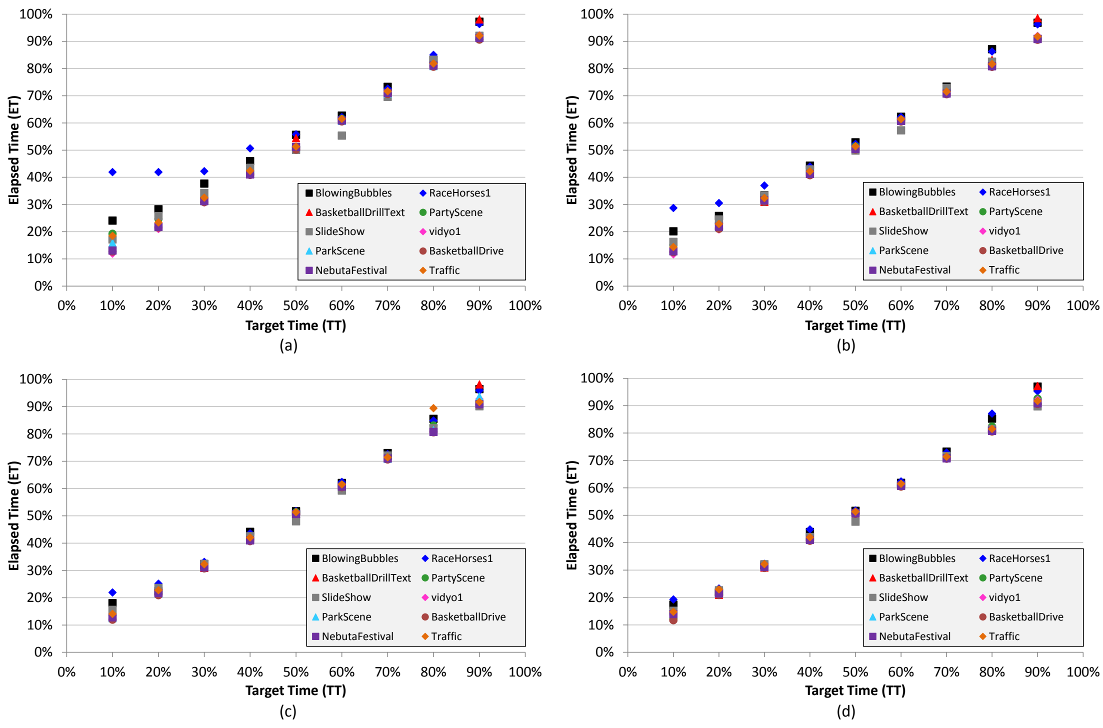


Fig. 5.19: Complexity scalability accuracy for 10 video sequences using QP 27 (a), QP 32 (b), QP 37 (c), and QP 42 (d).

Table 5.11: Average complexity and R-D results for CCUPU.

| TT (%) | ET (%) | BD-BR (%) | BD-PSNR (dB) |
|---|---|---|---|
| 90 | 93 | 0.03 | 0.00 |
| 80 | 83 | 0.23 | -0.01 |
| 70 | 72 | 0.46 | -0.02 |
| 60 | 61 | 0.80 | -0.04 |
| 50 | 51 | 1.28 | -0.06 |
| 40 | 42 | 3.98 | -0.18 |
| 30 | 32 | 11.55 | -0.49 |
| 20 | 23 | 22.64 | -0.94 |
| 10 | 16 | 39.47 | -1.56 |

These two operational regions are clearly defined in Fig. 5.20, which shows the effect of the computational complexity scalability on the encoder R-D performance. Just as in Table 5.10, the figure shows separate results for the different spatial resolutions. The first operational region, at the right side of the vertical dashed line, presents computational complexities ranging from 50% to 100% and provides an encoding process in which the R-D performance is minimally affected by the complexity constraining algorithm. The second region, at the left side of the vertical dashed line, allows further scaling of the computational complexity (down to 20%) at the cost of higher R-D performance losses. Notice, however, that even though these losses are not negligible, they are acceptable in many applications and situations that require video encoding with very low computational complexity and tolerate some image quality decrease as a trade-off. Fig. 5.20 also shows that the proposed method yields better R-D performance when scaling computational complexity of high-resolution video encoding.

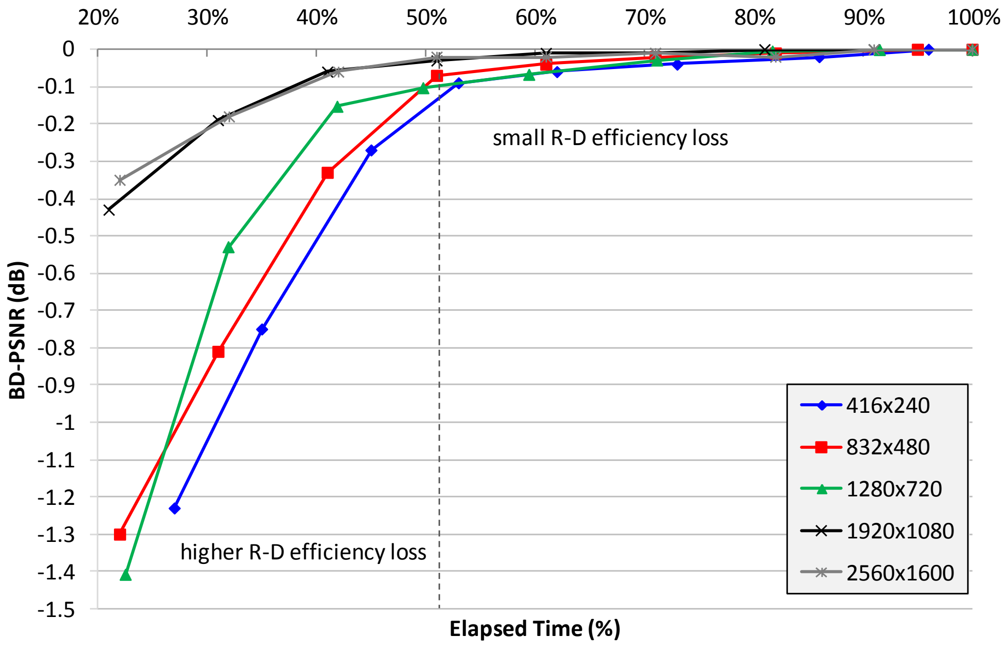
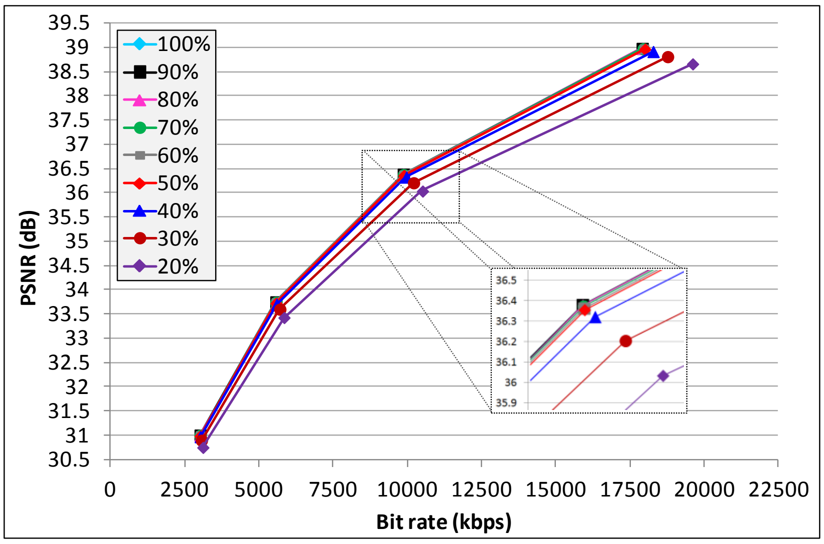
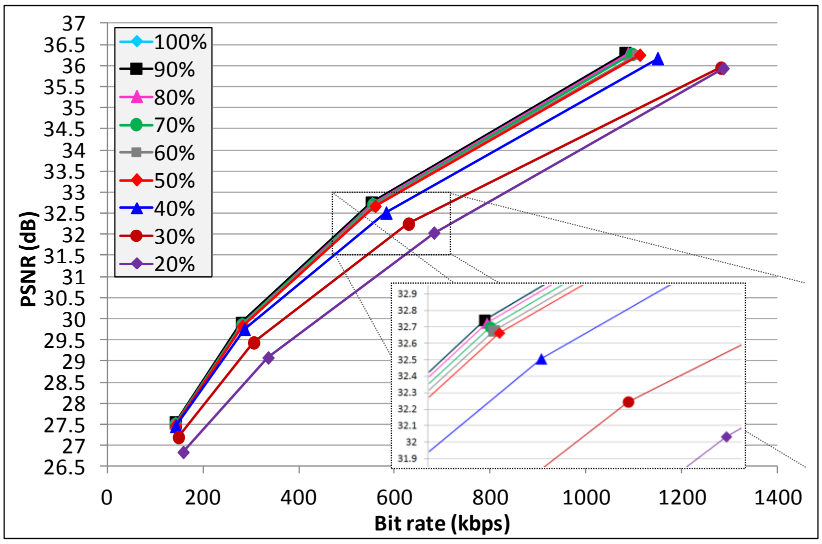


Fig. 5.20: Effect of computational complexity scalability over the R-D performance.

The R-D performance for each target complexity tested is presented in Fig. 5.21 for the best and worst cases, which correspond to the *Traffic* (Fig. 5.21(a)) and *RaceHorses1* (Fig. 5.21(b)) sequences, respectively. In both cases, it is possible to notice that the difference between results for target complexities between 50% and 100% is very small, especially in Fig. 5.21(a). In fact, the R-D results are so similar that these curves are only visible individually when a zoom is applied to the chart, as shown in the dotted box in both charts. When smaller target complexities are used (20%, 30% and 40%), the corresponding curves are more distant from each other due to larger R-D performances losses.



(a)



(b)

Fig. 5.21: R-D performance for each target complexity for videos (a) *Traffic* and (b) *RaceHorses1* encoded with QPs 27, 32, 37, 42.

Fig. 5.22 (a-d) presents the average encoding time per frame in each GOP for four video sequences (*PartyScene*, *vidyo1*, *BasketballDrive*, *Traffic*, respectively), each one encoded with a different QP (27, 32, 37, and 42, respectively). Besides the case in which no complexity scalability is applied (continuous black line at the top, labelled as "100%"), the 20% (red), 40% (green), 60% (blue) and 80% (grey) cases are presented in the charts together with their respective target times (dashed lines in the same colour). The remaining target times tested are not shown in these figures for the sake of clarity, but they exhibit similar behaviour. As it can be seen, in all cases the actual encoding times vary a little around the target times from GOP to GOP. This is a normal effect caused by the variation of the video signal characteristics over time, which requires different encoding operations and different computational resources. Variations in encoding time can also be noticed when no complexity scalability is used (100% case). The other video sequences presented behaviour similar to that shown in Fig. 5.22. The average difference between encoding time and target time for the four sequences presented in Fig. 5.22 are 4.6%, 1.3%, 2.3%, and 0.94%, respectively.

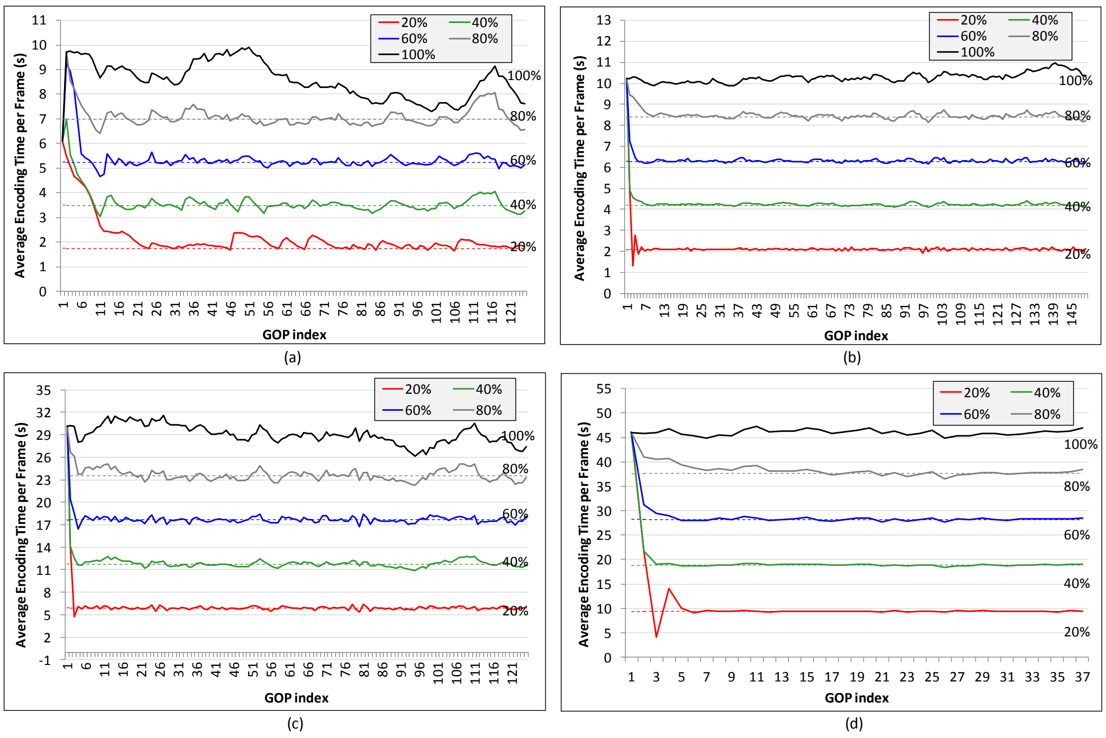


Fig. 5.22: Average encoding time per frame in each GOP for different target complexities: (a) *PartyScene*, QP 27; (b) *vidyo1*, QP 32; (c) *BasketballDrive*, QP 37; (d) *Traffic*, QP 42.

The specific case of the *RaceHorses1* sequence, which was the only exception in the results presented in Table 5.10 and Fig. 5.19, is shown in Fig. 5.23 for QP 32. It is possible to perceive that even though the encoding times do not scale down to values as close to the target times as in the other sequences, the method is still capable of decreasing the encoding time to levels not too far from the target. The average difference between encoding and target time for such sequence was 15%.

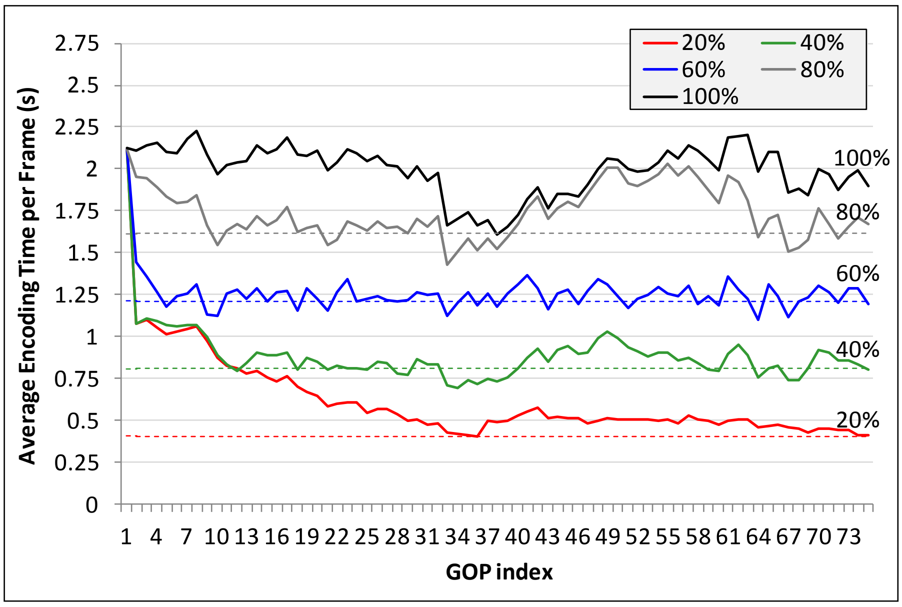


Fig. 5.23: Average encoding time per frame in each GOP for different target complexities, considering video *RaceHorses1* and QP 32.

## 5.7 Conclusions

The contribution of this chapter consists in a set of algorithms for scaling the computational complexity of an HEVC encoder. The R-D efficiency as well as the complexity scaling accuracy of each method was separately presented and discussed in their corresponding sections. In this section, the five methods are compared and their results are jointly discussed. The average results presented in Table 5.1, Table 5.2, Table 5.4, Table 5.6, and Table 5.11 for the FDCS, VDCS, MCTDL, CTDE and CCUPU methods, respectively, were all combined in the charts presented in Fig. 5.24, Fig. 5.25 and Fig. 5.26 in terms of complexity scaling accuracy, BD-rate increase and BD-PSNR decrease, respectively.

Fig. 5.24 shows that, in general, the complexity scaling accuracy of the five methods is quite similar, since the resulting running complexity is always close to the ideal case (dotted grey line). The figure also shows that the computational complexity can be scaled down to targets lower than 60% with the CCUPU method (dashed portion of the black curve), which is not possible with the remaining algorithms. The largest difference between target and running complexity appears, as already mentioned, when a target complexity of 10% is set for the CCUPU method. In that case, the complexity achieved is around 16%, which is still close to the target.

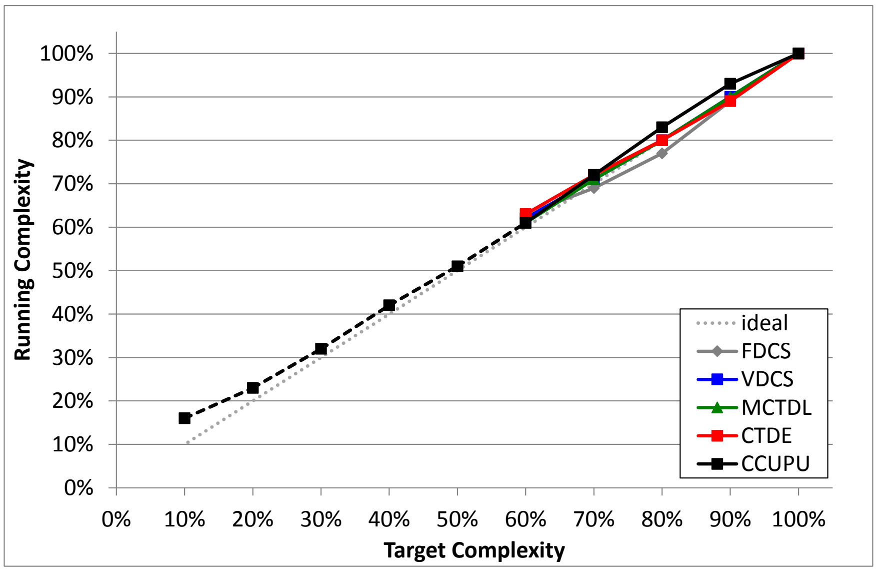


Fig. 5.24: Target complexity versus running complexity for the complexity scaling methods.

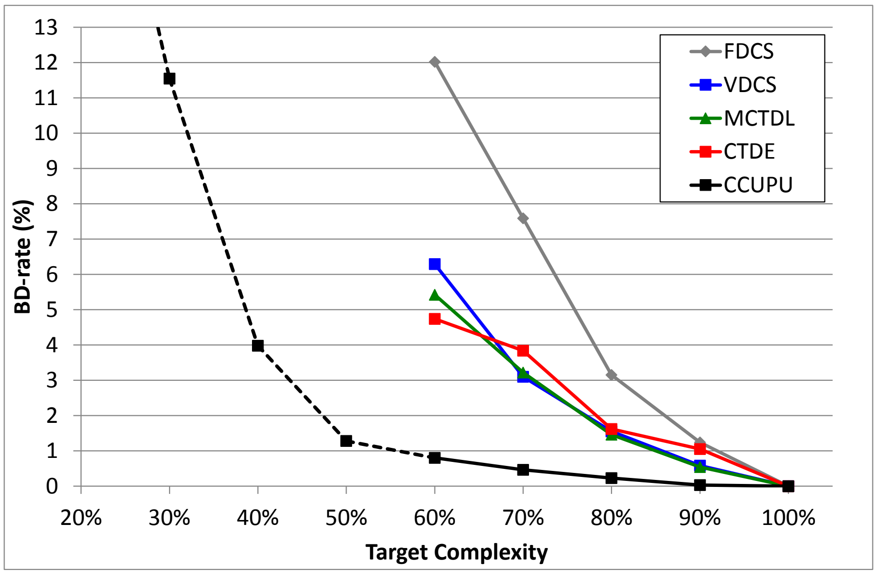


Fig. 5.25: BD-rate increase for the five complexity scaling methods.

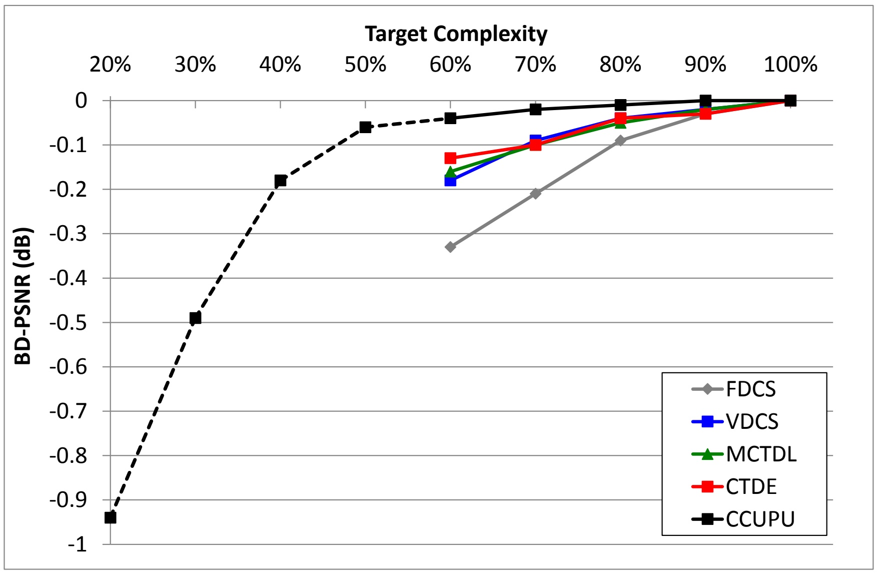


Fig. 5.26: BD-PSNR for the five complexity scaling methods.

Fig. 5.25 presents the BD-rate increase caused by applying different target complexities to the five methods. It is possible to notice from the first method developed (FDCS) to the last one (CCUPU) an increasing R-D efficiency, which is an outcome obtained by adding up more intelligent ways of constraining the frame partitioning structures of HEVC. In FDCS the maximum Coding Tree depth was constrained for the whole frame, which is a rather crude way of scaling the computational complexity. In contrast, in CCUPU the encoding of each CTU is independently adjusted and up to two levels of constraining can be performed per CTU. This reflects directly in the BD-rate and BD-PSNR results presented in Fig. 5.25 and Fig. 5.26. For example, when the target complexity is set to 50%, the BD-rate increase caused by the CCUPU method is smaller than the BD-rate increase caused by FDCS with target complexity set to 90%. Similarly, the CCUPU method is able to reduce complexity to 30% at a smaller BD-rate cost than FDCS in 60%.

The results compared in this section indicate that considering spatial and temporal correlation of video sequences in the complexity scaling algorithms (as done in VDCS, MCTDL and CTDE) decreases the R-D efficiency loss caused by FDCS. Furthermore, when constraining the PU splitting mode decision process before constraining the Coding Tree decision (as done in CCUPU), the computational complexity was further reduced at a much smaller R-D efficiency loss. These results show that the HEVC encoding complexity can be efficiently adjusted at small or negligible R-D losses if smart approaches are employed in the constraining process. If more information about the original video sequence characteristics, such as motion activity profile and texture complexity, as well as intermediate results obtained during the encoding process, were considered when adjusting the encoder complexity, R-D efficiency losses even smaller than those of CCUPU could be achieved. The research work presented in the next chapter 6 furthers these ideas by developing a new complexity reduction approach, which achieves even better results.

# Chapter 6

# Computational Complexity Reduction Using Data Mining

As the experiments presented in chapter 4 have shown and the complexity scaling methods proposed in chapter 5 have confirmed, an important share of the high computational complexity of HEVC comes from the use of very flexible partitioning structures, such as the CUs, the PUs and the TUs. This chapter describes the process of using data mining (DM) techniques to build a set of decision trees that are used to decide if the RDO-based partitioning structure decision process should be terminated early or run to its full extent. By using information from intermediate encoding variables collected during the encoding of a set of video sequences, a set of decision trees were built and implemented in the HM encoder. When using this modified encoder, the operation of the decision trees sidesteps the encoder from having to run the full RDO process to find the best partitioning structures. The study of correlations and information gains associated with each variable, recorded while encoding test videos with the original HM encoder, was essential to the development of the early termination schemes presented in this chapter.

As explained in previous chapters, among the three main partitioning structures of HEVC, CUs have a central role due to their interdependence with the remaining partitioning structures. This means that by changing the number of Coding Tree configurations tested for a CTU, the overall number of tests performed to define PUs and RQTs is also affected. For this reason, the first early termination investigated in this chapter focuses on the Coding Tree determination process. The second early termination method proposed focuses on the PU splitting mode decision, since the experiments presented in section 5.6.1 have shown that limiting the PU splitting mode decision results in significant computational complexity decreases with small compression efficiency losses. Finally, the third early termination focuses on the RQT decision, which is the partitioning structure that presents the smallest impacts on the encoding computational complexity.

When separately implemented, the early termination schemes achieved an average computational complexity reduction varying from 7.2% up to 50% for a negligible encoding performance reduction ranging from 0.05% up to 0.56% in terms of BD-rate increase. When jointly implemented, they permitted an average computational complexity reduction of up to 65% is achieved, with a small BD-rate increase of 1.36%. Extensive experiments and comparisons demonstrate that the proposed schemes achieve the best R-D-C trade-offs among all comparable works.

Part of the work presented in this chapter was published in [19-22].

## 6.1 Introduction to Data Mining and Decision Trees

*Knowledge Discovery from Data* (KDD) is an interdisciplinary subfield of computer science currently applied to several areas, such as medicine, market management, biology and image processing. The goal of KDD systems is to extract information from both structured and unstructured sources by using DM and machine learning algorithms. In the work presented in this chapter, a predictive DM approach is used.

Predictive DM techniques are used to determine the value of dependent variables by looking at the value of some attributes in the data set, identifying regularities and building generalisation rules that can be expressed as models. There are several methods of predictive DM currently available, which vary broadly from one another in terms of efficiency, complexity and applicability. Decision trees [126] are a type of commonly used predictive DM, in which a dependent variable can assume one among a finite number of outcomes. In classification trees, which are a specific type of decision trees used in the work presented in this chapter, the dependent variable is called the *class* attribute and it can take a finite number of outcomes.

When building decision trees, observations on a set of training data are mapped into arcs and nodes, as shown in the example given in Fig. 6.1. The inner nodes ($A$, $B$, $C$, *D*, in Fig. 6.1) represent the variables (attributes) tested, while the arcs are the possible values that the attributes can assume. In a binary classification tree, such as those designed and used in this work, the attributes can assume two results (*x1* and *y1* for attribute *A* in the example of Fig. 6.1). Finally, the leaves of decision trees are the values that the class attribute can assume and represent the possible outcomes of the whole decision process. In the example given in Fig. 6.1, the possible outcomes are *L* and *R*.

When the decision tree is implemented, in order to classify a determined instance into one of the possible outcomes the algorithm starts at its root (*A*, in Fig. 6.1), tests the corresponding attribute value and descends to the next node through the appropriate branch, depending on the test result. For example, a data instance with feature vector (*A* = *x1*, *B* = *x2*, *C* = *y3*, *D* = *y4*) would be classified as *R* in the decision tree of Fig. 6.1.

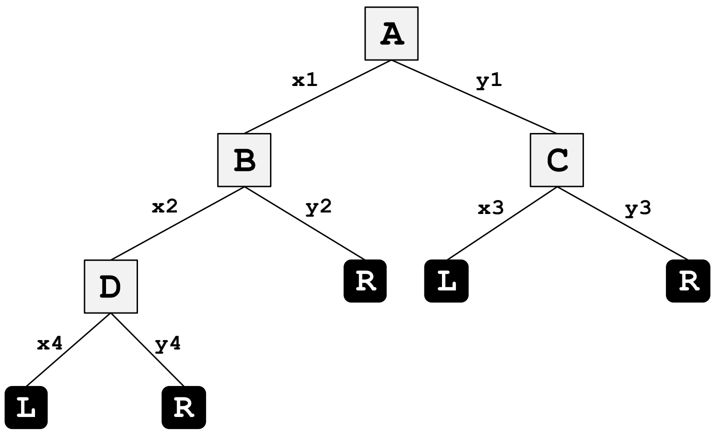


Fig. 6.1: Example of a binary classification tree.

Decision trees are commonly used mainly because of the following characteristics:

1) They usually achieve high prediction accuracy after trained;
2) They are easily understood by human beings and therefore simple to be implemented;
3) There are many efficient algorithms to build them from training data;
4) They can deal with both categorical and numerical values and
5) Once implemented, they execute predictions very fast.

The first and the fifth characteristics above are extremely important for the research work presented in this chapter, which aims at reducing the computational complexity of HEVC encoders without harming its R-D efficiency. This can only be performed if good prediction accuracy is achieved with the obtained decision trees and if the prediction process adds negligible extra computational complexity to the encoder.

## 6.2 Methodology

The *Waikato Environment for Knowledge Analysis* (WEKA) [127], version 3.6, was used to aid the DM process described in this chapter. WEKA is a free, open-source DM tool that includes several machine learning algorithms for pre-processing, classifying, clustering, and visualising the data set, supporting statistical evaluation of learning schemes. Data fed to WEKA was obtained through offline encodings of the following 10 video sequences, which characteristics can be found in Appendix B: *BlowingBubbles*, *RaceHorses1*, *PartyScene*, *BQMall*, *SlideShow*, *vidyo1*, *BasketballDrive*, *ParkScene*, *NebutaFestival*, and *Traffic*. All video sequences were encoded with QPs 22, 27, 32, 37, and 42, using the *Random Access* temporal configuration [43]. The HM software – version 12 (HM12) – was modified to save internal variables with intermediate encoding results into files that were used to create the training files used by WEKA.

The input for WEKA are ARFF (*Attribute-Relation File Format*) files, containing plain text describing a list of instances sharing a set of attributes. Fig. 6.2 shows an ARFF file example, with its two major sections separated by dashed boxes. The first section consists of a header with the name of the relation and the attributes declaration (i.e., name and type of values of that attribute). The second section contains the training data with one instance per line and one attribute value per column. In the specific case of building decision trees, the last line of the first section identifies the class attribute, for which the machine learning algorithms try to find a general (prediction) rule. In the example of Fig. 6.2, *SplitCU* is a class attribute that can assume a binary value (i.e., either 0 or 1). This is also the case of all decision trees proposed in this chapter.

```
@RELATION CodingTreeEarlyTermination                               Header

@ATTRIBUTE RDcost_MSM NUMERIC
@ATTRIBUTE RDcost_2Nx2N NUMERIC
@ATTRIBUTE RDcost_2NxN NUMERIC
@ATTRIBUTE RDcost_Nx2N NUMERIC
@ATTRIBUTE part {0,1,2,3,4,5,6,7}
@ATTRIBUTE MergeFlag {0,1}
@ATTRIBUTE SkipMergeFlag {0,1}
@ATTRIBUTE neighDepth NUMERIC
@ATTRIBUTE SplitCU {0,1}
```

```
@DATA                                                            Raw Data

839   3214  2801  2801  0     1     1     0.25        1.75        0
3920  5055  4421  4421  0     0     0     2.13542     0.13541     1
990   2617  2687  2148  0     1     1     0.875       1.125       0
...
5505  5001  5307  4895  7     1     1     1.95833     0.041667    1
```

Fig. 6.2: Example of an ARFF file.

To reduce the problem of the class data imbalance, which occurs when there are significantly more training instances belonging to one class than to the other(s), following common practice [128], the ARFF files used in the evaluation are composed of data sets with half the instances classified into each class (e.g., 50% classified as 0 and 50% classified as 1 in the example of Fig. 6.2). This was accomplished by resampling the training data instances when building the final ARFF files, which are composed of equal numbers of random samples of the data collected during the encoding of the 10 video sequences encoded with the five QPs used.

For each early termination scheme proposed in this chapter, several variables were recorded during execution of the HM encoder, such as the sum and the variance of luminance samples in a CU, the absolute sum and variance of prediction residues in a PU, the horizontal and vertical gradients in a possible PU edge, and the R-D cost of each PU splitting mode. The usefulness of each of these variables for the decision tree was assessed through the *Information Gain Attribute Evaluation* (IGAE) method in WEKA, which measures the information gain [129] achievable by using a variable to classify the data into the different classes represented in the data. This gain equates to the difference between the number of bits per data item necessary to convey its class identity before and after classification of the data set using decision rules based on the variable in question [126]. Therefore, the information gain of a variable indicates how relevant it is for the process of constructing a decision tree that correctly decides to which class each data item belongs. In the case of the WEKA software, this information gain is measured by the *Kullback-Leibler Divergence* (KLD) [130] of the pre and post-classification probability distributions. Based on this measure (IGAE), a manual analysis procedure was followed to identify the most useful variables for the (tree) decision processes. Then, the variables with higher information gain were selected as attributes for the tree training processes, as explained later in this chapter.

The training of the decision trees was performed with the *C4.5* algorithm [126], which also uses KLD to choose the best attribute for each decision step and the thresholds corresponding to each decision step. The *C4.5* algorithm starts by taking all instances fed to it as inputs and calculates the information gain of using each attribute to perform the classification using a determined threshold. By iterating among all attributes and adjusting the thresholds, *C4.5* measures the information gain of each variable and threshold pair. Then, the attribute (and its corresponding threshold) with the largest information gain is chosen to divide the training data into two sub-sets. The same process is applied recursively to the two sub-sets. More details on *C4.5* can be found in [126].

The accuracy of all obtained trees was measured with WEKA by applying a 10-fold cross-validation process. The level of accuracy was measured by the percentage of correct decisions in the total amount of instances used in the training process. Then, all trees obtained in the training phase were implemented in the HM software following a scheme of tests designed to allow a clear evaluation of each decision tree performance either individually or combined with others. A total of seven low-complexity schemes were implemented and the encoding R-D efficiency was measured using sequences different from those used in the training phase. The seven schemes and their respective acronyms are:

1) Early termination for determining Coding Trees (*CT ET*) [19, 20];
2) Early termination for determining Prediction Units (*PU ET*) [20-22];
3) Early termination for determining Residual Quadtrees (*RQT ET*);
4) Joint early terminations for determining Coding Trees and Prediction Units (*CT+PU ET*) [20];
5) Joint early terminations for determining Prediction Units and Residual Quadtrees (*PU+RQT ET*) [20];
6) Joint early terminations for determining Coding Trees and Residual Quadtrees (*CT+RQT ET*) [20] and

7) Joint early terminations for determining Coding Trees, Prediction Units and Residual Quadtrees (*CT+PU+RQT ET*) [20].

## 6.3 Early Termination for Determining Coding Trees

The proposed Coding Tree early termination [19, 20] consists in deciding whether or not the splitting of CUs into four smaller CUs should be tested. If the decision tree outcome is *yes* (i.e., *SplitCU* = 1), the current CU is split into four smaller CUs and the next Coding Tree depth is tested. Otherwise (i.e., *SplitCU* = 0), the Coding Tree splitting process is halted in the current CU.

As the HEVC standard allows up to four Coding Tree depths, three different decision trees were created for the cases that allow splitting into smaller CUs: 64×64, 32×32, and 16×16. Table 6.1 shows the HM variables that provided best performance as measured by the information gain with respect to the decision of splitting or not splitting a CU (i.e., those that were selected as attributes for the decision trees). Information gain values are presented separately for each CU size in the table and the attributes are described in detail in the following paragraphs.

The *Partition* attribute corresponds to which PU splitting mode was chosen for the current CU (i.e., *2N×2N*, *2N×N*, *N×2N*, *N×N*, *2N×nU*, *2N×nD*, *nL×2N*, or *nR×2N*), independently of whether inter or intra-frame prediction was applied. The idea behind saving this information is that when a large PU (e.g., *2N×2N*) is chosen as the best option to predict a determined CU, further tests to determine the Coding Tree configuration are probably not necessary, so that this CU does not need being split into smaller sub-CUs. Statistics that support this claim are presented in Fig. 6.3. The chart shows that most of the CUs predicted as a *2N×2N* PU were not split into sub-CUs. For example, 83% of 64×64 CUs predicted as a *2N×2N* PU did not need being split into four 32×32 CUs, as the leftmost black bar of Fig. 6.3(a) shows. Conversely, an average of 83.3% of 64×64 CUs encoded with the remaining modes were split into four 32×32 CUs (average of all grey bars of Fig. 6.3(a), except for the *2N×2N* case). By analysing the three charts of Fig. 6.3, it is possible to notice that, on the one hand, the correlation between using *2N×2N* PUs and not splitting the CU decreases in smaller CUs (70% for 16×16 CUs, as shows Fig. 6.3(c)). On the other hand, in smaller CUs the correlation between choosing the remaining PU modes and splitting the CU increases (on average, 90% for 16×16 CUs).

Table 6.1: Information Gain Attribute Evaluation for the Coding Tree early termination.

| **Attribute** | **Information Gain** | | |
|---|---|---|---|
| | **64×64** | **32×32** | **16×16** |
| *Partition* | 0.352 | 0.336 | 0.269 |
| *ΔNeighDepth* | 0.311 | 0.262 | 0.249 |
| *Ratio(2N×2N, MSM)* | 0.112 | 0.168 | 0.255 |
| *NormDiffRD(2N×2N, MSM)* | 0.109 | 0.163 | 0.249 |
| *RD(2N×2N)* | 0.035 | 0.042 | 0.053 |
| *RD(MSM)* | 0.034 | 0.061 | 0.108 |
| *RD(2N×N)* | 0.033 | 0.036 | 0.044 |
| *RD(N×2N)* | 0.031 | 0.032 | 0.042 |
| *SkipMergeFlag* | 0.046 | 0.066 | 0.065 |
| *MergeFlag* | 0.020 | 0.035 | 0.046 |

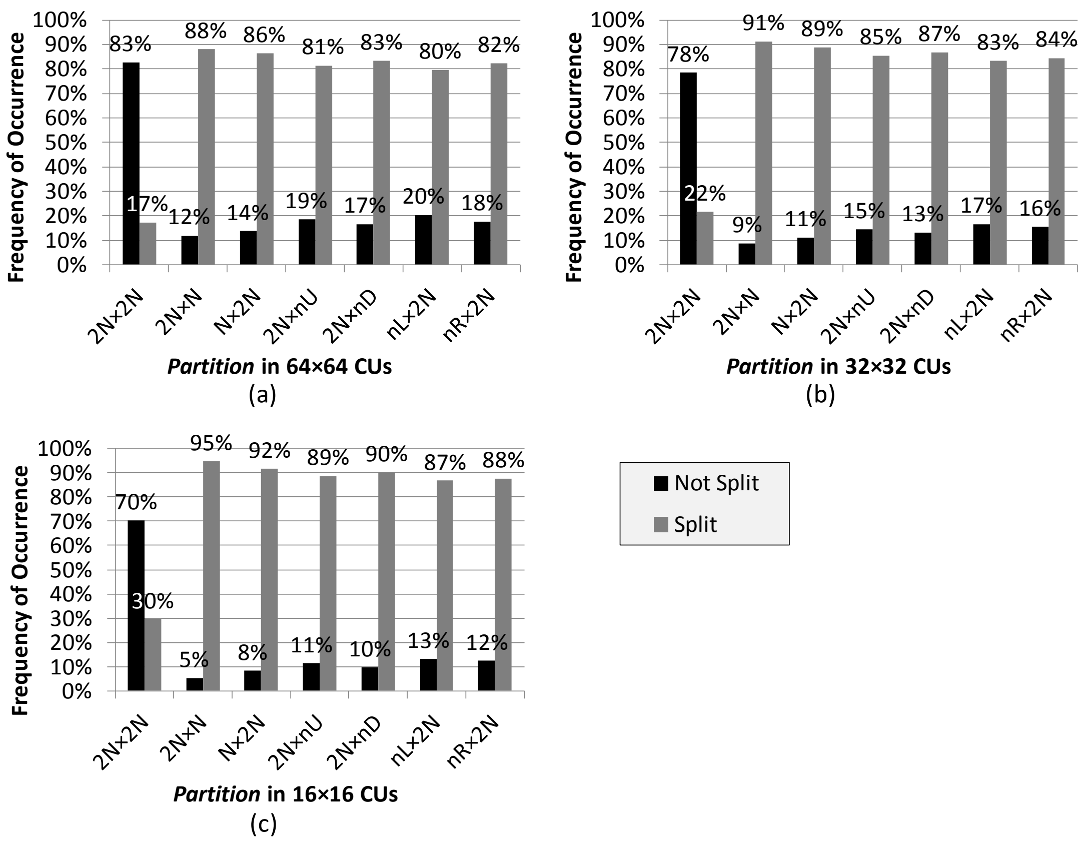


Fig. 6.3: Occurrence of (a) 64×64 CUs, (b) 32×32 CUs, and (c) 16×16 CUs split and not split into smaller CUs according to the PU mode chosen (i.e., *Partition*).

The *ΔNeighDepth* attribute is computed based on the difference between the Coding Tree depths used in neighbouring CTUs and the depth of the current CU. The rationale of considering such variable is that there exists a correlation among maximum depths of spatially and temporally neighbouring CTUs, as previously shown in chapter 5 (see Table 5.9). The attribute *ΔNeighDepth* is calculated as follows. First, for each neighbouring CTU, the average depth among all its composing CUs is computed. The top-left, top, top-right and left CTUs in the current frame, as well as the co-localised CTUs in the first frames of both reference lists (*List 0* and *List 1*) are considered as neighbours, so that up to six average depths are calculated. Fig. 6.4 shows the four neighbouring CTUs in the current frame and the co-localised CTUs in the reference frames, with the variables representing the average CU depths assigned to them. Let us call these averages as $A_1$ to $A_N$, where $N$ is the number of neighbouring CTUs available for the current CTU. The CTU assigned with label $C$ in Fig. 6.4 represents the current CTU. Finally, the value of *ΔNeighDepth* is calculated as an average of the averages $A_1$ to $A_N$, minus the depth of the CU currently being encoded, as shown in (Eq. 50). If the current CU depth is much smaller than the average of average depths of neighbouring CTUs, the splitting process should probably continue due to spatio-temporal correlation among neighbouring CTUs.

$$\Delta NeighDepth = \frac{\sum_{i=1}^{N} A_i}{N} - CurrDepth \qquad \text{(Eq. 50)}$$

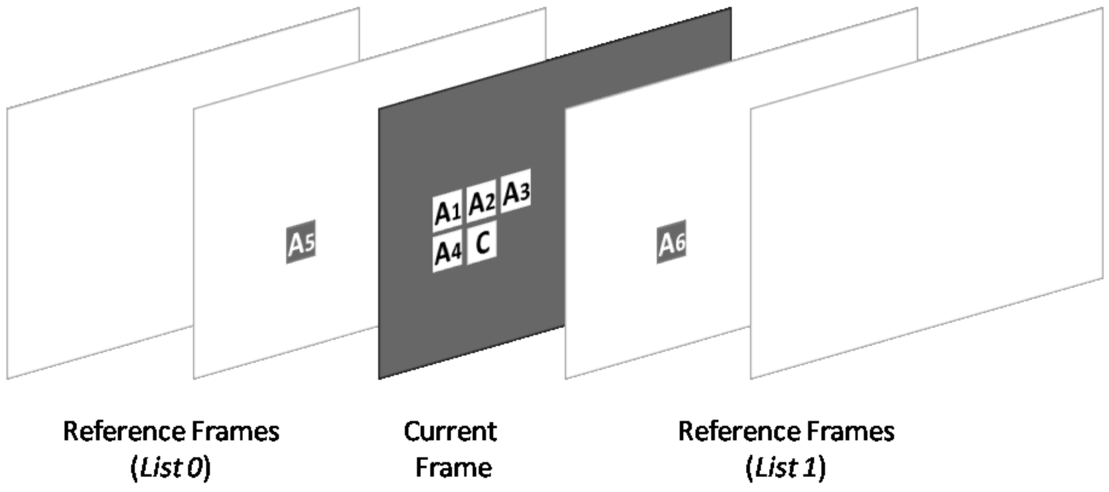


Fig. 6.4: Neighbouring CTUs used in the calculation of *ΔNeighDepth*.

Fig. 6.5 shows the distribution of *ΔNeighDepth* for different CU sizes. The curves show that there is a clear relationship between the distribution of *ΔNeighDepth* and the CU splitting decision. CUs that are not split into smaller CUs have *ΔNeighDepth* values that cluster towards low magnitudes, while the opposite occurs for those CUs that are split into smaller CUs. Since the two distributions do not fully overlap it is possible to determine an optimal decision threshold that minimises the classification error rate. WEKA computes these thresholds during the process of training the decision trees.

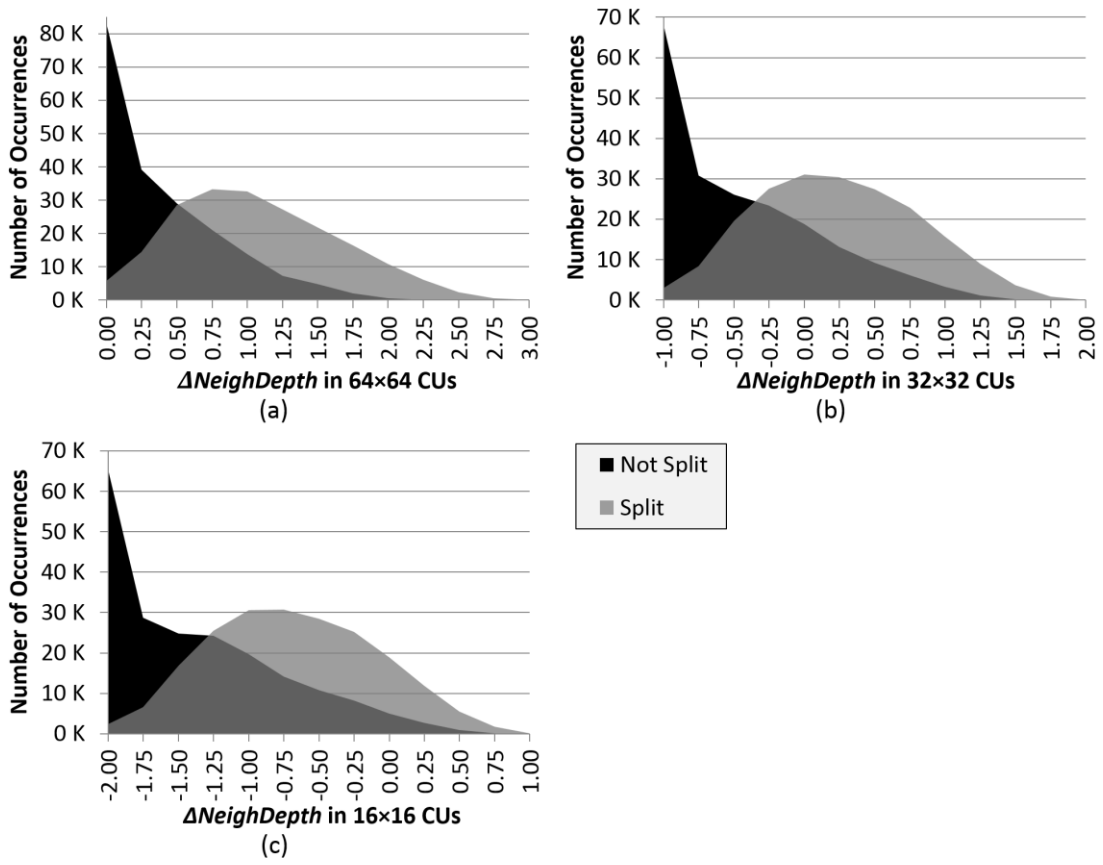


Fig. 6.5: Occurrence of (a) 64×64 CUs, (b) 32×32 CUs, and (c) 16×16 CUs split and not split into smaller CUs according to the average of CU depths in neighbouring CTUs.

The *Ratio(2N×2N, MSM)* shown in Table 6.1 is calculated as a simple division between the R-D costs of encoding the current CU as an inter-predicted *2N×2N* PU and as an MSM PU, as shown in (Eq. 51). The *NormDiffRD(2N×2N, MSM)* value is the normalised difference between the *RD(2N×2N)* and *RD(MSM)* costs, calculated as per (Eq. 52). The reason for considering these values in the IGAE analysis is that when a compression gain (i.e., a drop in R-D cost) is observed due to the use of motion-compensated prediction in a CU instead of encoding it with MSM, the block probably belongs to a medium/high-motion or complex-textured image region and usually in this type of situation it is advisable to split a CU into smaller CUs. Fig. 6.6 shows the distribution of *Ratio(2N×2N, MSM)* for different CU sizes. The smaller information gain level of this parameter in comparison to the two previously analysed cases is also clear in the charts, which shows the *Split* and *Not Split* areas more overlapped than in the charts of Fig. 6.3 and Fig. 6.5.

$$Ratio(2N \times 2N, MSM) = \frac{RD(2N \times 2N)}{RD(MSM)} \quad \text{(Eq. 51)}$$

$$NormDiffRD(2N \times 2N, MSM) = \left| \frac{RD(2N \times 2N) - RD(MSM)}{RD(MSM)} \right| \quad \text{(Eq. 52)}$$

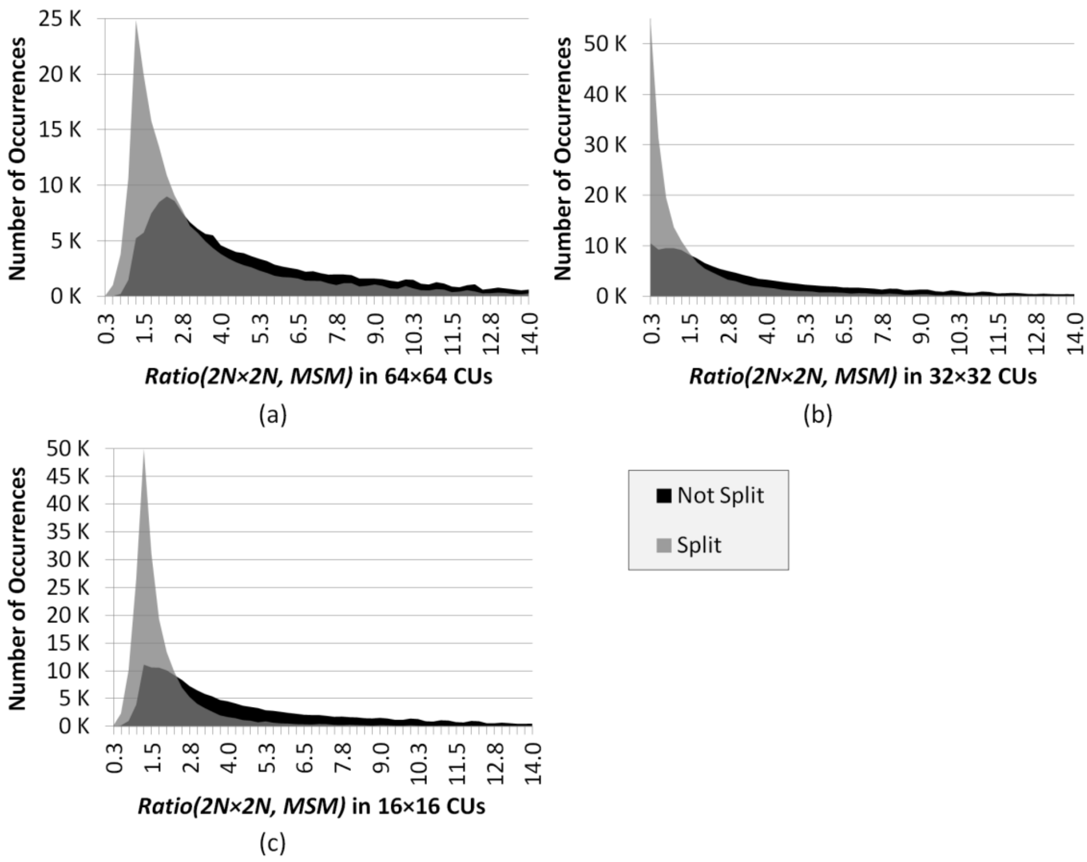


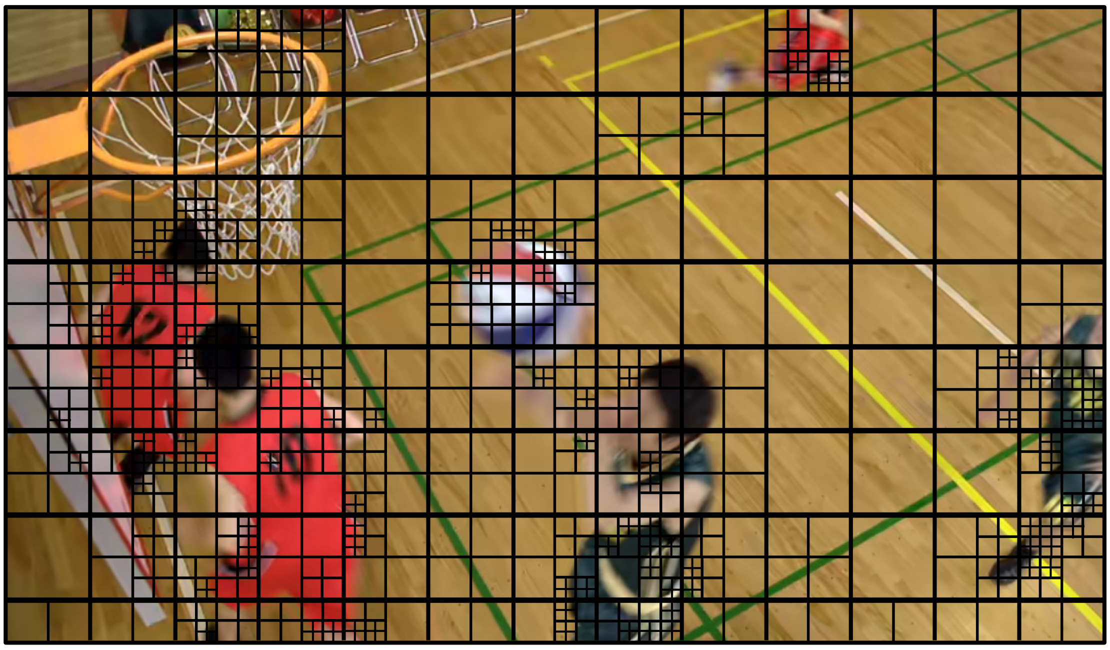
Fig. 6.6: Occurrence of (a) 64×64 CUs, (b) 32×32 CUs, and (c) 16×16 CUs split and not split into smaller CUs according to the ratio between the *2N×2N* and MSM R-D costs.

The decision trees were trained with the attributes shown in Table 6.1 and their most important characteristics are detailed in Table 6.2. The table presents the decision accuracy of each tree, as well as their depth (i.e., number of sequential tests), number of test nodes and number of leaves. As Table 6.2 shows, the three obtained trees achieve a decision accuracy slightly above 84%. The accuracies are measured by the ratio of the number of splitting decisions which agree with the splitting decision that would have been taken by the unmodified coder (both *Split* and *Not Split*) and the total number of CUs analysed in each case. However, these results count the case of splitting a CU that should not be split into smaller CUs as a decision error, even though it does not harm the encoding R-D efficiency, having as only deleterious effect an increase of the encoding complexity. In fact, the R-D efficiency is negatively affected by decision errors only when a CU that should be split is not split due to the early termination provided by the decision trees and so the CU is not as efficiently coded as it could be. The sixth column of Table 6.2 shows the percentage of such incorrect early terminations that could actually cause R-D efficiency losses due to inaccurate Coding Tree depth decisions. Regarding the topological characteristics of the decision trees, it is important to notice that all of them are composed of less than 10 decision levels (*Depth* column in Table 6.2), which means that the computational complexity added to the encoder due to implementing such trees is negligible. Detailed descriptions of the trees obtained after training are available in the Appendix C of this thesis.

Table 6.2: Characteristics and performance of trees trained for the Coding Tree early termination.

| **CU size** | **Depth** | **Test Nodes** | **Leaves** | **Decision Accuracy** | **Inaccurate Depth** |
|---|---|---|---|---|---|
| 64×64 | 5 | 6 | 19 | 84.2% | 7.1% |
| 32×32 | 8 | 20 | 33 | 84.5% | 7.5% |
| 16×16 | 9 | 23 | 44 | 84.6% | 6.9% |

Finally, to illustrate the effectiveness of the proposed method, Fig. 6.7 presents the 100th frame of the *BasketballDrill* video sequence and its corresponding CU boundaries according to the Coding Tree defined for each CTU. The sequence was encoded with QP 32 and the *Random Access* temporal configuration. Notice that the *BasketballDrill* video was not used in the training of the decision trees. The frame in Fig. 6.7(a) was encoded using the original HM encoder, while the frame in Fig. 6.7(b) was encoded with an HM encoder modified to include the Coding Tree early termination algorithm. It is possible to perceive that the differences between the boundaries in Fig. 6.7(a) and Fig. 6.7(b) are not expressive, which confirms that the early termination performs a correct Coding Tree determination in most cases.


(a)

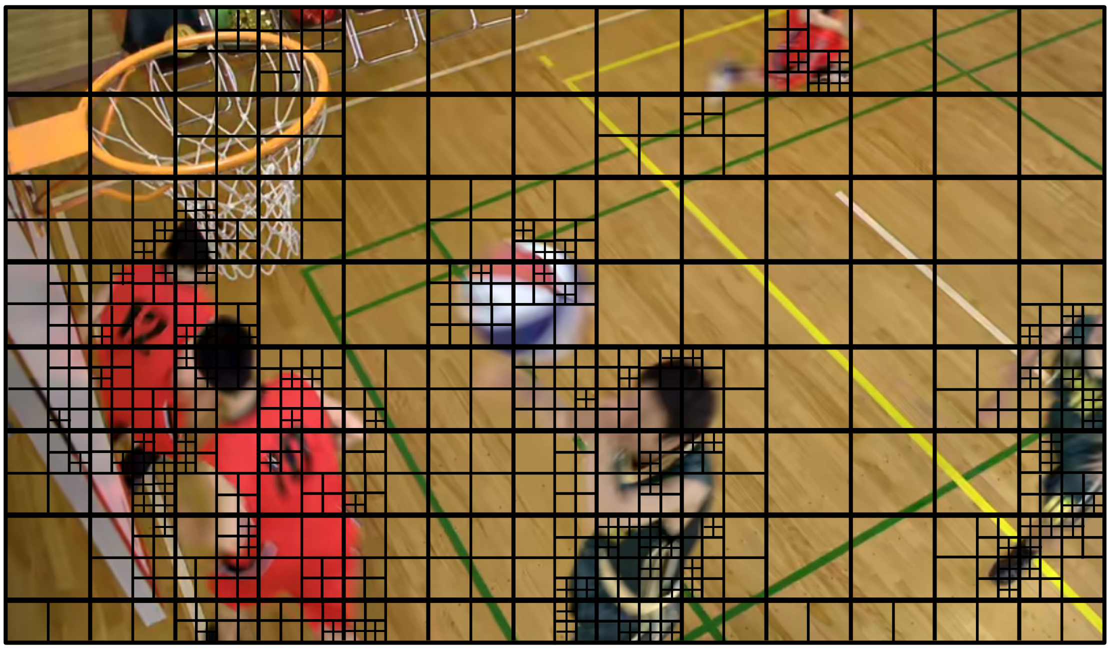

(b)

Fig. 6.7: CTUs divided into CUs in the 100th frame of the *BasketballDrill* video sequence encoded with QP 32 by (a) the original HM encoder and the (b) HM encoder with the Coding Tree early termination.

## 6.4 Early Termination for Determining Prediction Units

The second early termination proposed [20-22] is motivated by the statistics presented in Fig. 6.8, which shows the average occurrence probability of each PU splitting mode in inter-predicted CUs. The statistics in the charts are for the *BasketballDrive* video sequence encoded with QPs 27, 32, 37, and 42. The remaining sequences presented similar behaviour. In Fig. 6.8, it is clear that most inter-predicted CUs are encoded without being split into smaller PUs and employ mainly the MSM mode. On average, 58%, 76%, 89% and 95% of 64×64, 32×32, 16×16 and 8×8 CUs, respectively, are encoded with the MSM mode. However, even though the remaining modes are not so frequently used as MSM, especially in small CUs, they are still always tested in a full RDO-based decision, which is not ideal when trading off compression efficiency and computational complexity.

As in the case of early termination for Coding Trees presented in section 6.3, the decision trees for early terminating the PU splitting mode decision were designed to decide whether or not the search for the best PU structure should continue after some PU splitting modes have been tested. As most inter-predicted CUs are encoded as a single PU, the decision trees are used after testing the MSM and *2N×2N* modes, so that these two modes are always tested for every CU, as shown in the diagram of Fig. 6.9. In case of early termination (decision labelled as *E.T.* in Fig. 6.9), the mode with smallest R-D cost between MSM and *2N×2N* is chosen for the CU. Otherwise (decision labelled as *keep* in Fig. 6.9), the remaining modes are tested.

As in the case presented in section 6.3, 50% of the training data come from inter-predicted CUs that have been split into PUs smaller than *2N×2N* and 50% come from inter-predicted CUs that have not been split into PUs smaller than *2N×2N*. Four different decision trees were built, each one for a different CU size (64×64, 32×32, 16×16, and 8×8).

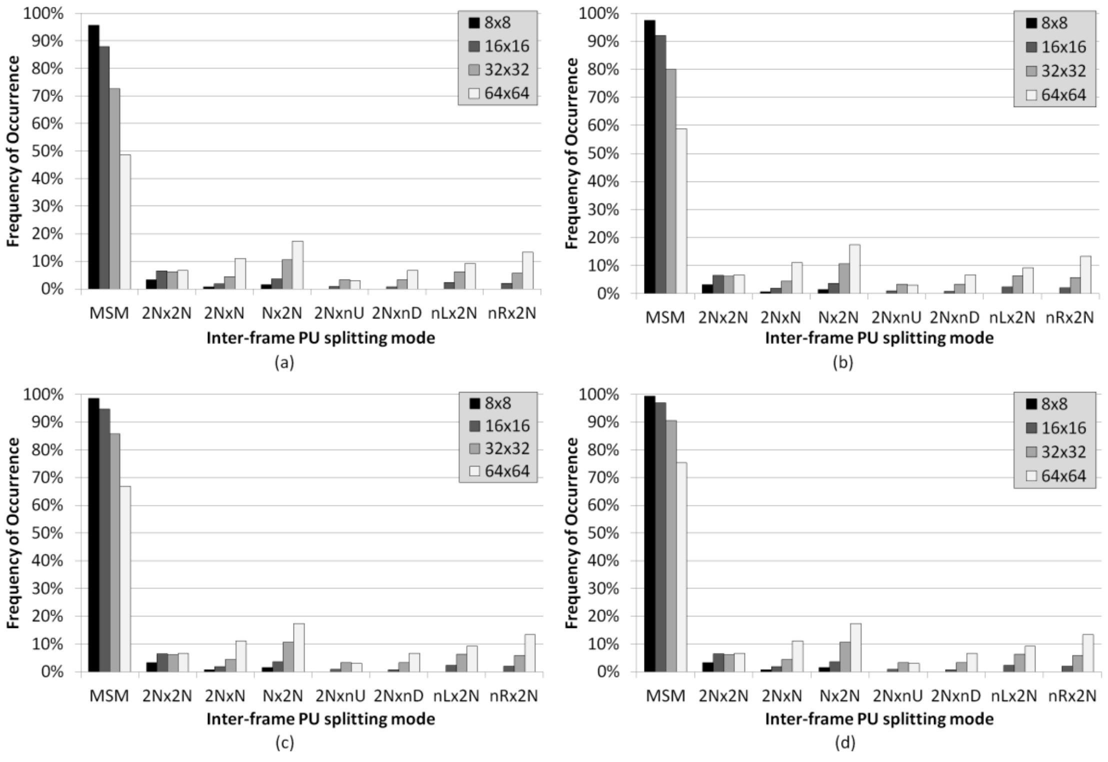
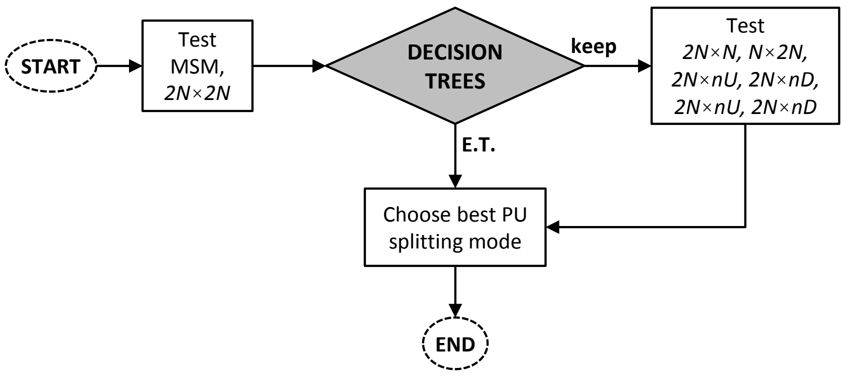


Fig. 6.8: Frequency of occurrence of each inter PU splitting mode in the *BasketballDrive* sequence encoded with (a) QP 27, (b) QP 32, (c) QP 37, and (d) QP 42.



START
Test MSM, 2N×2N
DECISION TREES
keep
Test 2N×N, N×2N, 2N×nU, 2N×nD, 2N×nU, 2N×nD
E.T.
Choose best PU splitting mode
END

Fig. 6.9: Inter-frame PU splitting mode decision with the proposed early termination.

In this early termination, the class attribute is the information of whether or not a CU should be split into PUs smaller than *2N×2N*. After an extensive observation on the collected data, the variables that provided the largest information gains were used for training the decision trees. These attributes are listed in Table 6.3, where information gain results are presented separately for each CU size.

Table 6.3: Information Gain Attribute Evaluation for the PU early termination.

| **Attribute** | **Information Gain** | | | |
|---|---|---|---|---|
| | **64×64** | **32×32** | **16×16** | **8×8** |
| *Ratio(2N×2N, MSM)* | 0.245 | 0.390 | 0.475 | 0.572 |
| *NormDiffRD(2N×2N, MSM)* | 0.129 | 0.245 | 0.341 | 0.433 |
| *RD(MSM)* | 0.224 | 0.306 | 0.383 | 0.422 |
| *RD(2N×2N)* | 0.165 | 0.139 | 0.101 | 0.135 |
| *RD(best)* | 0.208 | 0.284 | 0.364 | 0.407 |
| *UpperCU_div* | – | 0.223 | 0.297 | 0.240 |
| *Ratio(best, MSM)* | 0.195 | 0.266 | 0.229 | 0.208 |
| *NormDiffRD(best, MSM)* | 0.146 | 0.207 | 0.203 | 0.186 |

The attributes *Ratio(2N×2N, MSM)*, *NormDiffRD(2N×2N, MSM)*, *RD(MSM)*, and *RD(2N×2N)* have the same meaning as in section 6.3. Attribute *RD(best)* is the lowest R-D cost among the MSM and the *2N×2N* modes and *UpperCU_div* is the information of whether or not the CU in the upper Coding Tree depth was split into PUs smaller than *2N×2N*. Finally, the attributes *Ratio(best, MSM)* and *NormDiffRD(best, MSM)* correspond to the ratio and normalised difference (computed as in (Eq. 53) and (Eq. 54), respectively), between *RD(best)* and *RD(2N×2N)*.

$$Ratio(best, MSM) = \frac{RD(best)}{RD(MSM)} \quad \text{(Eq. 53)}$$

$$NormDiffRD(best, MSM) = \left| \frac{RD(best) - RD(MSM)}{RD(MSM)} \right| \quad \text{(Eq. 54)}$$

The attribute that yields the highest information gain, on average, is the *Ratio(2N×2N, MSM).* An explanation similar to the one exposed in the previous section applies to justify the use of such attribute: when a compression gain is noticed due to performing ME/MC for a CU instead of encoding it with MSM, there is a chance of the block belonging to an image region with some motion activity or texture heterogeneity. In such cases, it is advisable to test smaller PU sizes to verify if they yield additional compression gains. Fig. 6.10 presents statistical results in the form of distribution of the values of this attribute considering the four CU sizes. The statistics correspond to values obtained from all video sequences and QPs mentioned in section 6.2. Fig. 6.10 shows that the ratio between the *2N×2N* and MSM R-D costs is a relevant indicator of the necessity of testing the remaining modes. It is possible to see that most CUs with a small ratio were split into PUs smaller than *2N×2N*, while CUs with larger ratio values are mostly encoded with *2N×2N* or MSM.

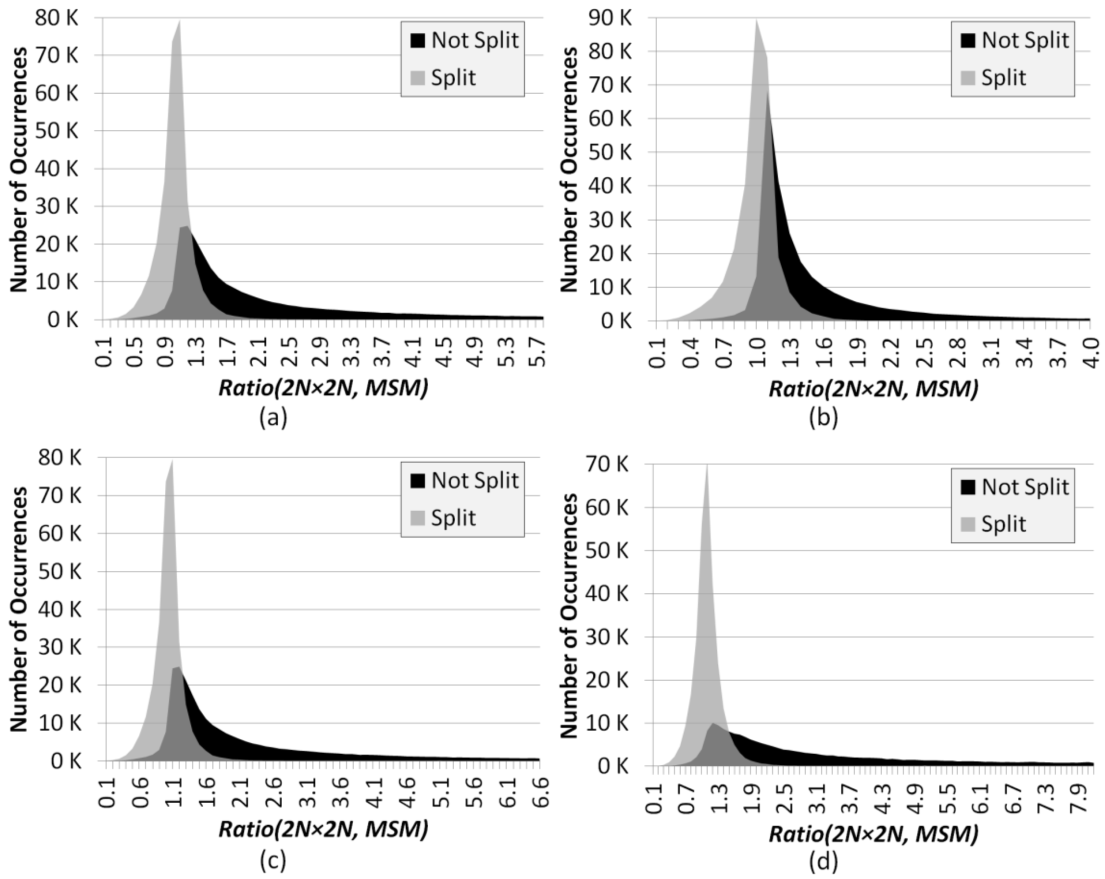


Fig. 6.10: Occurrence of (a) 64×64 CUs, (b) 32×32 CUs, (c) 16×16 CUs, and (d) 8×8 CUs split and not split into PUs smaller than *2N×2N* according to the ratio between the R-D costs of *2N×2N* and MSM modes.

The relevance of the *UpperCU_div* attribute is illustrated in Fig. 6.11, which shows three charts for the attribute considering 32×32, 16×16, and 8×8 CUs. As 64×64 CUs do not have a parent CU, since they are the largest CUs allowed in HEVC, the *UpperCU_div* attribute is not used in their corresponding decision trees. The charts show that there is a strong correlation between the PU splitting in the upper and current CU depths. For example, Fig. 6.11(b) shows that in 82.22% of the cases when a 16×16 CU was split into smaller PUs, its upper 32×32 CU was also split. Analogously, in 79.64% of the cases when a 16×16 CU was not split into smaller PUs, its corresponding parent 32×32 CU was also not split. The charts also show that the correlation increases for the *Split* case in small CUs (8×8) and increases for the *Not Split* case in large CUs (32×32).

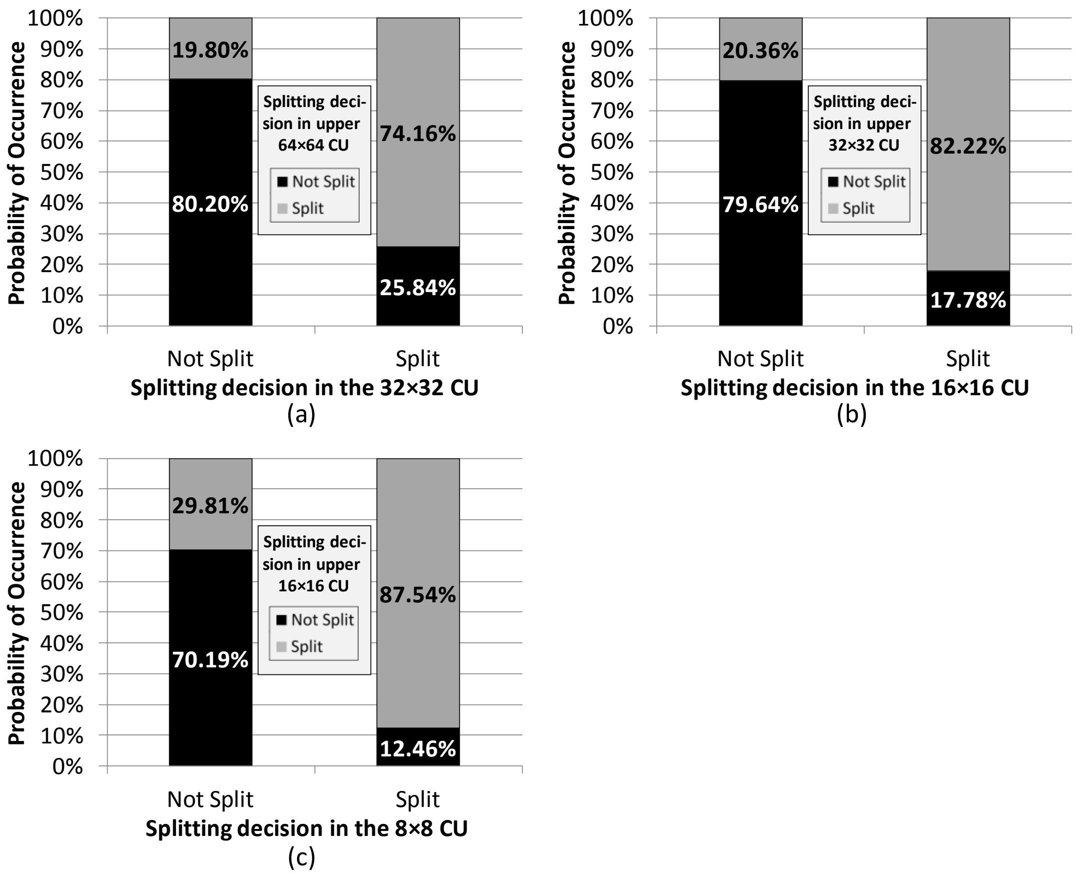


Fig. 6.11: Occurrence of (a) 32×32 CUs, (b) 16×16 CUs, and (c) 8×8 CUs split and not split into PUs smaller than *2N×2N* according to the splitting decision at the upper Coding Tree depth.

The decision trees were trained with the attributes shown in Table 6.3 and their characteristics are detailed in Table 6.4, which shows that their decision accuracy varies between 79.6% and 91.2%. Accuracy values are measured by dividing the number of correct splitting decisions by the total number of CUs analysed. However, regarding inaccurate decisions, it is important to notice that R-D efficiency losses would occur only when a CU should be predicted with PUs smaller than *2N×2N* but the decision process is early terminated, leading a non-optimal PU splitting mode to be chosen. In the remaining inaccurate decisions (i.e., when the CU should be predicted with MSM or *2N×2N* and the decision process is not early terminated), the encoder still chooses an optimal mode through RDO, since all modes are tested. The sixth column of Table 6.4 shows statistics only for the case that incurs in R-D losses, which varies between 2.1% and 8.6%.

Table 6.4: Characteristics and performance of trees trained for the Prediction Unit early termination.

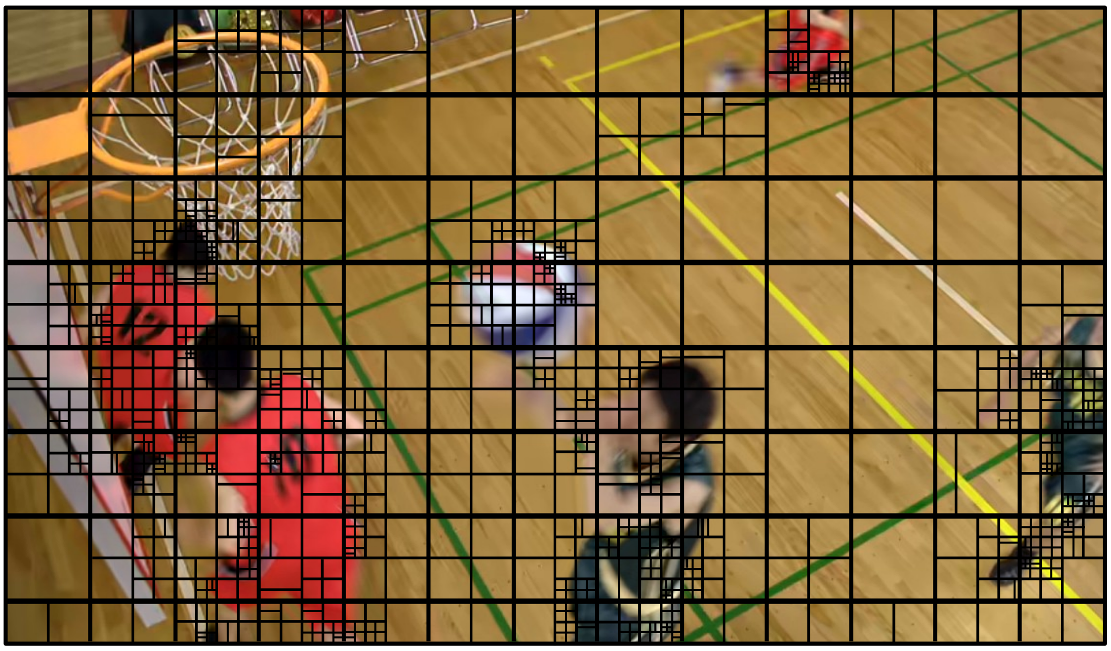
| CU size | Depth | Test Nodes | Leaves | Decision Accuracy | Inaccurate Depth |
|---|---|---|---|---|---|
| 64×64 | 5 | 8 | 9 | 79.6% | 8.6% |
| 32×32 | 7 | 9 | 10 | 86.0% | 5.0% |
| 16×16 | 5 | 9 | 10 | 89.2% | 4.0% |
| 8×8 | 4 | 5 | 6 | 91.2% | 2.1% |

Table 6.4 also shows that all the trained trees are very short, with a maximum depth of 7 in the case of 32×32 CUs. This means that the complexity added to the encoder due to implementing such trees is negligible, especially when considering that all attributes are already calculated by the HM encoder during its normal operation, except for the R-D cost ratios and normalised differences, which calculations represent an insignificant computational overhead in comparison to the whole encoding process. Detailed descriptions of the four decision trees are presented in Appendix C.

To illustrate the effectiveness of the proposed method, Fig. 6.12 presents the 100th frame of the *BasketballDrill* video sequence, which was not used in the training of the decision trees, and its corresponding PU boundaries for each CU. The sequence was encoded with QP 32 and the *Random Access* temporal configuration. The frame shown in Fig. 6.12(a) was encoded by the original HM encoder, while the frame in Fig. 6.12(b) was encoded by the HM encoder with the PU early termination implemented. It is possible to perceive that there are few differences between the boundaries in Fig. 6.12(a) and Fig. 6.12(b), which confirms that the early termination is capable of performing a correct PU determination in most cases.

## 6.5 Early Termination for Determining the RQT

As previously shown in chapter 4, even though restricting the maximum TU size does not provide substantial computational complexity reductions, it only affects the encoding R-D efficiency marginally. If these restrictions are carefully performed by a trained early termination scheme, R-D efficiency losses would most probably be insignificant and some complexity reduction could still be achieved. For this reason, the last early termination presented in this chapter focuses on halting the process for determining the best RQT structure, as described in this section.


(a)

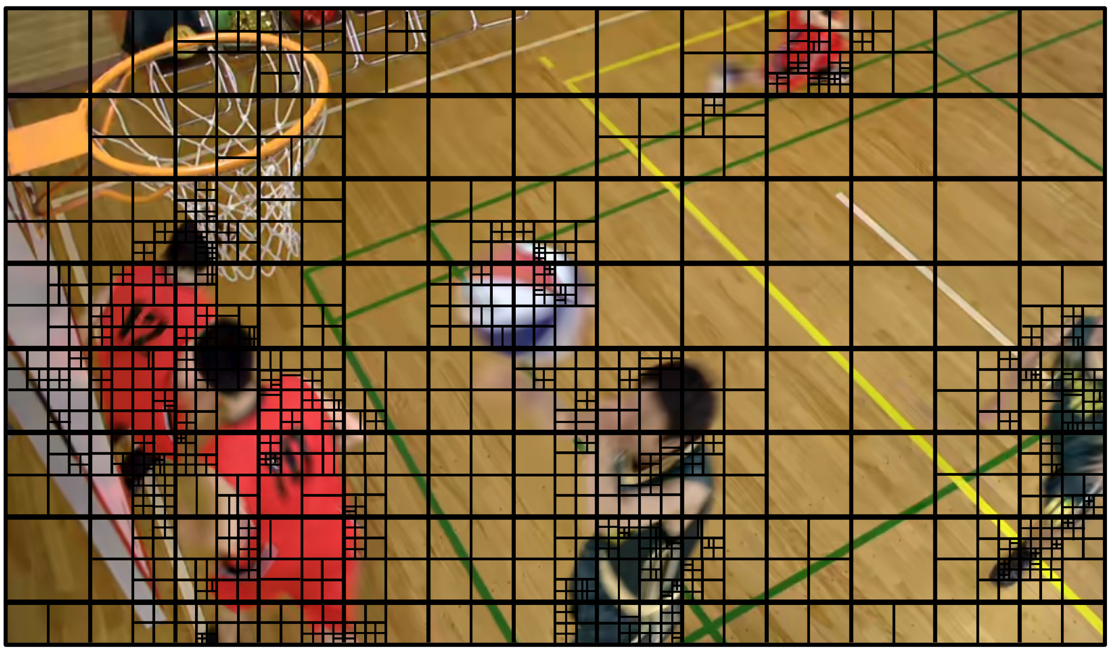

(b)

Fig. 6.12: PU boundaries in the 100th frame of the *BasketballDrill* video sequence encoded with QP 32 by (a) the original HM encoder and the (b) HM encoder with the PU early termination.

Similarly to the Coding Tree early termination scheme (section 6.3), the RQT early termination consists in deciding whether or not the splitting of TUs into four smaller TUs should be tried. If the tests return *yes*, the current TU is divided into four smaller TUs and the next RQT depth is tested. Otherwise, the process is finished for the current TU and its sub-TUs are not considered in the RDO-based decision.

As in the cases of the previously presented early terminations, the ARFF files used in the training process are 50% composed of data from TUs that have been split into four smaller TUs and 50% from TUs that have not been split. Given that TUs can assume three different dimensions in the *Main* encoder configuration used throughout this work (32×32, 16×16 and 8×8), two of which can be split into smaller TUs (32×32 and 16×16), only two different decision trees were designed for this early termination scheme.

In each training data set, the class attribute is the information of whether or not a TU should be split into four TUs. The remaining variables obtained from HM were analysed with IGAE and those selected as attributes to be used in the training of the decision trees are presented in Table 6.5, with their respective results in terms of information gain.

In Table 6.5, the attributes *AbsSumY*, *AbsSumU*, and *AbsSumV* are the sum of absolute values from the luminance, blue chrominance and red chrominance residues, respectively, and the *nonZeroCoeffY*, *nonZeroCoeffU*, and *nonZeroCoeffV* attributes represent the number of non-zero coefficients obtained after transforming the TU as a whole block (i.e., before splitting it). Finally, the *SingleCost* attribute is the R-D cost of encoding a TU as a whole block instead of splitting it.

Table 6.5: Information Gain Attribute Evaluation for the RQT early termination.

| **Attribute** | **Information Gain** | |
|---|---|---|
| | **32×32** | **16×16** |
| *AbsSumY* | 0.342 | 0.279 |
| *AbsSumU* | 0.057 | 0.033 |
| *AbsSumV* | 0.055 | 0.031 |
| *SingleCost* | 0.145 | 0.140 |
| *nonZeroCoeffY* | 0.348 | 0.284 |
| *nonZeroCoeffU* | 0.342 | 0.280 |
| *nonZeroCoeffV* | 0.342 | 0.280 |

As the residue samples are the inputs to the transform modules, we have investigated their probability of being encoded with small-sized TUs according to the intensity of its values. The values of *AbsSumY*, *AbsSumU* and *AbsSumV* attributes were analysed in order to check if the number of split TUs tend to increase with the sum of the residue samples. Fig. 6.13 shows that the high information gain associated to the *AbsSumY* attribute is due to the high correlation between its value and the splitting decision when the absolute sum of residues is null. Fig. 6.13(a) shows that 82.24% of the 32×32 TUs with *AbsSumY* value equals to zero were not split into smaller TUs. The charts also show that this correlation decreases significantly when *AbsSumY* is larger than zero. Notice, however, that in typical video sequences the value of *AbsSumY* will rarely be exactly equal to zero due to the presence of noisy source signals that generate residue information to be coded. Nevertheless, as the *C4.5* algorithm tries to find a threshold for the attribute that minimises the overall classification error of the tree, a value different from zero can be selected for the attribute if such choice yields a smaller error. Fig. C.8 and Fig. C.9 of Appendix C show that the thresholds chosen for the *AbsSumY* attribute in the decision trees corresponding to 16×16 and 32×32 CUs are 81 and 41, respectively, which are small numbers in comparison to their maximum values present in the training set (1149 and 3334 for 16×16 and 32×32 TUs, respectively).

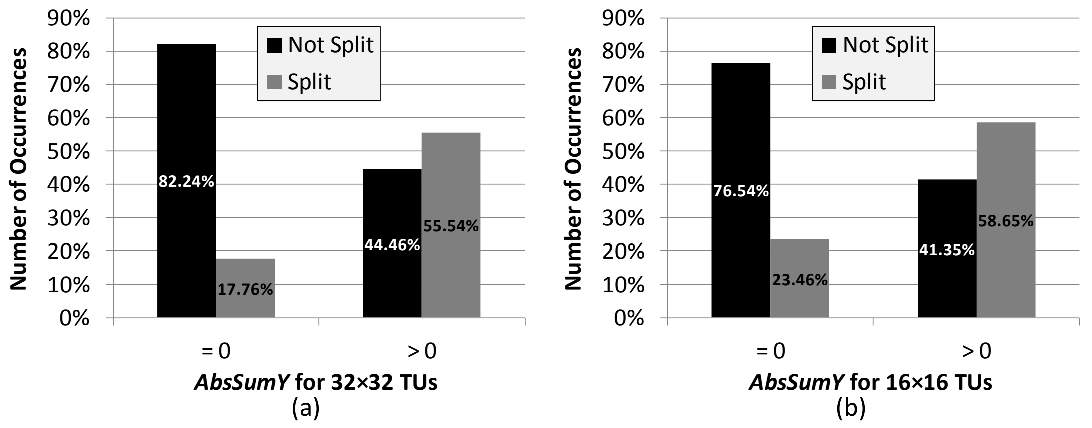


Fig. 6.13: Occurrence of (a) 32×32 and (b) 16×16 TUs split and not split into smaller TUs according to the absolute sum of luminance residues.

Differently from *AbsSumY*, which presents one of the largest information gain levels among all attributes, *AbsSumU* and *AbsSumV* add little information to the decision trees. However, as using them does not incur in any extra computational complexity, since these are variables already computed and available during the encoding process, they are considered in the training of the decision trees.

Statistics for the *nonZeroCoeffY* attribute are shown in Fig. 6.14. The *nonZeroCoeffU* and *nonZeroCoeffV* attributes presented very similar statistics. Notice that the statistics for the cases when a TU has only zero-valued coefficients (i.e., *nonZeroCoeffY* = 0) are identical to the statistics presented in Fig. 6.13 for the case when the sum of absolute luminance residues is equal to zero (i.e., *AbsSumY* = 0). This happens because when the transform receives only zero values as input, it yields only zero-valued coefficients. Although *nonZeroCoeffY* is redundant with the *AbsSumY* attribute in this case, it still yields relevant information when its value is larger than zero. For example, Fig. 6.14(a) shows that when *nonZeroCoeffY* is larger than zero, 82.58% of the 32×32 TUs are split into four 16×16 TUs and this information cannot be obtained with the *AbsSumY* attribute.

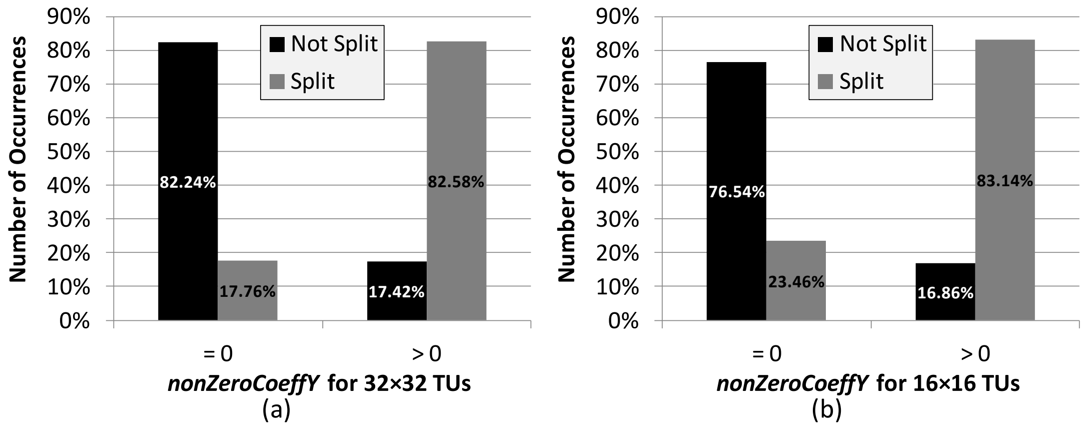


Fig. 6.14: Occurrence of (a) 32×32 and (b) 16×16 TUs split and not split into smaller TUs according to the number of non-zero luminance coefficients after the transform.

The *SingleCost* attribute is the R-D cost of encoding a TU as a whole block, i.e., without splitting it into four smaller TUs. Statistics for this attribute are presented in Fig. 6.15(a) for the specific case of 32×32 TUs and in Fig. 6.15(b) for 16×16 TUs. The figures show that there is a correlation between the TU splitting decision and the *SingleCost* value, since most of the non-split cases present very small R-D costs. Oppositely, for larger *SingleCost* values the number of non-split TUs decreases and the split cases become the majority, although by a small amount.

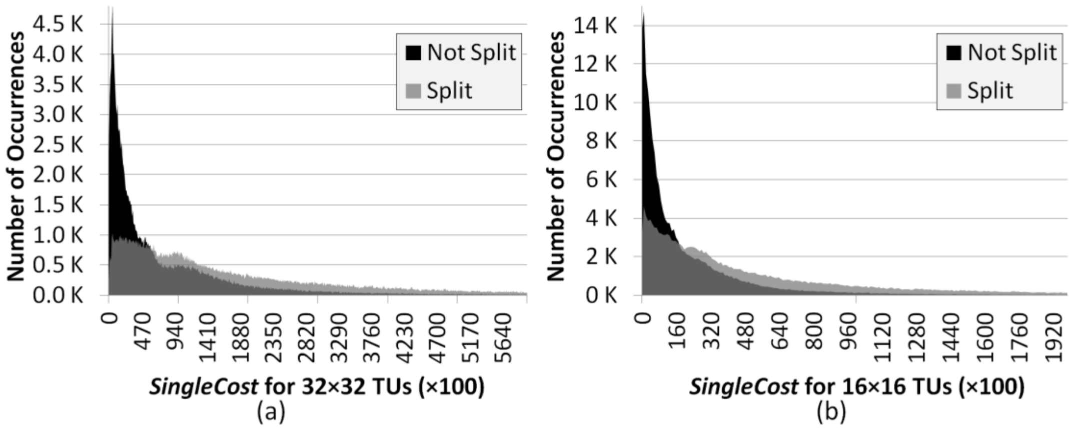


Fig. 6.15: Occurrence of (a) 32×32 and (b) 16×16 TUs split and not split into smaller TUs according to the R-D cost for the current TU depth.

The decision trees for 32×32 and 16×16 TUs are detailed in Table 6.6, which shows that their decision accuracy are 83.4% and 80.7%, respectively. As explained before in relation to the previous schemes, these decision accuracy results consider that splitting a TU when there is no need to do so is an error. However, the R-D efficiency is not harmed in such cases, since the best splitting option will be chosen through the exhaustive RDO evaluation procedure. Incorrect TU depth decisions occur only in the cases when the TU should be split into smaller TUs but the process is early terminated after the whole TU is evaluated. The sixth column of Table 6.6 shows statistics regarding the incorrect depth decisions caused by incorrect early termination, which varies between 8.9% and 12.1%. The two decision trees are detailed in the Appendix C of this thesis.

Fig. 6.16 shows the 100th frame of the *BasketballDrill* video sequence, which was not used in the training of the decision trees, and its corresponding TUs for each CU. Large, medium and small circles represent 32×32, 16×16 and 8×8 TUs, respectively. Grey, blue and red circles represent the luminance, blue chrominance and red chrominance TUs, respectively. In areas where the TUs are not shown, no prediction residue was generated to be encoded. The sequence was encoded with QP 32 and the *Random Access* temporal configuration. The frame shown in Fig. 6.16(a) was encoded by the original HM encoder, while the frame in Fig. 6.16(b) was encoded by the HM encoder with the RQT early termination implemented. It is possible to perceive that there are few differences between the circles shown in Fig. 6.16(a) and Fig. 6.16(b), which confirms that the early termination is capable of performing a correct RQT determination in most cases.

Table 6.6: Characteristics of trees trained for the RQT early termination.

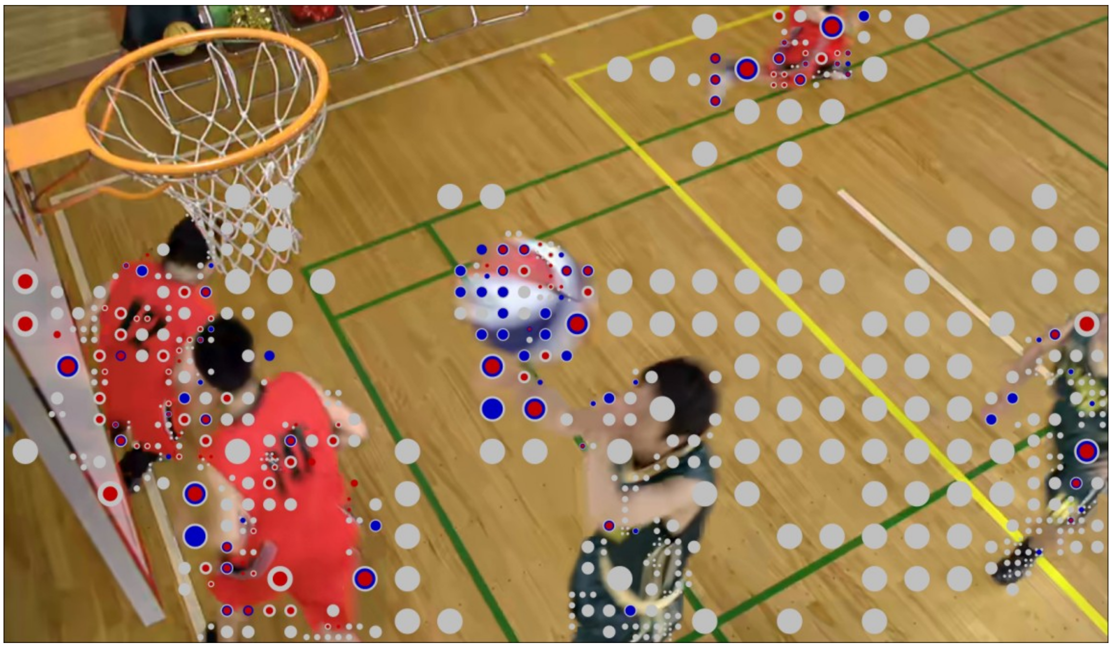
| CU size | Depth | Test Nodes | Leaves | Decision Accuracy | Incorrect Depth |
|---|---|---|---|---|---|
| 32×32 | 10 | 15 | 16 | 83.4% | 8.9% |
| 16×16 | 8 | 17 | 18 | 80.7% | 12.1% |

## 6.6 Experimental Results

### 6.6.1 Experimental Setup and Methodology

In order to evaluate the performance of the three early termination methods proposed in this chapter, the respective decision trees were implemented in the HM encoder so as to build the seven schemes listed in section 6.2. The HM encoder – version 12 (HM12) – was compiled with the *Microsoft Visual Studio C++ Compiler* and run on a clustered computer based on *Intel® Xeon® E5520* (2.27 GHz) processors running the *Windows Server 2008 HPC* operating system. Ten video sequences (*BasketballPass*, *BQSquare*, *BasketballDrill*, *ChinaSpeed*, *Kimono1*, *SlideEditing*, *BQTerrace*, *Cactus*, *PeopleOnStreet*, and *SteamLocomotive*), which are described in detail in Appendix B, were used to validate the early termination schemes and measure their R-D-C efficiency. As shown in Appendix B, these sequences differ from one another in terms of frame rate, bit depth, and spatial resolution. It is important to highlight that, in order to properly validate the early terminations, none of these test sequences was used in the training of the decision trees.

Compression efficiency was measured in terms of BD-rate and BD-PSNR and the encoding times were obtained with the *Intel VTune Amplifier XE* software profiler [124]. BD-rate, BD-PSNR and encoding time variations were computed using the corresponding values obtained with the unmodified HM encoder as reference. All video sequences were encoded with QPs 22, 27, 32, and 37, using the *Random Access* temporal configuration.


(a)

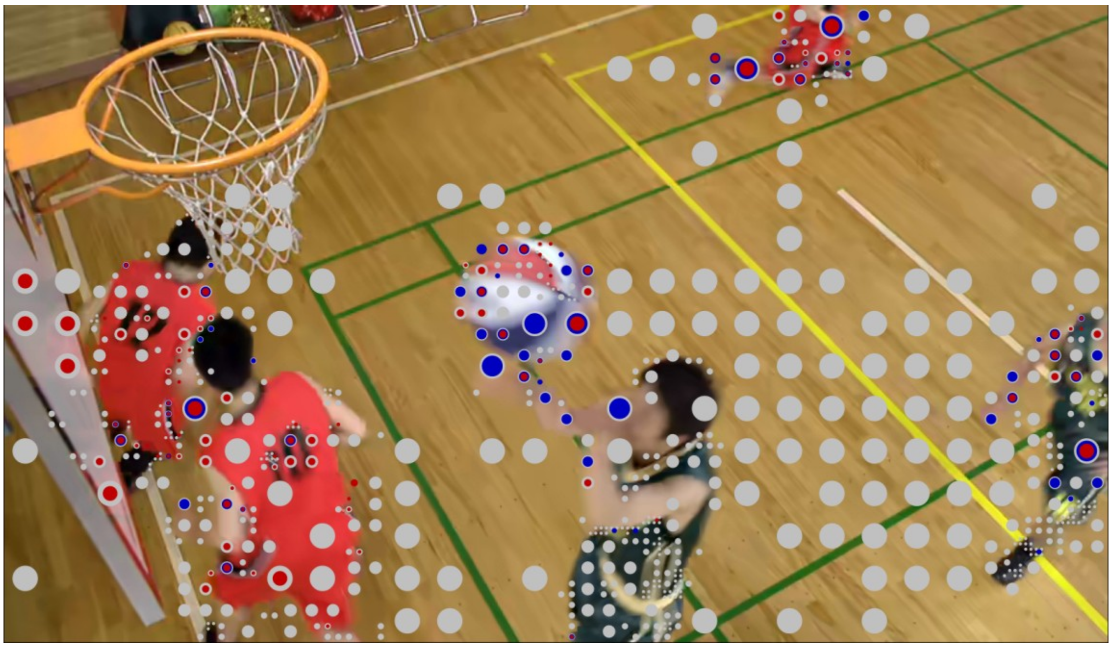

(b)

Fig. 6.16: TUs in the 100th frame of the *BasketballDrill* video sequence encoded with QP 32 by (a) the original HM encoder and the (b) HM encoder with the RQT early termination.

It should be pointed out that the HM encoder is a software implementation of HEVC developed during the standardisation process for tests and documentation purposes, so that it is not optimised for real-time operation. Still, as the definition of the partitioning structures is a high-level decision which does not directly affect the operation performed at any particular HEVC encoder module (though it influences the number of encoding iterations performed in the RDO process), similar complexity reduction factors are expected to be achieved in other encoder implementations more efficient than HM. It is also important to notice that the current version of the HM encoder already includes several complexity reduction techniques, such as RMD for intra-frame prediction, the CBF-based early termination, an early CU termination algorithm, and the early *SKIP* mode decision algorithm. The schemes presented in this chapter were implemented on top of all these methods, providing additional computational complexity reductions, and they were compared to the original HM encoder with all its built-in early terminations enabled.

The early terminations were first separately and then jointly implemented and evaluated. The next sub-sections present R-D-C results for the seven implementations listed in section 6.2.

### 6.6.2 Rate-Distortion-Complexity Results for Single Schemes

Table 6.7-Table 6.9 present R-D-C results for the three proposed schemes implemented separately. The results show that the PU early termination (Table 6.8) yields the largest reductions in computational complexity (*CCR* column), decreasing the total encoding time in a range from 37.4% to 68.1% (on average, 49.6%). These large reductions are explained by the fact that the scheme is applied to inter-predicted CUs and as inter-frame prediction is the most time-consuming task in the HEVC encoder, the complexity reductions achieved with early-terminated inter-predicted CUs have an important impact in the overall encoding complexity. The Coding Tree early termination scheme (Table 6.7) reduces the computational complexity in a range from 16.1% to 71.3% (on average, 36.7%), while the RQT early termination (Table 6.9) yields a computational complexity reduction varying from 5.6% to 8.9% (on average, 7.2%).

It is possible to notice from the results shown in Table 6.7 that the CCR values provided by the Coding Tree early termination are correlated with the texture characteristics of the videos. Sequences with large homogeneous areas (*SlideEditing* and *SteamLocomotiveTrain*, for example) are those which present the largest complexity reductions, because such areas are usually encoded with large CU sizes, allowing the early termination to halt the decision process in the first Coding Tree depths without significant loss of encoding performance and saving a significant amount of computation. High-resolution videos usually have larger numbers of homogeneous areas, but this is not always the case. For example, *BQTerrace* and *PeopleOnStreet* are examples of high-resolution videos that do not present very large complexity reductions because they are composed of detailed texture. Videos with small spatial resolution (e.g., *BQSquare*, *BasketballDrill*, *BasketballPass*) also present smaller complexity reductions, since they are mostly encoded with small-sized CUs.

The largest complexity reductions achieved with the PU early termination (Table 6.8) are also observed in those videos with large homogeneous areas, since in these cases the decision process is halted more frequently after testing the largest PU size (*2N×2N*). However, differently from the Coding Tree early termination, the complexity reductions achieved with the PU early termination also depend on the motion characteristics of the video. For example, the *BQTerrace* video sequence presents a larger complexity reduction with the PU early termination than with the Coding Tree early termination because it shows a scene in slow motion (mostly camera movement), which means that testing MSM or large PUs in reference frames is usually enough to find a good prediction. On the other hand, fast-motion scenes with non-continuous movement (e.g., *BasketballDrill*, *BasketballPass*) are those that present the smallest complexity reductions, since more modes need to be tested.

Notice that in some video sequences a small BD-rate decrease was noticed instead of the expected increase. This is the case of the *BQSquare* and *BQTerrace* sequences in Table 6.7, and the *BQTerrace* and *SlideEditing* sequences in Table 6.9. A careful analysis of the encoding results for these sequences showed that in some areas of some frames the early terminations lead to the use of larger partitioning structures than those chosen by the original HM encoder. This is the case of the 11th frame in the *SlideEditing* sequence. The analysis also showed that the motion fields for this frame and following ones, obtained when using the early termination procedures, were more coherent than those obtained when using the original encoder. Even though these early decisions do not yield the best R-D efficiency possible for the current CTU, it appears that the bits saved through the increased use of large partitions, especially with MSM and *SKIP* modes, result in lower BD-rates. As these particular video sequences present very homogeneous and moderate-motion characteristics, small or no image quality decreases are noticed due to these decisions, which, associated to the bit rate decrease, leads to the negative BD-rate values shown in the tables.

Table 6.7: R-D-C results for the Coding Tree early termination.

| **Video Sequence** | **BD-rate (%)** | **BD-PSNR (dB)** | **CCR (%)** | **Ratio BD-rate/CCR** |
|---|---|---|---|---|
| *BQSquare* | -0.007 | 0.000 | 23.0 | -0.030 |
| *BQTerrace* | -0.008 | 0.000 | 28.4 | -0.027 |
| *BasketballDrill* | +0.425 | -0.017 | 30.1 | 1.414 |
| *BasketballPass* | +0.128 | -0.006 | 23.9 | 0.534 |
| *Cactus* | +0.207 | -0.007 | 41.0 | 0.506 |
| *ChinaSpeed* | +0.131 | -0.007 | 29.2 | 0.449 |
| *Kimono1* | +0.552 | -0.020 | 44.0 | 1.253 |
| *PeopleOnStreet* | +0.089 | -0.004 | 16.1 | 0.556 |
| *SlideEditing* | +0.353 | -0.057 | 71.3 | 0.495 |
| *SteamLocomotiveTrain* | +0.971 | -0.022 | 60.3 | 1.608 |
| **Average** | **+0.284** | **-0.014** | **36.7** | **0.774** |

Table 6.8: R-D-C results for the Prediction Unit early termination.

| **Video Sequence** | **BD-rate (%)** | **BD-PSNR (dB)** | **CCR (%)** | **Ratio BD-rate/CCR** |
|---|---|---|---|---|
| *BQSquare* | +0.299 | -0.014 | 45.7 | 0.655 |
| *BQTerrace* | +0.091 | -0.002 | 54.4 | 0.168 |
| *BasketballDrill* | +0.491 | -0.020 | 44.6 | 1.101 |
| *BasketballPass* | +0.449 | -0.020 | 42.5 | 1.055 |
| *Cactus* | +0.401 | -0.012 | 48.6 | 0.827 |
| *ChinaSpeed* | +1.001 | -0.051 | 47.1 | 2.127 |
| *Kimono1* | +0.689 | -0.024 | 50.9 | 1.353 |
| *PeopleOnStreet* | +1.021 | -0.048 | 37.4 | 2.733 |
| *SlideEditing* | +0.206 | -0.030 | 68.1 | 0.302 |
| *SteamLocomotiveTrain* | +0.969 | -0.022 | 57.2 | 1.694 |
| **Average** | **+0.562** | **-0.024** | **49.6** | **1.132** |

Table 6.9: R-D-C results for the RQT early termination.

| Video Sequence | BD-rate (%) | BD-PSNR (dB) | CCR (%) | Ratio BD-rate/CCR |
|---|---|---|---|---|
| *BQSquare* | +0.073 | -0.003 | 8.9 | 0.820 |
| *BQTerrace* | -0.086 | +0.001 | 7.9 | -1.093 |
| *BasketballDrill* | +0.049 | -0.002 | 6.5 | 0.753 |
| *BasketballPass* | +0.033 | -0.002 | 6.8 | 0.482 |
| *Cactus* | +0.042 | -0.001 | 7.2 | 0.575 |
| *ChinaSpeed* | +0.131 | -0.006 | 6.7 | 1.959 |
| *Kimono1* | +0.169 | -0.006 | 5.6 | 2.988 |
| *PeopleOnStreet* | +0.249 | -0.012 | 6.2 | 4.022 |
| *SlideEditing* | -0.244 | +0.036 | 8.9 | -2.739 |
| *SteamLocomotiveTrain* | +0.132 | -0.004 | 7.3 | 1.806 |
| **Average** | **+0.055** | **0.000** | **7.2** | **0.758** |

To illustrate this explanation, the chart in Fig. 6.17 shows the number of bits per frame obtained when encoding the *SlideEditing* video (QP 32) with the original and the modified (*RQT ET*) HM encoders. The data shows that no differences are noticed in the number of bits between the two encoder versions, except for frame 11, where the modified encoder shows a much smaller number of bits than the original one. When analysing the whole video sequence, it was noticed that frame 11 is the one with the largest motion activity. A detailed analysis of this frame was then performed in order to understand the effect and its causes.

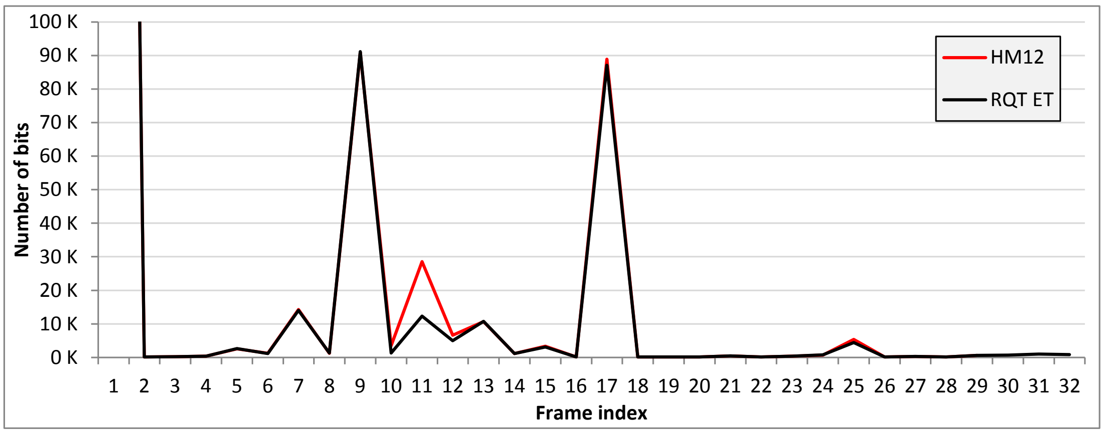


Fig. 6.17: Number of bits per frame in the *SlideEditing* sequence (QP 32) encoded with the original HM and the modified HM with the RQT early termination.

Fig. 6.18 shows the fragment of the 11th frame in which most of the motion activity occurs. In Fig. 6.18(a), the PU borders and MVs are shown for the fragment encoded with the original HM, while in Fig. 6.18(b) such information is omitted. Similarly, Fig. 6.18(c) and Fig. 6.18(d) show the same area encoded the modified HM. Blue and violet lines represent MVs referring to frames in *List 0* and *List 1*, respectively. By comparing Fig. 6.18(a) and Fig. 6.18(c), it is possible to notice that the number of MVs and PUs in the fragment encoded with the original HM is much larger than in the modified version. In the specific case of this video, and especially in this region where most of the motion activity occurs, the early termination led to the choice of larger partitioning structures and MVs than in the original encoder, which finally resulted in a much smaller bit rate. As this is a very homogeneous video (computer-generated imagery), such larger partitioning structures and MVs led to the choice of more coherent motion fields than in the original encoder, which can be noticed in Fig. 6.18(b) and Fig. 6.18(d) by the quality of the encoded images. The circled area in Fig. 6.18(b), for example, shows that the original encoder predicted the image area corresponding to the word "minor" by dividing it into several small PUs (see Fig. 6.18(a)), which were predicted from different areas of reference frames, resulting in something that reads as "mind". Oppositely, the same circled area in Fig. 6.18(d), shows that the modified encoder predicted the area as a single large PU with a MV pointing to a vertically offset position in a previous frame (*List 0*), resulting in the correct word "minor".

When compared to the six alternative partitioning structure configurations tested in section 4.2, the three schemes proposed here achieve much better R-D-C efficiency. The alternative partitioning structure configuration of section 4.2 that provided a complexity reduction closer to an early termination scheme proposed in this chapter, namely the PU early termination, is *PAR 3*. While that configuration provides a complexity reduction of 53.5% in comparison to the original HM encoder (see Table 4.6), the PU early termination achieves a slightly smaller complexity reduction of 49.6% (see Table 6.8). However, the BD-rate increase observed with *PAR 3* is 18.1% (see Table 4.5), while the increase for the PU early termination is only 0.56% (see Table 6.8). Clearly, the R-D-C efficiency of the method proposed in this chapter is much better. In fact, while the BD-rate/CCR ratio for those configurations in section 4.2 varied between 3.2 and 44.2 (see Table 4.7), the three early terminations proposed in this chapter resulted in average ratios between 0.76 and 1.13, when implemented separately.

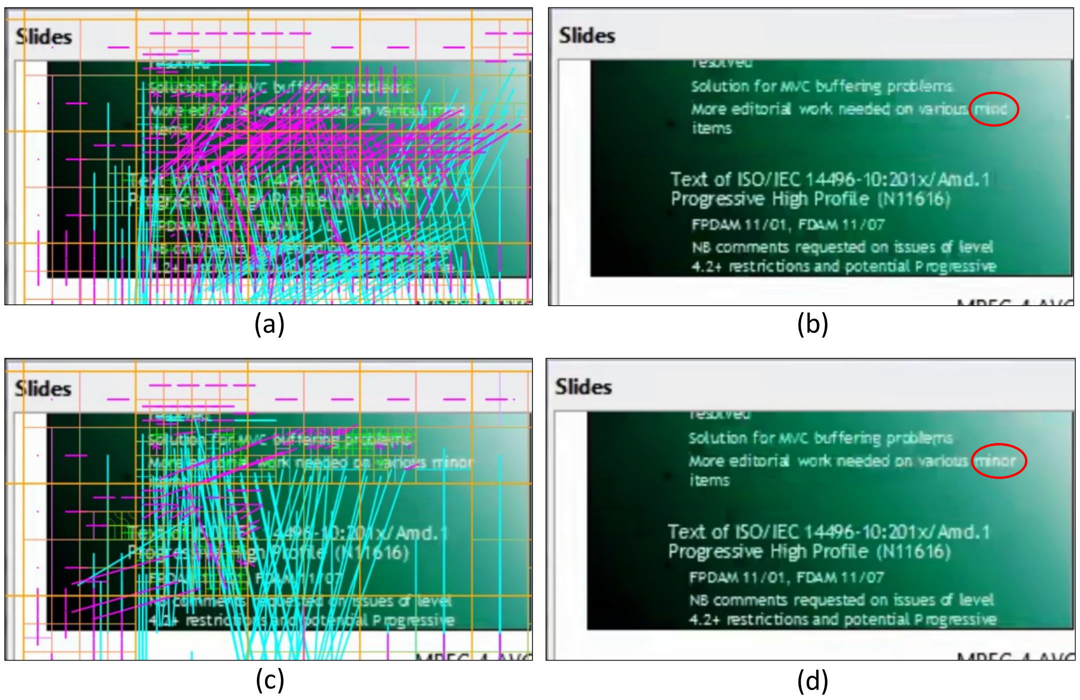


Fig. 6.18: Fragment of the 11th frame in the *SlideEditing* sequence (QP 32) encoded with (a, b) the original HM and (c, d) the modified HM with the RQT early termination.

Fig. 6.19 shows the R-D efficiency of the three proposed schemes for four video sequences encoded with four different QPs (22, 27, 32, and 37). In all charts of Fig. 6.19, the curves represent R-D results for the original encoder (*HM12*), the Coding Tree early termination (*CT ET*), the PU early termination (*PU ET*), and the RQT early termination (*RQT ET*) implementations, respectively. The charts of Fig. 6.19(a) and Fig. 6.19(b) show results for the *SlideEditing* and *BQTerrace* video sequences, which are those that presented the best R-D-C efficiency considering the three early terminations. The two remaining charts (Fig. 6.19(c) and Fig. 6.19(d)) are for the *Kimono1* and *PeopleOnStreet* sequences, which are those that presented the worst R-D-C efficiency results. It is perceptible that the R-D efficiency achieved with the three proposed early terminations is very close to that of the original encoder, since the curves overlap in all charts presented in Fig. 6.19. A closer detailed look (300% zoom boxes) shows that the curves overlap even in the worst-case video sequences.

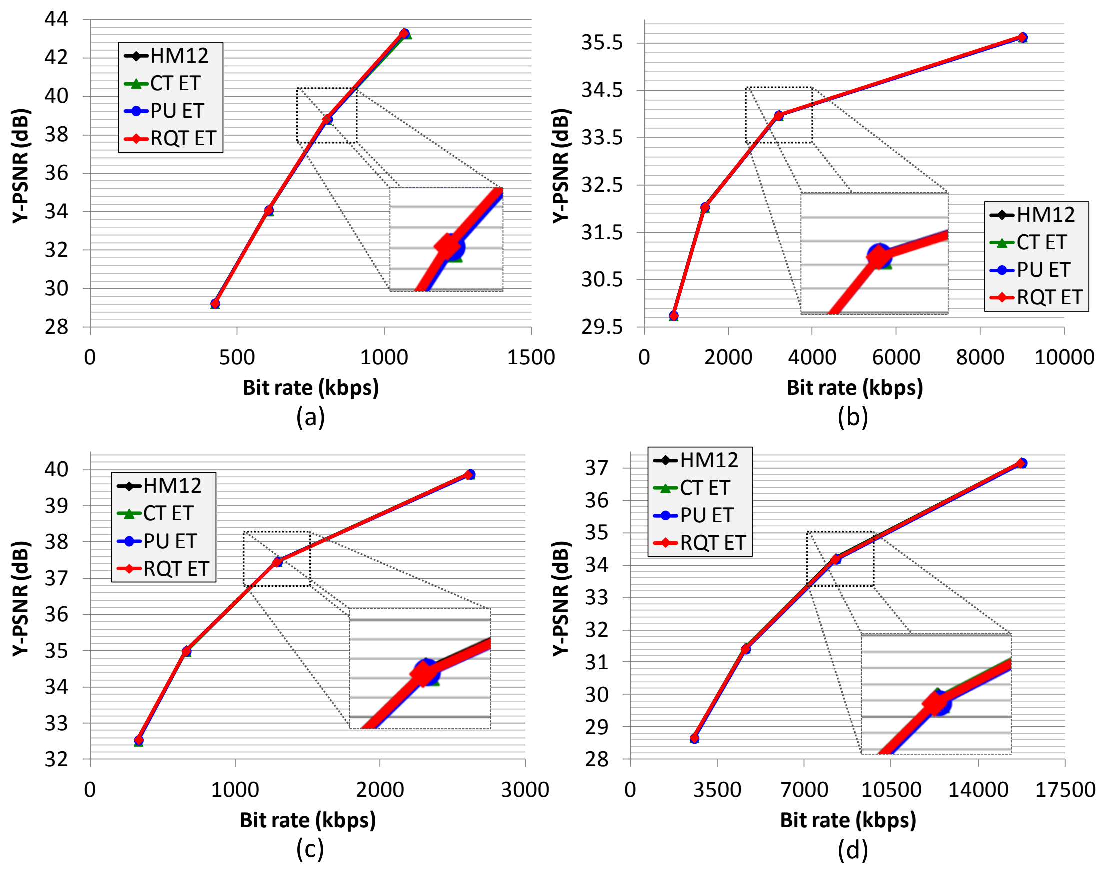


Fig. 6.19: Rate-Distortion efficiency for the (a) *SlideEditing*, (b) *BQTerrace*, (c) *Kimono1*, and (d) *PeopleOnStreet* video sequences encoded with the original HM and the three early terminations implemented separately.

### 6.6.3 Rate-Distortion-Complexity Results for Joint Schemes

The early terminations were jointly implemented in HM to investigate if by combining the different methods the individual complexity reductions added or if the use of some methods hindered the operation of the others, resulting in a total complexity reduction smaller than the sum of the partial reductions [20].

Table 6.10-Table 6.12 present results for three schemes that use two early terminations in conjunction: Coding Tree and PU early terminations, PU and RQT early terminations, and Coding Tree and RQT early terminations, respectively. Finally, results for an implementation that aggregates the three early terminations are presented in Table 6.13. As expected, the largest computational complexity reductions are achieved when all the early termination schemes are jointly implemented in HM. In this case, the computational complexity reductions vary from 46.4% up to 87.6% (65.3%, on average, as shown in Table 6.13). Notice that the reduction achieved is not equivalent to the sum of the reductions achieved with each scheme separately. This happens due to the nature of the partitioning structures used in HEVC. For example, when constraining the number of R-D evaluations performed to decide the best Coding Tree structure, the overall number of PU partitioning evaluations is also decreased, since PUs are defined within CUs. Similarly, the number of RQT evaluations also decreases because CUs are used as roots for RQTs and so if there are less CUs with residues to be coded, there will be less RQTs whose optimal partition structure needs to be found.

In terms of R-D efficiency, the scheme with all early terminations presented the largest losses in comparison to the remaining versions, as expected. However, the average BD-rate increase of 1.36% is still negligible in face of the considerable computational complexity reduction achieved, 65.3%. For this reason, this configuration would be the most advisable solution for an implementation that admits a small loss in R-D efficiency. When compared to the alternative partitioning structure configurations tested in section 4.2, the scheme presents much better R-D-C efficiency. *PAR 3* and *PAR 4* of section 4.2 are the configurations that provided a complexity reduction closer to the achieved with the scheme that implements all early terminations. The two configurations provide a complexity reduction of 53.5% and 74.1%, respectively (see Table 4.6), while the joint early terminations achieve a reduction of 65.3% (see Table 6.13). Nevertheless, the BD-rate increases of *PAR 3* and *PAR 4* are 18.1% and 36% (see Table 4.5), respectively, while the increase for the joint early terminations scheme is only 1.36% (see Table 6.13).

Fig. 6.20 shows R-D efficiency results for the four joint implementations described above. In all charts of Fig. 6.20, the curve labelled as *HM12* represents R-D results for the original encoder, while the curves labelled as *CT+PU ET*, *PU+RQT ET*, *CT+RQT ET*, and *CT+PU+RQT ET* represent R-D results for the joint Coding Tree and PU early terminations, the joint PU and RQT early terminations, the joint Coding Tree and RQT early terminations, and the joint implementation with the three early terminations, respectively. The charts in Fig. 6.20(a) and Fig. 6.20(b) show results for the *BQTerrace* and *BQsquare* sequences, respectively, which presented the best R-D-C results in the four joint schemes. Fig. 6.20(c) and Fig. 6.20(d) show results for the *Kimono1* and *PeopleOnStreet* sequences, respectively, which presented the worst R-D-C efficiency results. As in the previous comparisons for the schemes implemented separately, the R-D efficiency achieved in the joint cases is also very close to that of the original encoder. Even in the worst-case videos, a detailed look (300% zoom) shows that the curves are very close.

Table 6.10: R-D-C results for joint Coding Tree and PU early terminations.

| Video Sequence | BD-rate (%) | BD-PSNR (dB) | CCR (%) | Ratio BD-rate/CCR |
|---|---|---|---|---|
| *BQSquare* | +0.378 | -0.017 | 54.3 | 0.697 |
| *BQTerrace* | +0.283 | -0.006 | 72.1 | 0.393 |
| *BasketballDrill* | +1.295 | -0.052 | 55.7 | 2.324 |
| *BasketballPass* | +0.873 | -0.038 | 50.9 | 1.717 |
| *Cactus* | +0.999 | -0.031 | 64.0 | 1.561 |
| *ChinaSpeed* | +1.410 | -0.072 | 56.3 | 2.505 |
| *Kimono1* | +2.745 | -0.096 | 67.7 | 4.052 |
| *PeopleOnStreet* | +1.463 | -0.068 | 43.3 | 3.382 |
| *SlideEditing* | +1.293 | -0.190 | 86.6 | 1.493 |
| *SteamLocomotiveTrain* | +2.573 | -0.059 | 77.8 | 3.305 |
| **Average** | **+1.331** | **-0.063** | **62.9** | **2.118** |

Table 6.11: R-D-C results for joint PU and RQT early terminations.

| Video Sequence | BD-rate (%) | BD-PSNR (dB) | CCR (%) | Ratio BD-rate/CCR |
|---|---|---|---|---|
| *BQSquare* | +0.378 | -0.018 | 51.3 | 0.737 |
| *BQTerrace* | +0.188 | -0.005 | 58.9 | 0.320 |
| *BasketballDrill* | +0.446 | -0.018 | 48.2 | 0.925 |
| *BasketballPass* | +0.735 | -0.032 | 46.6 | 1.580 |
| *Cactus* | +0.442 | -0.013 | 52.7 | 0.839 |
| *ChinaSpeed* | +1.206 | -0.061 | 50.5 | 2.388 |
| *Kimono1* | +0.876 | -0.031 | 53.9 | 1.627 |
| *PeopleOnStreet* | +1.269 | -0.059 | 40.8 | 3.113 |
| *SlideEditing* | +0.075 | -0.011 | 72.1 | 0.104 |
| *SteamLocomotiveTrain* | +1.408 | -0.032 | 60.9 | 2.311 |
| **Average** | **+0.702** | **-0.028** | **53.6** | **1.311** |

Table 6.12: R-D-C results for joint Coding Tree and RQT early terminations.

| **Video Sequence** | **BD-rate (%)** | **BD-PSNR (dB)** | **CCR (%)** | **Ratio BD-rate/CCR** |
|---|---|---|---|---|
| *BQSquare* | +0.091 | -0.004 | 30.3 | 0.301 |
| *BQTerrace* | -0.028 | 0.000 | 56.4 | -0.050 |
| *BasketballDrill* | +0.453 | -0.018 | 33.6 | 1.349 |
| *BasketballPass* | +0.074 | -0.003 | 29.0 | 0.255 |
| *Cactus* | +0.284 | -0.009 | 44.5 | 0.637 |
| *ChinaSpeed* | +0.242 | -0.012 | 33.1 | 0.733 |
| *Kimono1* | +0.839 | -0.030 | 46.7 | 1.796 |
| *PeopleOnStreet* | +0.396 | -0.018 | 20.9 | 1.895 |
| *SlideEditing* | +0.544 | -0.072 | 73.2 | 0.744 |
| *SteamLocomotiveTrain* | +0.553 | -0.015 | 62.6 | 0.884 |
| **Average** | **+0.345** | **-0.018** | **43.0** | **0.802** |

Table 6.13: R-D-C results for joint Coding Tree, PU and RQT early terminations.

| **Video Sequence** | **BD-rate (%)** | **BD-PSNR (dB)** | **CCR (%)** | **Ratio BD-rate/CCR** |
|---|---|---|---|---|
| *BQSquare* | +0.406 | -0.019 | 59.0 | 0.689 |
| *BQTerrace* | +0.282 | -0.007 | 74.4 | 0.379 |
| *BasketballDrill* | +1.182 | -0.048 | 58.3 | 2.029 |
| *BasketballPass* | +0.958 | -0.042 | 54.1 | 1.773 |
| *Cactus* | +1.006 | -0.031 | 66.3 | 1.517 |
| *ChinaSpeed* | +1.547 | -0.078 | 58.8 | 2.629 |
| *Kimono1* | +3.013 | -0.105 | 69.2 | 4.355 |
| *PeopleOnStreet* | +1.740 | -0.081 | 46.4 | 3.752 |
| *SlideEditing* | +0.920 | -0.134 | 87.6 | 1.050 |
| *SteamLocomotiveTrain* | +2.493 | -0.058 | 79.1 | 3.154 |
| **Average** | **+1.355** | **-0.060** | **65.3** | **2.075** |

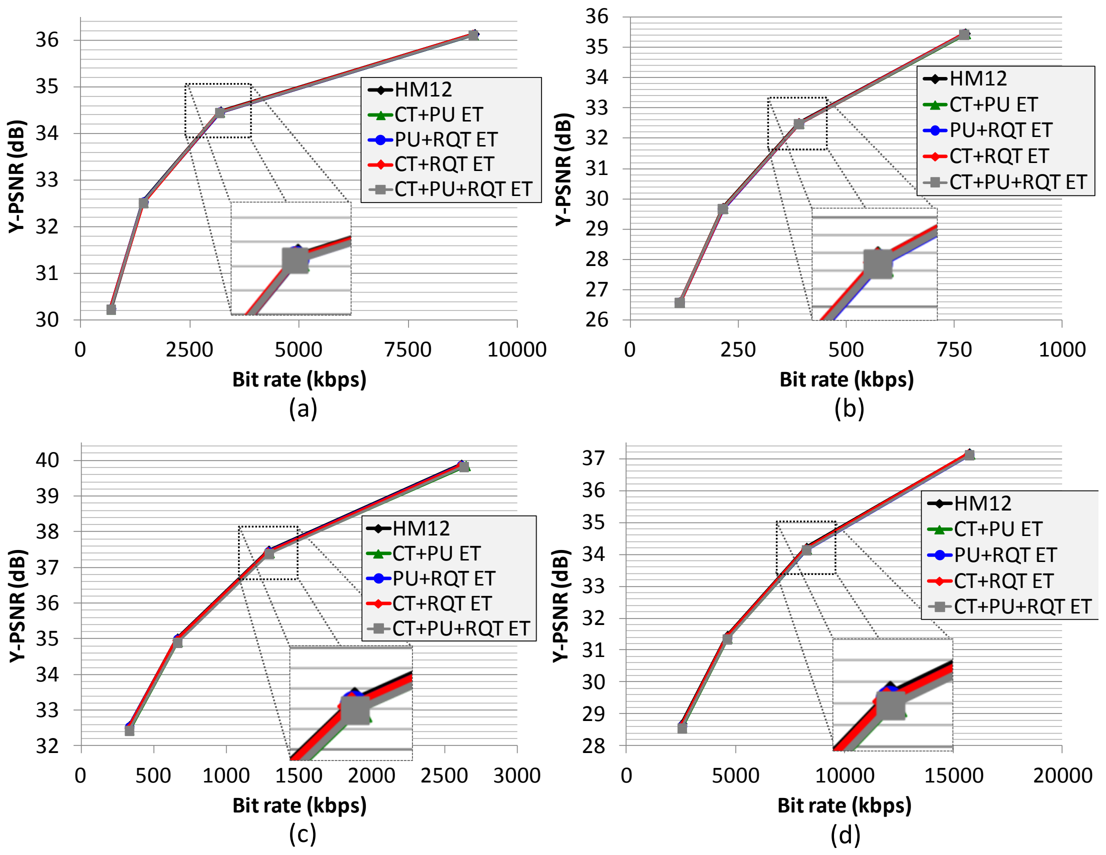


Fig. 6.20: Rate-Distortion efficiency for the (a) *BQTerrace*, (b) *BQSquare*, (c) *Kimono1*, and (d) *PeopleOnStreet* video sequences encoded with the original HM and the four joint early termination schemes implemented.

## 6.7 Results Discussion

To quantify the quality of the methods proposed in this section, we compared their performance with those of the best performing HEVC complexity reduction methods reviewed in chapter 3, which also operate through modifications in the partitioning structure decision process. Only those works that provide comparable results (i.e., BD-rate and complexity reduction values using the original HM encoder as reference) were considered in the analysis. All the compared works were also tested with the *Random Access* configuration, QPs 22, 27, 32, 37, and were tested for at least seven video sequences with at least four different spatial resolutions, except for [98], which was kept in the comparisons because it was the only comparable work in its category.

Table 6.14 presents the results of these experiments in terms of BD-rate, computational complexity reduction (*CCR* column) and the ratio between the two values for all compared works. Since each method under evaluation presents different values for BD-rate and complexity reduction, the ratio between these two quantities (BD-rate/CCR) was used to permit comparisons of the competing complexity reduction methods in terms of R-D efficiency loss per computational complexity saved. Using this performance indicator, a method which presents a given BD-rate and CCR is ranked higher than some other method with higher BD-rate but equal (or smaller) CCR (i.e., higher BD-rate/CCR ratio). The methods are grouped according to the partitioning structure that is constrained to achieve the desired complexity reduction. To ease the comparisons, the average results for the schemes proposed in this chapter are also reproduced at the end of each group. We can conclude from the data in Table 6.14 that all the schemes proposed in this chapter achieve better BD-rate/CCR ratios than the competing methods in their corresponding category. Even though an extensive research has been done in the available literature, no works have been found for comparisons in the categories *PU+RQT ET*, *CT+RQT ET* and *CT+PU+RQT ET*.

In the Coding Tree early termination category, the best related work [85] has a BD-rate/CCR ratio equals to 1.2, which is still larger than the ratio of the scheme proposed in section 6.3 (0.77). Besides, the method in [85] is only applicable to intra-predicted CUs, which means that its complexity reductions are only achieved in *All Intra* temporal configurations. In the PU early termination category, all related works present a ratio at least twice larger the obtained with the PU early termination scheme proposed in this chapter. Moreover, the PU early termination scheme presented in this thesis achieves a computational complexity reduction equal to or greater than that obtained in the related works. Finally, although the only comparable RQT early termination work achieves complexity reduction levels larger than those of the proposed RQT early termination, the cost of such reductions in terms of R-D efficiency loss is too large, producing a BD-rate/CCR ratio equals to 6.36, which is 8.4 times larger than the ratio of scheme proposed in section 6.5 (0.76).

The four joint early termination schemes proposed in this chapter are listed in the last lines of Table 6.14 and compared to joint Coding Tree and PU early termination methods found in the literature. We can observe that all joint schemes proposed in this chapter achieve better results in both terms of R-D efficiency and computational complexity reduction. The proposed joint scheme that integrates the three early terminations achieves complexity reductions that largely exceed those obtained in other related works, while still maintaining a low BD-rate increase and, consequently, BD-rate/CCR ratios smaller (i.e., better) than those achieved by the comparable joint implementations.

Table 6.14: Comparisons with related works.

| **Category** | **Reference** | **BD-rate (%)** | **CCR (%)** | **Ratio BD-rate/CCR** |
|---|---|---|---|---|
| *CT ET* | Seunghyun [85] | +0.6 | 50 | 1.20 |
| | J.-Hyeok [87] | +1.2 | 48 | 2.50 |
| | Goswami [88] | +1.7 | 38 | 4.42 |
| | Jian [89] | +1.9 | 43 | 4.42 |
| | **Proposed** | **+0.28** | **37** | **0.77** |
| *PU ET* | Vanne [93] | +1.3 | 51 | 2.55 |
| | Zhao [122] | +5.9 | 50 | 11.8 |
| | Khan [97] | +1.3 | 44 | 2.88 |
| | **Proposed** | **+0.56** | **50** | **1.13** |
| *RQT ET* | Yunyu [98] | +1.4 | 22 | 6.36 |
| | **Proposed** | **+0.05** | **7.2** | **0.76** |
| *CT+PU ET* | Wei-Jhe [100] | +5.1 | 43 | 5.11 |
| | Liquan [101] | +1.5 | 42 | 3.55 |
| | Xiaolin[102] | +1.9 | 41 | 4.54 |
| | Xiaolin [131] | +1.4 | 45 | 3.11 |
| | **Proposed** | **+1.33** | **63** | **2.12** |
| *PU+RQT ET* | **Proposed** | **+0.7** | **54** | **1.31** |
| *CT+RQT ET* | **Proposed** | **+0.34** | **43** | **0.80** |
| *CT+PU+RQT ET* | **Proposed** | **+1.36** | **65** | **2.07** |

## 6.8 Conclusions

This chapter presented a set of early termination schemes that reduce the computational complexity of the HEVC encoding process. All the schemes were developed making use of DM tools for the construction of decision trees that exploit intermediate encoding results to decrease the computational complexity involved in the decision of the best Coding Tree, PU and RQT structures. One single early termination scheme was separately developed and implemented in the HM software for each partitioning structure decision after performing an extensive analysis on the information gain yield by each possible attribute. The three implementations were then combined in pairs and finally all together in joint schemes, aiming at further reducing the HEVC computational complexity.

The effectiveness of the approach and the performance of the decision trees was validated through extensive experiments using a set of video sequences different from those used in the training phase. Experimental results have shown that an average complexity reduction of 65% can be achieved when the three early termination schemes are jointly implemented, with a compression efficiency loss of only 1.36% (BD-rate). Complexity reductions that go beyond those achieved in any other previously published comparable works were achieved through the proposed early termination methods, with smaller losses in terms of compression efficiency.

The proposed schemes do not require any computationally intensive operations to be added to the HEVC encoding process and the decision trees use only intermediate encoding results computed during normal HEVC encoding. Therefore, these early termination schemes can be seamlessly incorporated to any other HEVC encoder implementation with very small increase in the computational burden of the encoding. For reference, all trained decision trees were made available in Appendix C as supplemental material to this chapter.

# Chapter 7

# Complexity Scaling and Reduction Applied to Encoding Time Control

The main goal of this chapter is to combine the findings and methods proposed in chapters 4, 5 and 6, to build a control system that dynamically adjusts the encoder operation to keep the encoding time under a specific target. This system will be used as a showcase for the applicability of this thesis' major contributions described in the previous chapters.

Chapter 4 presented an R-D-C analysis that identified a set of tools and parameter settings that have a large influence in the overall encoding computational complexity and showed that large complexity reductions can be achieved with small compression efficiency loss by choosing judiciously those settings. Chapter 4 has also shown that the HEVC frame partitioning structures are responsible for a very large share of its encoding complexity. These observations motivated the introduction of a set of complexity scaling and reduction methods described in chapters 5 and 6 that operate by constraining the decision process followed to arrive at the optimal topology of these structures in R-D sense.

However, there is a remaining important problem of practical nature related to the subject of this dissertation: how can we combine the best low-complexity encoding tool configurations and the proposed complexity scaling and reduction algorithms to guarantee that encoding times are kept below a pre-defined target? This problem is addressed in this chapter, which presents an example of a system designed to control the encoding time per GOP based on the computational complexity reduction and scaling methods proposed in previous chapters.

The design of this encoder control system starts by the definition of a set of encoder operational configurations that combine different arrangements of encoding tools (based on the findings of chapter 4) and different schemes for early terminating the partitioning structure decision process (based on the decision trees of chapter 6). Then, each possible configuration is tested in an offline training process that is performed to derive the best configuration options in terms of R-D-C efficiency. These configurations are then sorted in descending order of encoding time. The encoding time control algorithm is then implemented using these configurations to adjust, at a medium granularity level, the time spent by the HEVC encoder to process each GOP. Once an operational configuration is chosen, a finer adjustment of the encoding time control is performed based on the CCUPU method proposed in chapter 5. This finer control is done by changing the number of constrained CTUs per frame, which allows achieving encoding times per GOP closer to the target time.

The high-level diagram of the encoding time control system proposed in this chapter is presented in Fig. 7.1. The inputs of the encoding time control are a target time per GOP, which might be specified from an external entity like an operation system process scheduler, and the encoding time of previously encoded GOPs (or a function of previous GOPs encoding times). The necessary complexity reduction factor is calculated according to the difference or ratio of these two input values. Based on this factor, an encoder operating point is chosen in such a way that the encoding time is adjusted towards the desired value.

Section 7.1 explains how the encoder operating points were obtained and section 7.2 presents the proposed encoding time controller. Experimental results that illustrate the control system operation and quantify its performance are presented and analysed in section 7.3.

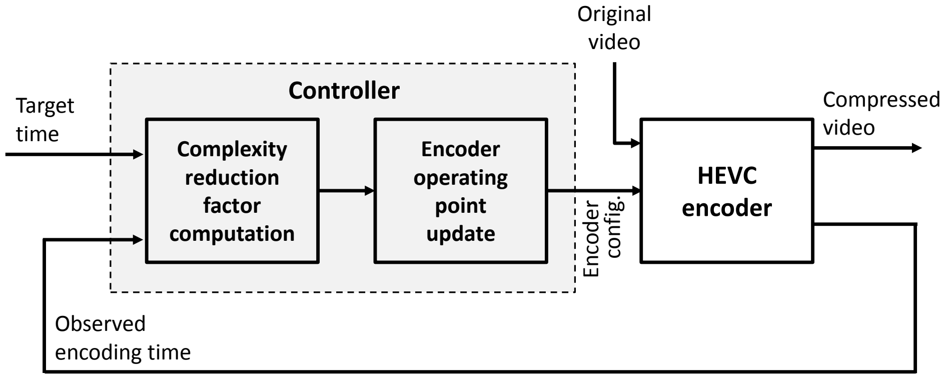


Fig. 7.1: Encoding time control system integrated with the HEVC encoder.

## 7.1 Finding the Encoder Operating Points

As previously explained, improved R-D efficiency was achieved through the addition of new tools and encoding features, so that the resulting encoder supports several configurations at the cost of different levels of computational complexity, as shown in chapter 4. If computational complexity is to be considered in the selection of encoding parameters, a new dimension must be added to the RDO problem, resulting in an R-D-C optimisation (RDCO). However, RDCO is a very difficult task to be performed in real-time encoding, mainly because, differently from the rate (R) and distortion (D) variables, computational complexity cannot be retrieved once spent, so that it is not possible to go back and try another parameterisation without incurring in more computational cost. On the other hand, not testing all possibilities can lead to a significant decrease in the compression efficiency if a careful selection of tested configurations is not performed.

Fig. 7.2 shows the R-D-C space obtained when encoding a certain video sequence using a set of 240 different configurations. Each point in the chart represents the R, D, and C values of a given configuration in terms of bit rate (kbps), distortion (MSE) and encoding time (seconds), respectively. If a point is not outscored by any other in the three terms (R, D, and C) it is said to be a non-dominated point. For example, take the point labelled as *P* in Fig. 7.2. There is no other point in the chart that possesses, simultaneously, a smaller distortion, a smaller bit rate, and a smaller encoding time than *P* and for this reason it is called a non-dominated point. All non-dominated points in Fig. 7.2 are shown as red points and they correspond to the so-called Pareto frontier[5], which means that they are those that present the best R-D-C efficiency among all tested cases and thus should be selected for use in an R-D-C optimised encoding time control system. The next sub-sections explain how these best-performing points were obtained and utilised in the control system proposed in this chapter to adjust the encoding time per GOP.

[5] The Pareto frontier is composed of every point that is Pareto-efficient (i.e., not dominated by any other alternative). A point *A* is said to dominate *B* if *A* outscores *B* regardless of the trade-off between *A* and *B*. In other words, if *A* is better and cheaper than *B*, *A* dominates *B*. In the specific case of Fig. 7.2, a configuration *A* dominates another configuration *B* if it results in a smaller distortion, in a smaller bit rate and in a smaller encoding time than *B*. All configurations that are not dominated by any other are thus called Pareto-efficient configurations.

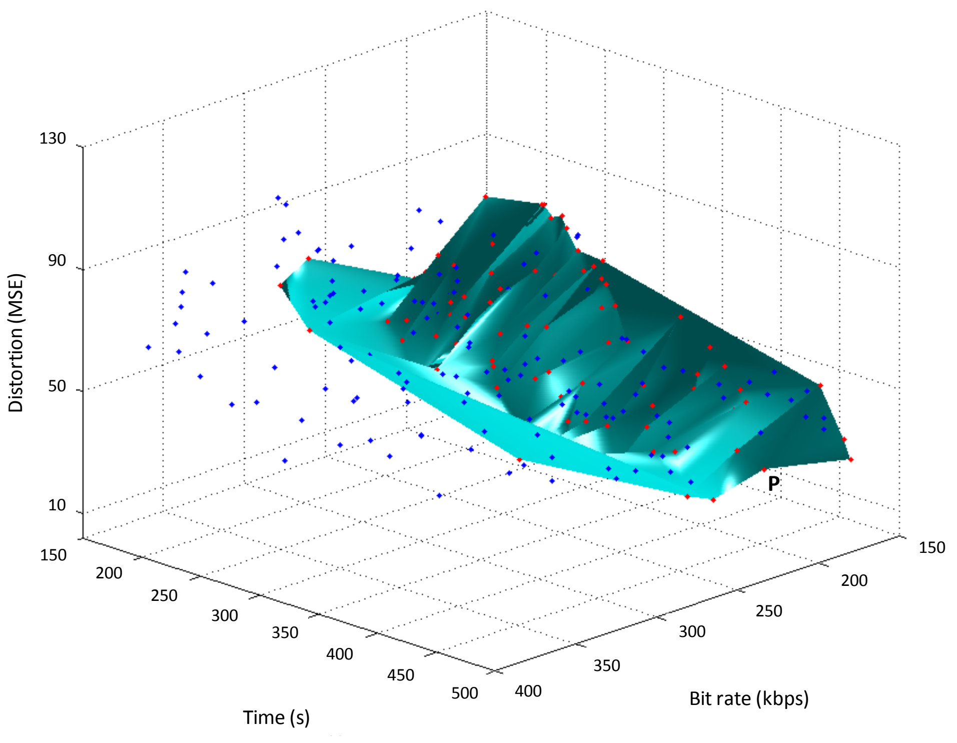


Fig. 7.2: Example of R-D-C space and 3-D Pareto frontier for a set of 240 encoding parameter configurations.

## 7.1.1 Parameters Selection for R-D-C Analysis

In order to implement the encoding time control system presented in this chapter, the encoding configurations that yield the best R-D efficiency for a given computational complexity were selected through an extensive set of offline evaluations. Even though the HM encoder presents a large number of configuration parameters, only those identified in chapter 4 as having the largest computational complexity impact were selected for the experimental study described in this section. Besides those parameters, the three single early termination schemes proposed in chapter 6 were also included in the R-D-C analysis, so that three parameters were added to the HM encoder representing the activation or deactivation state of each early termination scheme. The R-D-C analysis could include several other encoding parameters of HM, but, as shown later in this chapter, a limited subset of them had to be chosen with care to make this experimental analysis feasible in a reasonable time. Expanding the set of parameter to larger numbers would call for a very large number of experimental evaluations of encoder performance due to the rapid growth of the number of configuration possibilities with the number of parameters.

Section 4.1.2 examined 17 configurations and identified seven which, when used, resulted in an increase of the encoding computational complexity in a range from 5% to 53%, depending on the video sequence used (see Fig. 4.1). The remaining configurations did not alter the encoding computational complexity significantly. The seven most complex configurations in section 4.1.2 are identified as *TEST 2 – TEST 7* and *TEST 14* and they correspond, respectively, to modifications in the *Inter 4×4*, *Search Range*, *Bi-prediction Refinement*, *Hadamard ME*, *Fast Encoding*, *Fast Merge Decision*, and *AMP* encoding parameters. In section 4.1.4 the accumulated effect of enabling each tool was also analysed and it was concluded that the same seven parameters were responsible for the largest increases in computational complexity when compared to a baseline configuration (see Fig. 4.4).

Due to their large impact in computational complexity, these seven parameters are the best candidates for use in the construction (by combinations and variations of their values) of the configurations to be used in the R-D-C analysis presented in this section. However, only three of them could actually be analysed[6]: the *Search Range* (*SR*), the *Bi-prediction Refinement* (*BPR*) and the *Hadamard ME* (*HME*). They are respectively referred as *SR*, *BPR* and *HME* from now on in this chapter. Similarly, the three early termination schemes described in chapter 6, namely the *Coding Tree Early Termination* (*CTET*), the *Prediction Unit Early Termination* (*PUET*) and the *Residual Quadtree Early Termination* (*RQTET*), were used to define the configurations. In the experiments that will be described shortly, these six parameters can take values from the following sets:

- $SR \in \{64, 32, 16, 8, 4\}$;
- $BPR \in \{4, 2, 1\}$;
- $HME \in \{\text{on, off}\}$;
- $CTET \in \{\text{on, off}\}$;
- $PUET \in \{\text{on, off}\}$;
- $RQTET \in \{\text{on, off}\}$.

[6] The *Inter 4×4* parameter was not included in the R-D-C analysis because it is not supported in current HM versions. The configurations modifying the *Fast Encoding* and the *Fast Merge Decision* parameters in chapter 4 actually disabled them (thus increasing the computational complexity and decreasing R-D efficiency), so that they are always enabled in the CTC-based configurations used in the R-D-C analysis presented in this chapter. The *AMP* parameter controls the use of *Asymmetric Motion Partitions*, which is already controlled by the *Prediction Unit Early Termination* (*PUET*).

Table 7.1 shows part of the encoding configurations that were created by modifying the value of each parameter, one at a time. Every parameter value was tested with all possible combinations of values of the remaining parameters, totalising 240 encoding configurations, all of which are detailed in Appendix D.

Table 7.1: Encoding configurations tested in the R-D-C analysis.

| **Config.** | ***SR*** | ***BPR*** | ***HME*** | ***CTET*** | ***PUET*** | ***RQTET*** |
|---|---|---|---|---|---|---|
| 1 | 64 | 4 | on | off | off | off |
| 2 | 32 | 4 | on | off | off | off |
| 3 | 16 | 4 | on | off | off | off |
| 4 | 8 | 4 | on | off | off | off |
| ... | ... | ... | ... | ... | ... | ... |
| 31 | 64 | 4 | on | on | off | off |
| ... | ... | ... | ... | ... | ... | ... |
| 240 | 4 | 1 | off | on | on | on |

### 7.1.2 Operating Points at the Pareto Frontier

Each one of the 240 configurations mentioned in the previous section was used to encode four high-resolution test video sequences (*BQTerrace*, *Cactus*, *PeopleOnStreet*, *SteamLocomotiveTrain*), with QPs 22, 27, 32, 37, and the *Random Access* temporal configuration, totalising 3,840 encodings. Bit rate, PSNR and encoding time corresponding to each encoding run and each video sequence were saved and used to calculate the average BD-rate, BD-PSNR and computational complexity for each configuration, using the base configuration 1 of Table 7.1 as the reference. Detailed results for each configuration are presented in Appendix D.

In a real-case encoder implementation, either constant bit rate (with variable video quality) or constant video quality (with variable bit rate) are used, so that not all points in the R-D-C space must be considered simultaneously in the analysis to find the best-performing configurations. The R-D-C analysis was thus simplified by eliminating either the rate or the distortion term by comparing BD results in two separate analyses. The first analysis considers a constant bit rate scenario, thus providing a Distortion-Complexity (D-C) space, while the second analysis considers variable bit rate with constant quality, which characterises a Rate-Complexity (R-C) space.

Fig. 7.3 shows a chart in which the 240 configurations are plotted as blue points in a D-C space. As configuration 1 was used as reference to compute both the BD-PSNR and the normalised computational complexity for each case, it appears in the rightmost corner of the plot, with minimum encoding performance reduction and maximum computational complexity. In other words, configuration 1 yields the best image quality and the largest computational complexity among all cases. Configuration 240 is in the other extreme of the chart, with the largest loss of encoding performance (-0.15 dB) and the smallest normalised computational complexity (0.37). The points connected by the dashed line in Fig. 7.3 represent the configurations in the upper convex hull of the D-C space, which are those with the best trade-off between quality and complexity. Red-circled configurations (1, 31, 32, 97, 212, 222, and 237) are those selected as best options in a possible D-C optimised system. This is because although configurations 238, 239 and 240 are in the upper convex hull, they incur in a very large increase in distortion, so that there is no advantage in including them in a D-C optimised scheme.

Fig. 7.4 presents a chart similar to the one in Fig. 7.3, but shows BD-rate results and the lower convex hull of the R-C space. Similarly, configuration 1 is the one that yields that best compression rates and the largest computational complexity. On the other hand, configuration 240 presents the largest BD-rate increase (4.6%) and the smallest normalised computational complexity (0.37). Notice that most of the points (i.e., encoding configurations) that compose the lower convex hull in Fig. 7.4 also belong to the upper convex hull in Fig. 7.3. The only exception is configuration 217, which appears only in the lower convex hull of Fig. 7.4. The coincidence of points happens because each BD measure takes into consideration the variation of both bit rate and image quality in its calculation (i.e., a variation in bit rate will affect both BD-PSNR and BD-rate values), which reveals that the convex hull configurations in both charts yield the best R-D-C results among all cases.

Notice that configuration 31 incurs in almost no efficiency loss in both Fig. 7.3 and Fig. 7.4. In a complexity-constrained encoder, there would be no reason to select configuration 1 instead of 31, since the latter results in virtually the same R-D efficiency at a much smaller computational complexity. Configuration 31 is an alternative, less complex configuration that does not introduce R-D efficiency loss with respect to configuration 1. Table 7.1 shows that configurations 1 and 31 only differ in the activation of *CTET*.

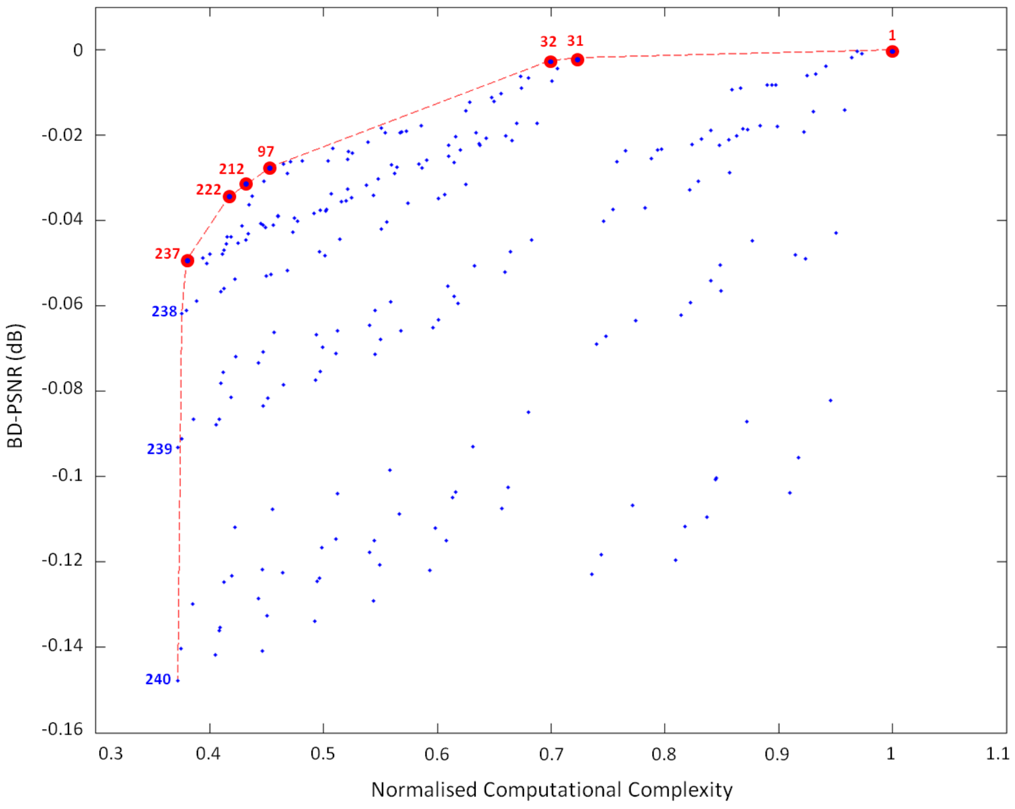


Fig. 7.3: Upper convex hull of the D-C space for the 240 encoder configurations tested.

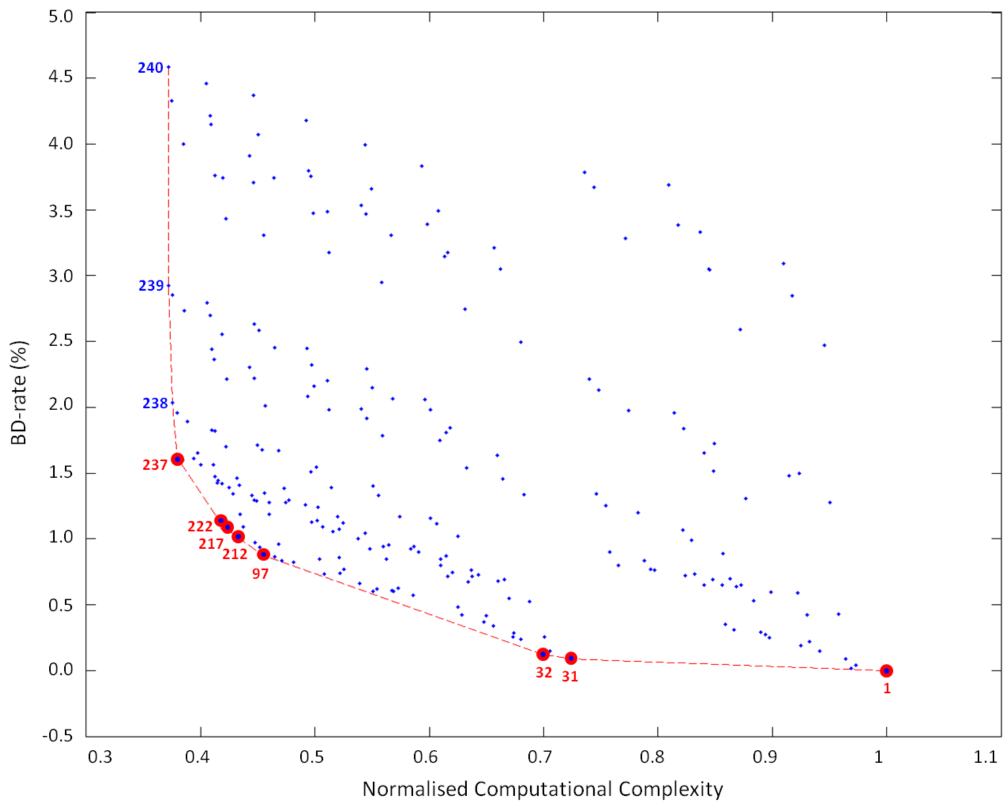


Fig. 7.4: Lower convex hull of the R-C space for the 240 encoder configurations tested.

It is possible to perceive that as we traverse the frontier line increasing the configuration number in the convex hulls, the normalised computational complexity decreases and the encoding performance diminishes. This can be used to allow complexity scaling for encoding time control in a system that admits some R-D efficiency loss by adaptively selecting an encoder configuration among those belonging to the convex hulls of Fig. 7.3 and Fig. 7.4 (i.e., configurations 31, 32, 97, 212, 217, 222, and 237). However, as one can notice in both charts, there are not too many points sufficiently close to each other to permit controlling the encoding time with at least medium granularity. For example, the average computational complexity difference between configurations 32 and 97 is 25%, which is a rather large value.

This problem was solved by selecting the configurations belonging to the Pareto frontier, which was previously defined. In the specific case of Fig. 7.3, a configuration $A$ dominates another configuration $B$ if it results in a smaller BD-PSNR and in a smaller computational complexity than $B$. In the case of Fig. 7.4, a configuration $A$ dominates a configuration $B$ if it yields a smaller BD-rate and a smaller computational complexity than $B$. By selecting the points that belong to the Pareto frontier (i.e., non-dominated points), we can increase the amount of configurations selected for complexity scaling through the inclusion of points that are close to the convex hull frontier line (dashed lines in Fig. 7.3 and Fig. 7.4).

Fig. 7.5 and Fig. 7.6 show the Pareto frontier corresponding to the D-C and the R-C charts, respectively. All points that belong to the frontier are circled in red. Except for the configuration that incurs in virtually no R-D efficiency loss (configuration 31), the remaining selected configurations are close enough to be used in a medium-granularity encoding time control system. Configurations 238, 233, 234, and 239 would not be useful, since they present almost the same computational complexity as configuration 232 and result in a much larger R-D efficiency loss. For this reason, they were not included in the set of encoder configurations selected for use in the encoding time control system presented in the next section.

Table 7.2 shows the 27 selected configurations that belong to the Pareto frontier of the plots shown in Fig. 7.5 and Fig. 7.6. The table shows the value for each parameter and the resulting normalised computational complexity, BD-PSNR and BD-rate for the configurations, which are listed in descending order of computational complexity and

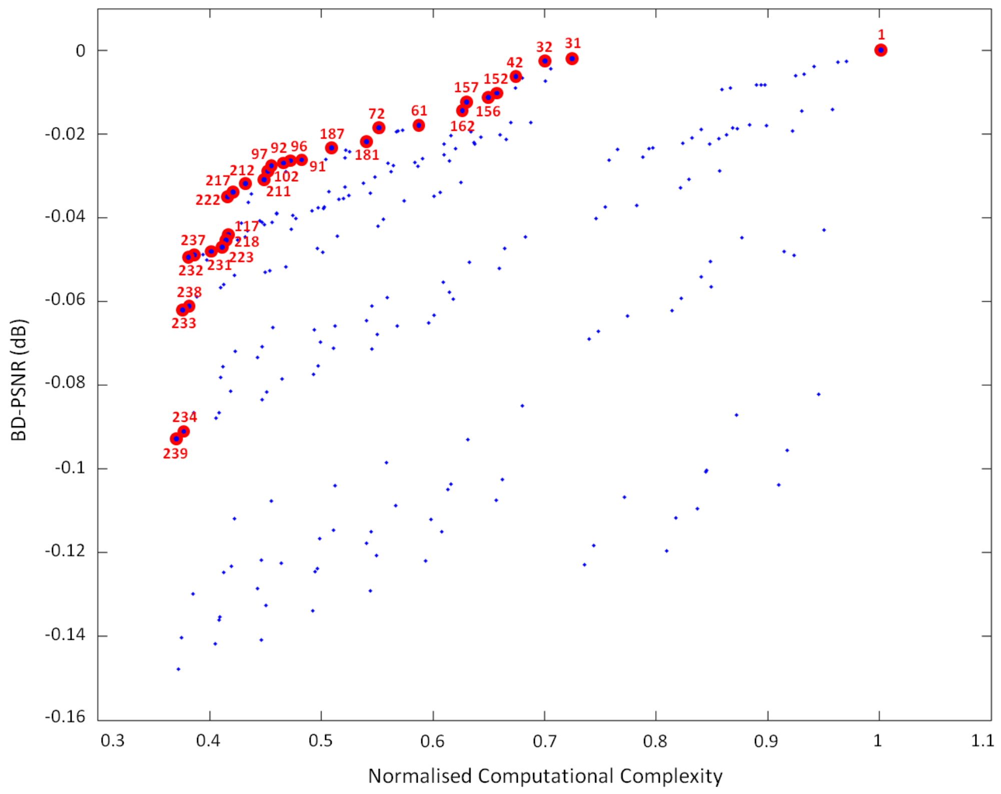


Fig. 7.5: Pareto frontier of the D-C space for the 240 encoder configurations tested.

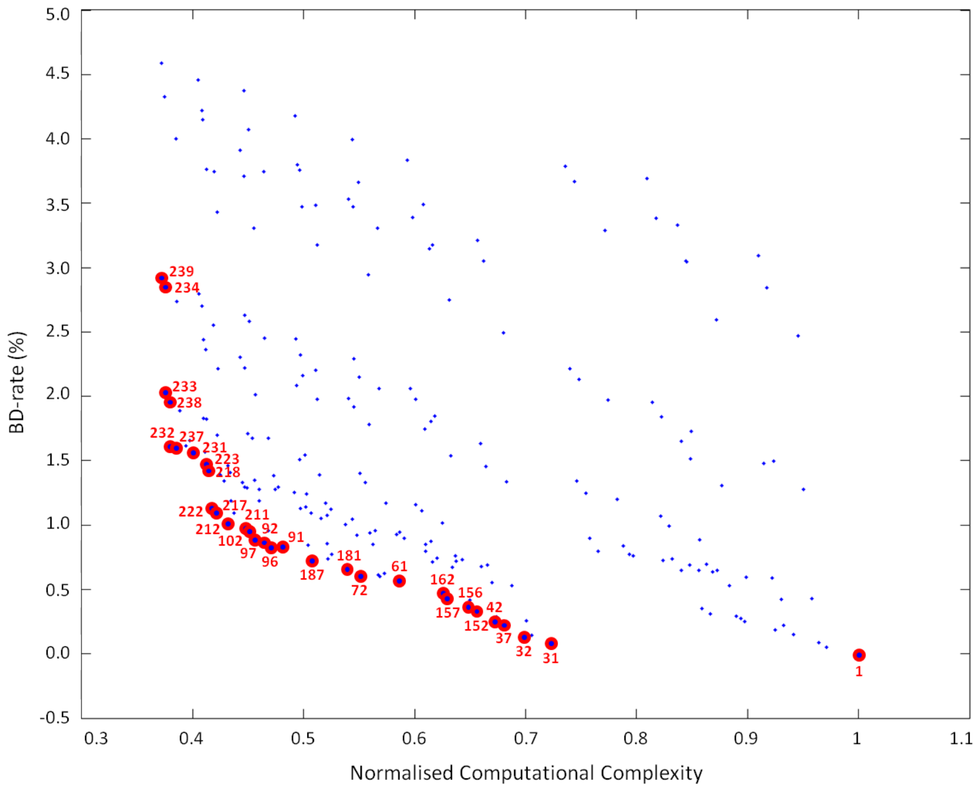


Fig. 7.6: Pareto frontier of the R-C space for the 240 encoder configurations tested.

Table 7.2: Parameters, normalised computational complexity, BD-PSNR and BD-rate for the configurations belonging to the Pareto frontier.

| ***p*** | **Conf.** | ***SR*** | ***BPR*** | ***HME*** | ***CTET*** | ***PUET*** | ***RQTET*** | **Normal. Complex.** | **BD-PSNR (dB)** | **BD-rate (%)** |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 64 | 4 | on | off | off | off | 1 | 0 | 0 |
| 2 | 31 | 64 | 4 | on | on | off | off | 0.723 | -0.002 | 0.086 |
| 3 | 32 | 32 | 4 | on | on | off | off | 0.698 | -0.003 | 0.124 |
| 4 | 37 | 32 | 2 | on | on | off | off | 0.680 | -0.007 | 0.239 |
| 5 | 42 | 32 | 1 | on | on | off | off | 0.673 | -0.006 | 0.256 |
| 6 | 156 | 64 | 2 | on | on | off | on | 0.656 | -0.010 | 0.338 |
| 7 | 152 | 32 | 4 | on | on | off | on | 0.648 | -0.011 | 0.369 |
| 8 | 157 | 32 | 2 | on | on | off | on | 0.628 | -0.012 | 0.424 |
| 9 | 162 | 32 | 1 | on | on | off | on | 0.625 | -0.014 | 0.482 |
| 10 | 61 | 64 | 4 | on | off | on | off | 0.586 | -0.018 | 0.572 |
| 11 | 72 | 32 | 1 | on | off | on | off | 0.551 | -0.018 | 0.600 |
| 12 | 181 | 64 | 4 | on | off | on | on | 0.539 | -0.022 | 0.659 |
| 13 | 187 | 32 | 2 | on | off | on | on | 0.508 | -0.023 | 0.734 |
| 14 | 91 | 64 | 4 | on | on | on | off | 0.481 | -0.026 | 0.823 |
| 15 | 96 | 64 | 2 | on | on | on | off | 0.471 | -0.026 | 0.832 |
| 16 | 92 | 32 | 4 | on | on | on | off | 0.465 | -0.027 | 0.865 |
| 17 | 97 | 32 | 2 | on | on | on | off | 0.455 | -0.027 | 0.879 |
| 18 | 102 | 64 | 1 | on | on | on | off | 0.451 | -0.029 | 0.939 |
| 19 | 211 | 64 | 4 | on | on | on | on | 0.447 | -0.031 | 0.969 |
| 20 | 212 | 32 | 4 | on | on | on | on | 0.431 | -0.032 | 1.018 |
| 21 | 217 | 32 | 2 | on | on | on | on | 0.420 | -0.034 | 1.091 |
| 22 | 222 | 32 | 1 | on | on | on | on | 0.417 | -0.034 | 1.133 |
| 23 | 218 | 16 | 2 | on | on | on | on | 0.415 | -0.045 | 1.425 |
| 24 | 223 | 16 | 1 | on | on | on | on | 0.412 | -0.047 | 1.472 |
| 25 | 231 | 64 | 2 | off | on | on | on | 0.400 | -0.048 | 1.563 |
| 26 | 232 | 32 | 2 | off | on | on | on | 0.384 | -0.049 | 1.601 |
| 27 | 237 | 32 | 1 | off | on | on | on | 0.380 | -0.049 | 1.616 |

indexed by the *p* column. Notice that the BD-PSNR and BD-rate values of configuration 31 are very small and provide average computational complexity reduction of approximately 28%. The remaining configurations yield computational complexity reductions varying from 30% to 62% and incur in BD-PSNR decreases and BD-rate increases of up to 0.049 dB and 1.616%, respectively.

## 7.2 Encoding Time Control System

As Fig. 7.5, Fig. 7.6 and Table 7.2 have shown, there is a very small compression efficiency loss associated to configuration 31, so that if any level of computational complexity reduction is required to maintain encoding times below a determined target, the first encoder operating point should be configuration 31. From configuration 31 onwards, larger encoding time reduction levels can be achieved, but more expressive losses in compression efficiency are noticed. This way, the implementation presented in this section was designed to operate in two modes: the first one, called high-efficiency encoding time reduction (HETR), sets the encoder to operate at a fixed configuration (31), providing a non-adjustable computational complexity reduction; the second, called low-complexity encoding time control (LCTC) mode, adjusts the encoder to operate with configurations that present smaller computational complexity than configuration 31, utilising the encoding time control system described in this section.

Fig. 7.7 presents the detailed diagram of the encoding time control system used in the LCTC mode. All blocks within the controller will be explained in the next sub-sections, using this figure as reference. Section 7.2.1 explains how the system chooses the encoder operating configuration (*Medium-granularity encoding time control* block, in Fig. 7.7) and section 7.2.2 shows how the CCUPU method was combined with it (*Fine-granularity encoding time control* block, in Fig. 7.7). The operations of the *Complexity reduction factor computation* block are explained in both sections 7.2.1 and 7.2.2.

### 7.2.1 Medium-Granularity Time Control

The encoding process starts at full computational complexity and the first GOP is encoded with configuration 1 ($p_{(i)} = 1$, where $i$ is the current GOP index). Then, its encoding time is compared to the target time, which is an input to the algorithm. If there is need for complexity reduction (i.e., the encoding time is larger than the target), the HETR mode is activated and the next configuration in Table 7.2 is chosen for the next GOP (i.e., $p_{(i)} = 2$, configuration 31). The time spent encoding the following GOPs is continuously observed and if further reductions are necessary to reach the target, the LCTC mode is activated and the encoding time control starts.

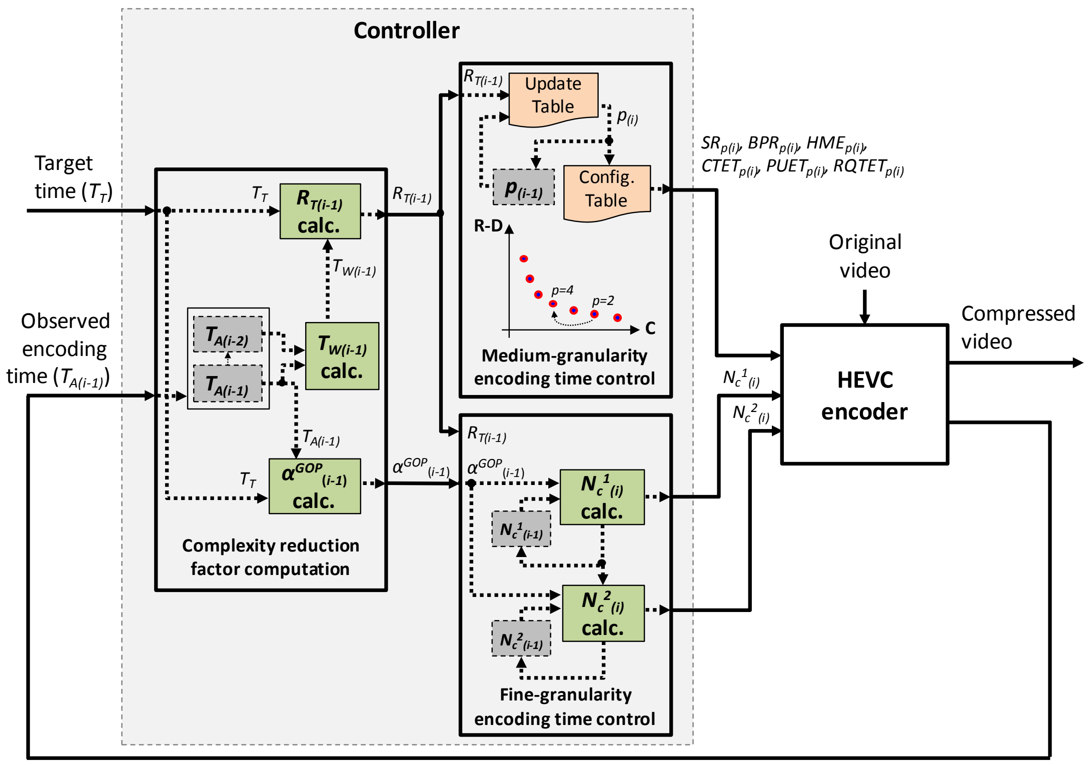


Fig. 7.7: Detailed encoding time control system integrated with the HEVC encoder.

The encoder operating point to be used while encoding the next GOP is chosen in the block named as *Medium granularity encoding time control* in Fig. 7.7, based on the ratio $R_{T(i-1)}$ between the target time ($T_T$) and the observed time ($T_{W(i-1)}$), a weighted average of the encoding times of the last two GOPs, calculated as shown in (Eq. 55) and as illustrated in the block *Complexity reduction factor computation* of Fig. 7.7. The weighted average $T_{W(i-1)}$ is calculated as in (Eq. 56), where $T_{A(i-1)}$ and $T_{A(i-2)}$ are the encoding times of GOPs *i-1* and *i*-2 (*i* is the current GOP index). The dark grey, dashed boxes corresponding to $T_{A(i-1)}$ and $T_{A(i-2)}$ in Fig. 7.7 are memory elements that store the two most recent GOP encoding times. The weights 2 and 1 applied to $T_{A(i-1)}$ and $T_{A(i-2)}$ in the average calculation were manually chosen through experimentation. If the calculated value of $R_{T(i-1)}$ is larger than 1, the encoding time is below the target, so that more computational resources can be employed in the encoding process of the current GOP. On the other hand, if $R_{T(i-1)}$ is smaller than 1, the encoding time must be decreased when encoding the current GOP to achieve the target.

$$R_{T(i-1)} = \frac{T_T}{T_{W(i-1)}} \quad \text{(Eq. 55)}$$

$$T_{W(i-1)} = \frac{2 \cdot T_{A(i-1)} + 1 \cdot T_{A(i-2)}}{3} \quad \text{(Eq. 56)}$$

Once $R_{T(i-1)}$ is computed, it is used to index a look-up table (LUT) containing the pre-calculated ratios between the normalised computational complexities of the configurations presented in Table 7.2 (*Normal. Complex.* column in the table) to determine which one should be used in the current GOP to adjust the encoding time. The pre-computed LUT is presented in Table D.2 of Appendix D and is illustrated as *Update Table* in Fig. 7.7. In Table D.2, the value of each cell is calculated as the ratio between the normalised computational complexity (shown in Table 7.2) of those configurations with *p* index indicated by the column and the line number in Table D.2. For example, the ratio between configurations with *p* index 3 and 2 in Table 7.2 is 0.965 and this value is shown in the cell located at the intersection of column 3 and line 2 in the LUT. This ratio means that configuration 3 is 3.5% less complex than configuration 2, and this information is used for complexity scaling. Both the values of $R_{T(i-1)}$ and the encoder operating point used in the previous GOP ($p_{(i-1)}$) are used to index the LUT. The $p_{(i-1)}$ value indicates the line where the closest value to $R_{T(i-1)}$ must be searched. Once it is found, the column number where it is located indicates the $p_{(i)}$ operating point to be used in the current GOP.

As an example of operation, let us consider that the encoder operated in configuration 72 (*Normal. Complex.* = 0.551, according to Table 7.2) in the previous GOP, which is indexed by $p_{(i-1)} = 11$. If (Eq. 55) returns, for example, an $R_{T(i-1)}$ value equals to 0.9, it means that the encoding time was about 11% above the target. By looking at line 11 ($p_{(i-1)} = 11$) in Table D.2, it is possible to see that the closest value to 0.9 in that line is 0.922, which is located in column 13. This means that configuration 13 yields a computational complexity reduction of 7.8% in relation to configuration 11, as can be confirmed in Table 7.2 ($p = 13$ yields a normalised computational complexity of 0.508, which is 7.8% smaller than 0.551). The value of $p_{(i)}$ is thus set to 13.

With the new configuration index $p_{(i)}$, a second LUT called *Configuration Table* in Fig. 7.7 is accessed to find the encoding parameters to be used in the current GOP. The *Configuration Table* is simply composed of the *p*, *SR*, *BPR*, *HME*, *CTET*, *PUET* and *RQTET* columns of Table 7.2. The value of $p_{(i)}$ is then used to index the table in the *p* column and the parameter values are retrieved and used to reconfigure the HEVC encoder ($SR_{p(i)}$, $BPR_{p(i)}$, $HME_{p(i)}$, $CTET_{p(i)}$, $PUET_{p(i)}$ and $RQTET_{p(i)}$, in Fig. 7.7).

Fig. 7.8 presents the pseudo-code for the algorithm implemented in HM to allow medium-granularity encoding time control. From lines 01 to 12, the algorithm monitors whether any encoding time reduction is needed. In line 01, the configuration index $p_{(i)}$ is set to 1, so that no encoding time reduction is applied to the first GOP. If encoding time reduction is never required (i.e., the test in line 05 always returns *false*), the $p_{(i)}$ index will keep set to 1 during all the encoding process, so that the unmodified encoding process will always take place. Otherwise, whenever the algorithm detects that an encoding time reduction is required (i.e., the test in line 05 returns *true*), the HETR mode is activated and $p_{(i)}$ is set to 2 (line 07), which means that configuration 31 of Table 7.2 is used to encode the next GOP.

```
encode(video, $T_T$):
 01   $i \leftarrow 1$, $p_{(i)} \leftarrow 1$
 02   encode GOP$_{(i)}$ with SR$_{p(i)}$, BPR$_{p(i)}$, HME$_{p(i)}$,
                     CTET$_{p(i)}$, PUET$_{p(i)}$, RQTET$_{p(i)}$
 03   $T_{A(i)} \leftarrow$ time spent to encode GOP$_{(i)}$
 04   $i \leftarrow i + 1$
 05   if($T_{A(i-1)} > T_T$)
 06      if($p_{(i-1)} = 1$)                // HETR mode
 07         $p_{(i)} \leftarrow 2$
 08         go to line 02
 09      else                               // LCTC mode
 10         go to line 13
 11   else
 12      go to line 02
 13   calculate $R_{T(i-1)}$
 14   $p_{(i)} \leftarrow$ the $p$ configuration indexed by $R_{T(i-1)}$
 15   encode GOP$_{(i)}$ with SR$_{p(i)}$, BPR$_{p(i)}$, HME$_{p(i)}$,
                     CTET$_{p(i)}$, PUET$_{p(i)}$, RQTET$_{p(i)}$
 16   $T_{A(i)} \leftarrow$ time spent to encode GOP$_{(i)}$
 17   $i \leftarrow i + 1$
 18   go to line 13
```

Fig. 7.8: Pseudo-code for the medium-granularity encoding time control.

If further complexity reductions are required after the HETR mode is already active (i.e., the test in line 05 returns *true* and $p_{(i-1)}$ is set to 2), the LCTC mode is enabled and a new configuration $p_{(i)}$ is chosen from the *Update Table* based on the ratio $R_{T(i-1)}$ between the target time and the average encoding time of the two previous GOPs, calculated as per (Eq. 55) and (Eq. 56) (see lines 13-14). The new $p_{(i)}$ index is then used to access the parameters corresponding to its corresponding configuration, which are used to reconfigure the HEVC encoder for the next GOP (line 15). The encoding time keeps being monitored for further adjustments in the encoder operating point (line 16) and the process is repeated at the beginning of the next GOP.

The encoding time control was evaluated using four different target times, defined with reference to an average encoding time per GOP computed for four high-resolution test video sequences (*Kimono1*, *PoznanStreet*, *Shark*, *Tennis*[7], detailed in Appendix B) that were encoded with four QPs 22, 27, 32, and 37, utilising the HETR mode. Based on the average encoding time per GOP of these 16 encodings (333s), four target times per GOP were defined: 66s (20% of 333s), 133s (40%), 200s (60%), and 266s (80%). Notice that these encoding time limits were computed with reference to the encoding time measured when using the HETR mode only for test purposes. The only requirement observed during their choice was that their values should be of about the same order of magnitude or one order of magnitude below the times for the HETR mode. In a real-case implementation of an HEVC encoder, the time constraints will be imposed, for example, by computational resources constraints, transmission and/or memory limitations, user preferences and real-time encoding requirements.

Fig. 7.9 shows, as an example of operation of the algorithm, the encoding time per GOP for the *Kimono1* sequence (QP 32). Target times are presented as dotted lines and actual encoding times are presented as solid lines. Besides the encoding times observed for each of the four target times, encoding times corresponding to the original encoder and the encoder operation with the HETR mode are also presented for comparison. The *Kimono1* sequence was chosen for this analysis because it contains a scene change at GOP 34, providing a more complete analysis on the operation of the

[7] Video sequence *Tennis* is not included in the most recent version of the CTC document [43]. However, in order to allow tests with high-resolution sequences that were not used in the training of the decision trees and in the parameter selection presented in section 7.1, it has been used in these evaluations. Details on the *Tennis* sequence are available in Appendix B.

algorithm. It is possible to notice in the figure that the smallest target time (66s) cannot be achieved between GOPs 1 and 36. In fact, the smallest encoding times achieved for these GOPs are around 110s. This limitation happens because the 27 configurations listed in Table 7.2 do not allow reducing encoding times to values below 110s for that particular video segment. However, when a scene change occurs and the video encoding process becomes less complex due to characteristics of the video sequence, encoding times below 110s can be achieved.

This effect can be better observed in Fig. 7.10, which shows the evolution of the $p_{(i)}$ index (i.e., the encoding configuration index) for the same sequence presented in Fig. 7.9. The curve corresponding to the 66s target time saturates with $p_{(i)} = 27$ in most of the chart of Fig. 7.10, which means that the maximum computational complexity downscaling was applied, but even so the target time could not be reached, at least until GOP 36, as shown in Fig. 7.9. Further complexity reduction ratios are achieved with the fine-granularity encoding time control introduced in the next section.

Fig. 7.9 also shows a ripple effect for the 266s target time case. The effect is also visible Fig. 7.10 for the same target time. This happens because the controller is not able to find a configuration among the 27 in the LUT that yields an encoding time close enough to the target. In such cases, the encoder alternates between two configurations: one that yields an encoding time above the target and another that yields an encoding time under the target. The fine-granularity encoding time control presented in the next section also attenuated this problem.

### 7.2.2 Fine-Granularity Time Control

Section 5.6 of chapter 5 presented the CCUPU method, which yielded the best complexity scaling accuracy among all methods proposed in that chapter, as well as the best R-D efficiency results. Now, this section describes how the CCUPU method was integrated with the encoding time control system presented in section 7.1.2, aiming at providing a finer granularity level to the control algorithm and a wider range of achievable target times.

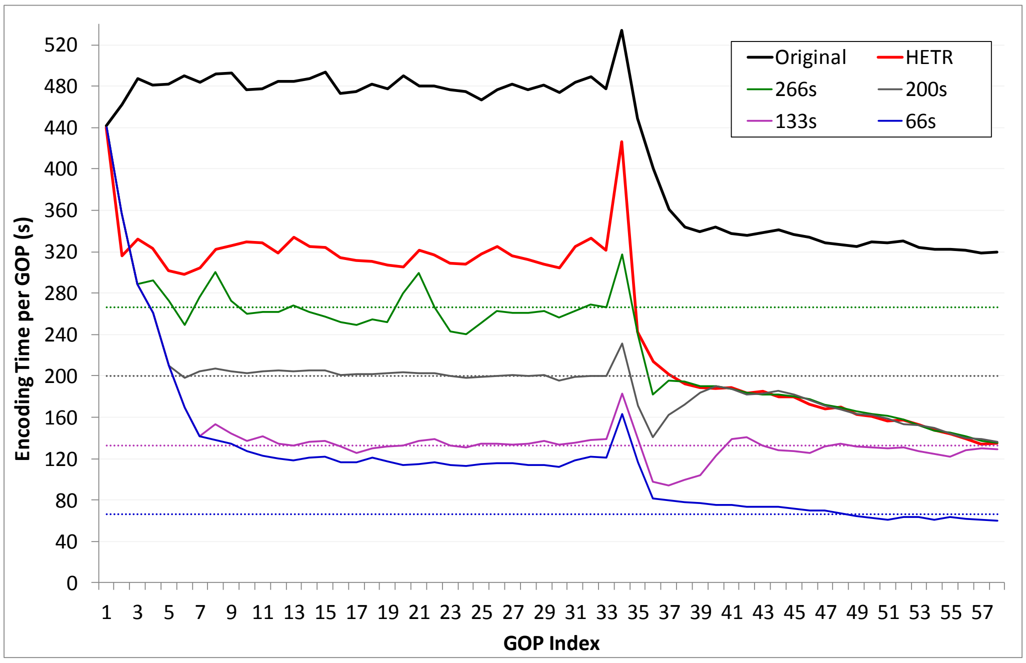


Fig. 7.9: Encoding times per GOP for the *Kimono1* sequence (QP 32) encoded with different target times using the medium-granularity encoding time control.

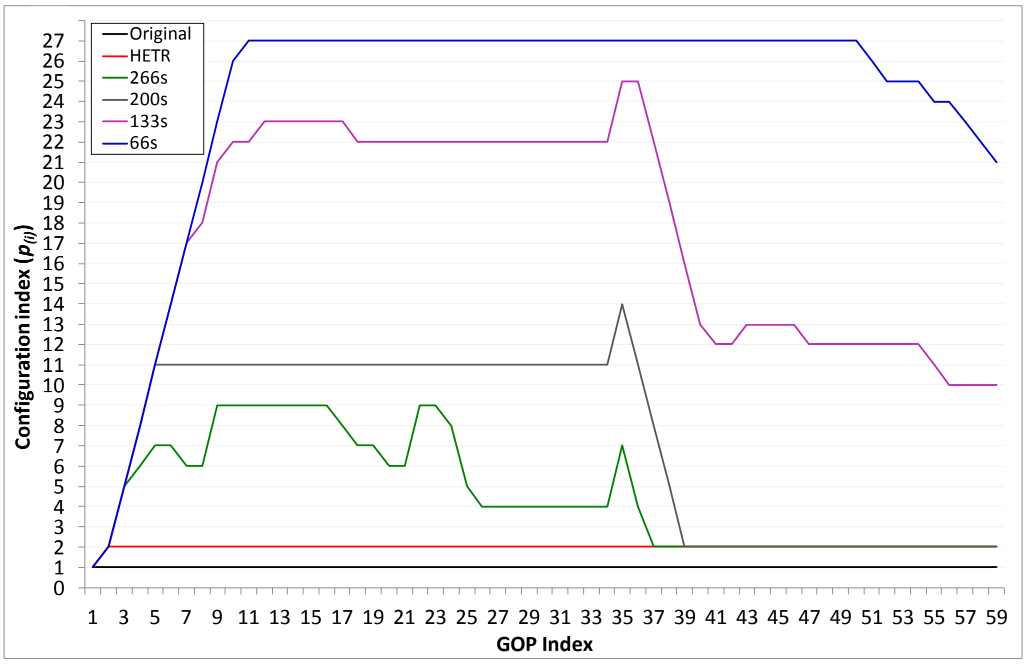


Fig. 7.10: Encoder configurations used to encode the *Kimono1* sequence (QP 32) with different target times using the medium-granularity encoding time control.

As shown in Fig. 7.7, the *Fine-granularity encoding time control* block receives as input the $\alpha^{GOP}{}_{(i\text{-}1)}$ parameter, which was calculated in the *Complexity reduction factor* *computation* block. The $\alpha^{GOP}_{(i-1)}$ parameter is based on the $\alpha^{GOP}$ adjusting parameter used in CCUPU (see section 5.6), which is the ratio between the encoding time of the previous GOP ($T_{A(i-1)}$) and the target time ($T_T$), as shown in (Eq. 57). The value of $\alpha^{GOP}_{(i-1)}$ is then used to adjust the number $N_c^k{}_{(i)}$ of constrained CTUs per frame in the current GOP *i*, as shown in (Eq. 58), where *k* indicates which of the two CCUPU complexity constraining parameters is used[8]. Once $N_c^1{}_{(i)}$ reaches its limit (i.e., the number of CTUs in a frame), the second parameter in the scheme is used, represented as $N_c^2{}_{(i)}$.

$$\alpha^{GOP}_{(i-1)} = \frac{T_{A(i-1)}}{T_T} \qquad \text{(Eq. 57)}$$

$$N^k_{c\,(i)} = \alpha^{GOP}_{(i-1)} \cdot N^k_{c\,(i-1)} \qquad \text{(Eq. 58)}$$

The algorithm presented in Fig. 7.8 for medium-granularity encoding time control was extended as shown in Fig. 7.11 to operate in two adjustment phases (dashed boxes). The first lines of the algorithm are identical to those of Fig. 7.8, except for the initialization of the $N_c^1{}_{(i)}$ and $N_c^2{}_{(i)}$ variables, which represent the two complexity adjusting parameters of the CCUPU algorithm, as explained in section 5.6. In line 13, the ratio $R_{T(i-1)}$ is calculated as in (Eq. 55) and its value is used to trigger either the medium or the fine-granularity encoding time control. If the $R_{T(i-1)}$ ratio shows that an adjustment larger than 15% (either positive or negative) is required, the medium-granularity control described in section 7.2.1 takes place. Otherwise, the fine-granularity control based on the CCUPU algorithm starts. The 15% threshold was defined through experimental tests. As the value of $R_{T(i-1)}$ is used to trigger the operation of either the medium or the fine-granularity encoding time control, it is shown as an input of both blocks in Fig. 7.7.

As detailed in section 5.6, in the fine-granularity control, constrained CTUs are distributed according to the R-D costs of co-localised CTUs in the previous frame (lines 23-30 in Fig. 7.11). Notice that when the fine-granularity control is applied, the encoding operating point chosen in the last execution of the medium-granularity control is used, so that the parameter values are still retrieved and used to reconfigure the HEVC encoder according to the $p_{(i)}$ index, as shown in line 30.

[8] Recall that in the CCUPU method, $N_c^1$ represents the number of CTUs that allow using PUs smaller than *2N×2N* and $N_c^2$ is the number of CTUs with maximum coding tree depth constrained according to the depth used in spatially and temporally neighbouring CTUs.

```
encode(video, $T_T$):
01  $i \leftarrow 1$, $p_{(i)} \leftarrow 1$, $N_c^1{}_{(i)} \leftarrow 0$, $N_c^2{}_{(i)} \leftarrow 0$
02  encode GOP$_{(i)}$ with SR$_{p(i)}$, BPR$_{p(i)}$, HME$_{p(i)}$, CTET$_{p(i)}$, PUET$_{p(i)}$, RQTET$_{p(i)}$
03  $T_{A(i)} \leftarrow$ time spent to encode GOP$_{(i)}$
04  $i \leftarrow i + 1$
05  if($T_{A(i-1)} > T_T$)
06     if($p_{(i-1)} = 1$)              // HETR mode
07        $p_{(i)} \leftarrow 2$
08        go to line 02
09     else                         // LCTC mode
10        go to line 13
11  else
12     go to line 02
13  calculate $R_{T(i-1)}$                                  medium-granularity
14  if(($R_{T(i-1)} > 1.15$) OR ($R_{T(i-1)} < 0.85$))      complexity scaling
15     $p_{(i)} \leftarrow$ the $p$ configuration indexed by $R_{T(i-1)}$
16     $N_c^1{}_{(i)} \leftarrow 0$, $N_c^2{}_{(i)} \leftarrow 0$
17  else                                                    fine-granularity
18     calculate $\alpha^{GOP}{}_{(i-1)}$                    complexity scaling
19     if($N_c^1{}_{(i-1)} < nCTU$)
20        calculate $N_c^1{}_{(i)}$
21     else
22        calculate new $N_c^2{}_{(i)}$
23  for each $i$ from 0 to $nFR$
24    sort CTUs in ascending order of R-D cost
25    for each $j$ from 0 to $nCTU$
26      if($j < N_c^k$)
27        mark CTU $j$ as constrained
28      else
29        mark CTU $j$ as unconstrained
30      encode CTU $j$ with SR$_{p(i)}$, BPR$_{p(i)}$, HME$_{p(i)}$, CTET$_{p(i)}$, PUET$_{p(i)}$, RQTET$_{p(i)}$
31    if last frame go to line 01
32  $T_{A(i)} \leftarrow$ time spent to encode GOP$_{(i)}$
33  $i \leftarrow i + 1$
34  go to line 13
```

Fig. 7.11: Pseudo-code for the fine-granularity encoding time control.

## 7.3 Experimental Results

The encoding time control accuracy was evaluated as previously described in section 7.2.1 by testing its behaviour with four different target times (66s, 133s, 200s, 266s) and the same four video sequences (*Kimono1*, *PoznanStreet*, *Shark*, *Tennis*) with QPs 22, 27, 32, and 37.

Table 7.3-Table 7.6 show results in terms of average encoding time control accuracy, R-D efficiency and encoding time reduction for the four video sequences and the four target times tested. R-D efficiency and average encoding time reductions were measured in terms of BD-rate and BD-PSNR using the original HM encoder as reference. Encoding time control accuracy was measured in two ways: the first one ($TE_{ALL}$) calculates the average difference between encoding time and target time considering all GOPs in a video sequence after the settling phase of the control, while the second measure ($TE_{LTT}$) calculates the average difference between encoding and target times considering only those GOPs that yielded an encoding time larger than the target. In both cases, the differences are normalised with reference to the target time. Although $TE_{ALL}$ shows smaller errors, $TE_{LTT}$ provides a fairer measure between target and encoding time, since we are only taking into account the undesirable cases where the actual encoding time was larger than the target time. On average, the $TE_{ALL}$ and $TE_{LTT}$ errors are 4.1% and 8.9%, respectively.

Notice that the four tables also present results for the HETR case, which is not associated to any target. In this case, the $TE_{ALL}$ and $TE_{LTT}$ values were calculated as the average absolute differences between the encoding time per GOP and the average encoding time among all GOPs in the sequence (in other words, $TE_{ALL}$ and $TE_{LTT}$ are encoding time absolute deviations in the case of HETR). In all tables, the deviations noticed for HETR are larger than the $TE_{ALL}$ and $TE_{LTT}$ errors for the LCTC cases tested, which means that the system is capable of controlling the encoding time with reduced variation around the target. In the worst case (*Tennis* sequence, 66s, in Table 7.6), the errors $TE_{ALL}$ and $TE_{LTT}$ for LCTC are 8.27% and 20.01%, which are still smaller than the deviations observed with HETR (10.84% and 29.96%).

Table 7.3: Encoding time errors, BD-rate, and BD-PSNR for the HETR and the LCTC modes (four target times) for the *Kimono1* sequence.

| **Mode and Target Time** | **Average $TE_{ALL}$ (%)** | **Average $TE_{LTT}$ (%)** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|---|
| HETR | 10.17 | 24.57 | +0.87 | -0.028 |
| LCTC@266s | 1.59 | 4.40 | +0.90 | -0.028 |
| LCTC@200s | 0.45 | 2.40 | +2.00 | -0.066 |
| LCTC@133s | 2.82 | 7.43 | +4.03 | -0.123 |
| LCTC@66s | 6.14 | 10.95 | +10.39 | -0.310 |

Table 7.4: Encoding time errors, BD-rate, and BD-PSNR for the HETR and the LCTC modes (four target times) for the *PoznanStreet* sequence.

| **Mode and Target Time** | **Average $TE_{ALL}$ (%)** | **Average $TE_{LTT}$ (%)** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|---|
| HETR | 6.30 | 14.79 | +1.35 | -0.049 |
| LCTC@266s | 0 | 0 | +1.35 | -0.049 |
| LCTC@200s | 0.91 | 4.10 | +2.93 | -0.107 |
| LCTC@133s | 3.39 | 7.85 | +3.09 | -0.112 |
| LCTC@66s | 4.89 | 8.63 | +5.71 | -0.201 |

Table 7.5: Encoding time errors, BD-rate, and BD-PSNR for the HETR and the LCTC modes (four target times) for the *Shark* sequence.

| **Mode and Target Time** | **Average $TE_{ALL}$ (%)** | **Average $TE_{LTT}$ (%)** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|---|
| HETR | 8.09 | 15.15 | +0.26 | -0.012 |
| LCTC@266s | 7.70 | 13.53 | +1.87 | -0.085 |
| LCTC@200s | 6.01 | 10.03 | +2.97 | -0.134 |
| LCTC@133s | 5.20 | 9.74 | +6.48 | -0.282 |
| LCTC@66s | 4.98 | 9.34 | +15.12 | -0.622 |

Table 7.6: Encoding time errors, BD-rate, and BD-PSNR for the HETR and the LCTC modes (four target times) for the *Tennis* sequence.

| **Mode and Target Time** | **Average $TE_{ALL}$ (%)** | **Average $TE_{LTT}$ (%)** | **BD-rate (%)** | **BD-PSNR (dB)** |
|---|---|---|---|---|
| HETR | 10.84 | 29.96 | +0.77 | -0.024 |
| LCTC@266s | 3.74 | 9.87 | +2.64 | -0.073 |
| LCTC@200s | 3.69 | 9.31 | +4.88 | -0.138 |
| LCTC@133s | 5.72 | 15.09 | +7.67 | -0.216 |
| LCTC@66s | 8.27 | 20.01 | +18.64 | -0.505 |

It is also necessary to comment on the result for target 266s in Table 7.4, which shows both errors equal to zero. This happens because in that case the target time of 266s is larger than the encoding time observed with the HETR mode in all GOPs, so that the LCTC mode is never actually activated and no encoding time control is performed. For the same reason, both LCTC@266s and HETR cases present the same BD-rate and BD-PSNR results, since they performed exactly the same encoding operations.

As expected, in the remaining cases from Table 7.3 to Table 7.6, larger encoding times incurred in larger decreases in R-D efficiency: overall, the HETR mode yielded BD-rate increases from 0.26% to 1.35% only, while the LCTC mode presented BD-rate increases varying from 0.9% to 18.64%, depending on the target time.

Fig. 7.12 shows, as a comparison to Fig. 7.9, the encoding time per GOP for the *Kimono1* sequence (QP 32). It is possible to perceive in Fig. 7.12 that the smallest target time tested (66s) can be achieved now, differently from Fig. 7.9. This is only possible because, differently from the medium-granularity encoding time control, the fine-granularity control allows further encoding time reductions by increasing the number of constrained CTUs per frame with the $N_c^1{}_{(i)}$ and $N_c^2{}_{(i)}$ parameters. Fig. 7.12 also shows that the ripple effect noticed in Fig. 7.9 was decreased with the fine-granularity control.

The algorithm operation for the given example can be observed in Fig. 7.13 and Fig. 7.14, which show the evolution of the $p_{(i)}$ index and the $N_c^1{}_{(i)}$ and $N_c^2{}_{(i)}$ parameters, respectively. Notice that even though the curve corresponding to the 66s target time still saturates with $p_{(i)} = 27$ in most of the chart of Fig. 7.13, the encoding time per GOP can be further reduced by adjusting the $N_c^1{}_{(i)}$ and $N_c^2{}_{(i)}$ parameters, as shown in Fig. 7.14. Fig. 7.14(b) shows a very small increase of $N_c^2{}_{(i)}$ in GOP 36, which happens when the saturation of $N_c^1{}_{(i)}$ occurs in Fig. 7.14(a). In the remaining target time cases, only the $N_c^1{}_{(i)}$ parameter is used to adjust the encoding time at a fine level of granularity.

## 7.4 Conclusions

This chapter presented an encoding time control system for HEVC encoders that was designed with the aim of maintaining the encoding time per GOP below a target limit. As previously explained, the main goal of this chapter was to integrate the methods and results described in the previous chapters of this dissertation into an encoder control system in order to exemplify their applicability.

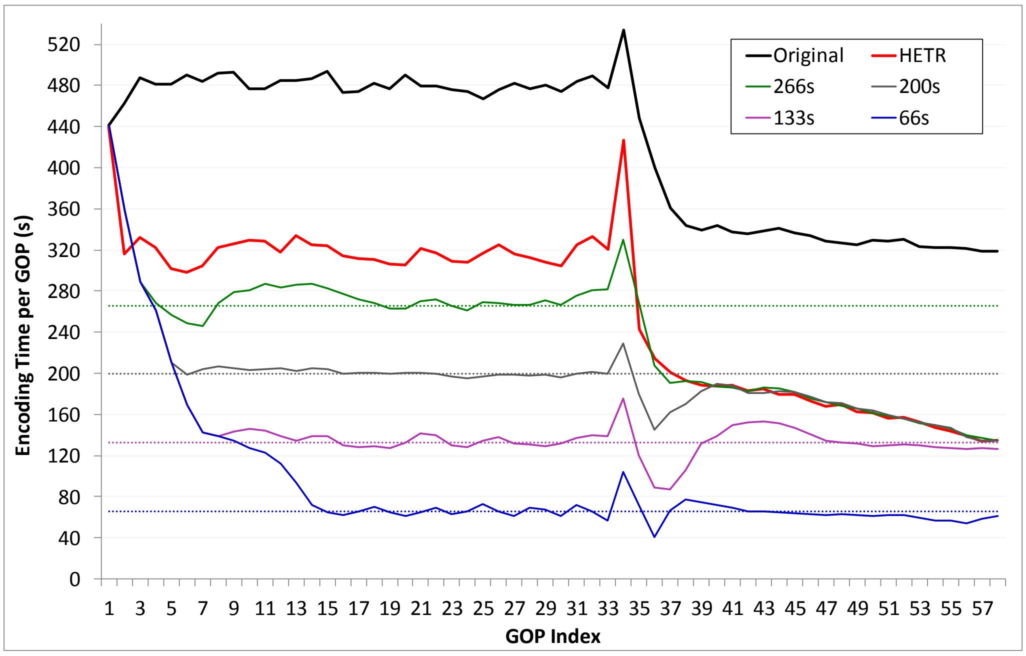


Fig. 7.12: Encoding times per GOP for the *Kimono1* sequence (QP 32) encoded with the fine-granularity encoding time control.

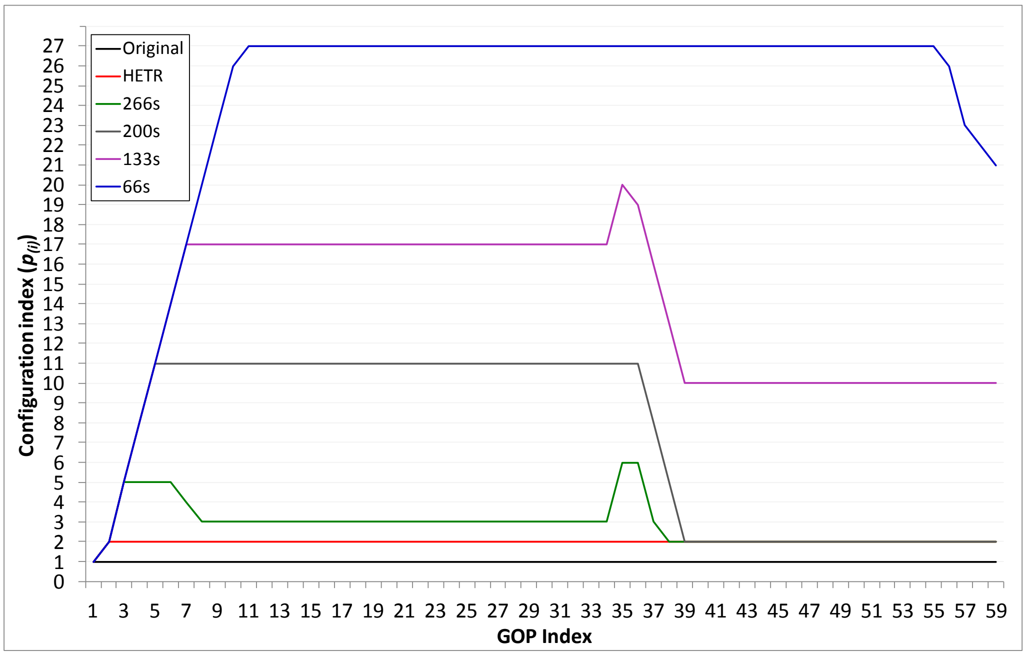
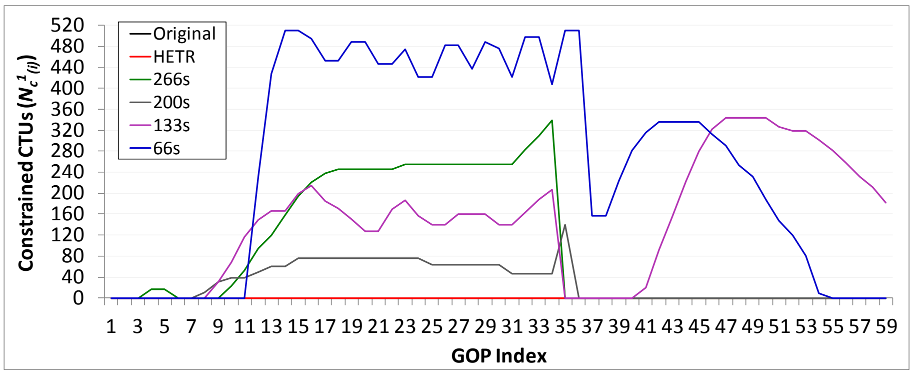
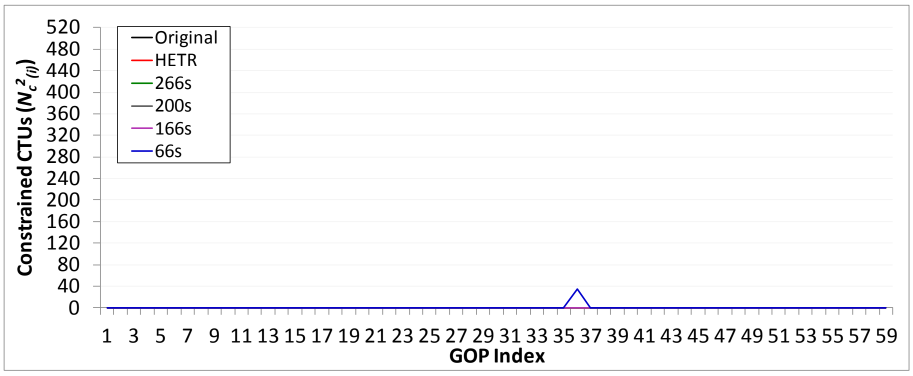


Fig. 7.13: Encoder configurations used to encode the *Kimono1* sequence (QP 32) with different target times with the fine-granularity control.



(a)



(b)

Fig. 7.14: Variation of the (a) $N_c^1{}_{(i)}$ and (b) $N_c^2{}_{(i)}$ parameters when encoding the *Kimono1* sequence (QP 32) with the fine-granularity control.

The proposed system was designed based on the findings of chapter 4 and the methods proposed for complexity scaling and reduction in chapters 5 and 6. An offline training process was performed to determine the points that compose the Pareto frontier belonging to the R-C and D-C spaces of a set of encoding configurations, with the goal of providing high R-D-C efficiency when computational complexity restrictions are required. The encoding configurations tested make use of the parameters identified in chapter 4 as those related to the most complexity-demanding tools and the schemes proposed in chapter 6 for complexity reduction of HEVC encoders. By selecting the configurations that yield the smallest R-D efficiency losses among all those analysed, it was possible to implement a system with two R-D efficiency operating modes: one with high encoding efficiency but limited complexity reduction ratio (HETR) and another with low computational complexity but increased R-D efficiency loss (LCTC), which made use of a medium-granularity encoding time control. Finer encoding time control granularity was obtained by integrating the CCUPU method proposed in chapter 5.

Experimental results have shown that the encoding time per GOP can be controlled at a medium to fine granularity level, providing an average error between encoding time and target time of 4.1% or 8.9%, depending on the measure adopted for comparison.

# Chapter 8

# Conclusions

The research work developed in the scope of this thesis contributed with novel solutions for reducing and scaling the computational complexity of high efficiency video encoders. The study included extensive simulations and experiments focused on the study and critical analysis of computational complexity for HEVC. In this chapter, this thesis is concluded with a summary of the main achievements, as well as some directions for future research.

## 8.1 Final Remarks

The HEVC standardisation process, which started in early 2010 and was finalised in 2013, incorporated a series of encoding tools and functionalities to the basic hybrid video compression system used in the most recent previous standards. By employing such tools and functionalities, HEVC is currently able to achieve average bit rate reductions of 40%-50% in comparison with H.264/AVC, at the same subjective image quality. However, as chapter 4 has shown, these improvements came with associated increases in the encoder computational complexity.

At the beginning of the research work presented in this thesis, there was no work published in the technical literature focusing on computational complexity analysis, reduction or scaling for HEVC. This motivated the study and analysis of the compression efficiency and computational complexity required by each encoding tool and functionality, as presented in chapter 4, in order to identify which are the most computationally demanding operations in the encoding process. Based on the results of such analysis, a set of methods for complexity reduction and scaling were proposed in chapters 5, 6 and 7.

## 8.2 Research Contributions

The first main contribution of this thesis is presented in chapter 4 and consists of an experimental investigation and analysis of the R-D efficiency and computational complexity of HEVC encoders. Two separate analyses were performed to evaluate the encoding tools and the frame partitioning structures introduced by the HEVC standard. A set of encoding configurations was created to investigate the impact of each tool or frame partitioning structure, varying the encoding parameter set and comparing the results with a baseline encoder.

The results obtained in chapter 4 allowed concluding that HEVC complexity can be largely decreased at practically no coding efficiency cost, if the coding tools are wisely combined and configured. It was observed that by first enabling those tools which provide higher R-D efficiency gains for the least computational complexity costs, a near-optimal trade-off between computational complexity and encoding efficiency can be achieved. An encoder configuration with high R-D efficiency and low computational complexity levels was proposed. Besides, a set of configurations which resulted in the largest computational complexity increases were identified for use in the R-D-C optimised scheme presented in chapter 7. Still in chapter 4, it was also found that the encoding computational complexity can be thoroughly reduced by managing the frame partitioning structures of HEVC, even though some configurations incur in much larger losses in R-D efficiency than others. The frame partitioning structures that produce the largest impact in the computational complexity and the best trade-off between complexity and R-D efficiency were selected for the development of the complexity reduction and scaling strategies presented in chapters 5 and 6.

The complexity scaling algorithms proposed in chapter 5 represent the second main contribution of this thesis. All developed methods aim at dynamically adjusting the frame partitioning structures (CTUs, CUs and PUs) in order to adapt the encoding process according to the available computational complexity in the encoder. Five algorithms were proposed in this chapter (FDCR, VDCR, MCTDL, CTDE, and CCUPU) and each one overcame the previous in terms of R-D efficiency for the same target complexities, which is an outcome obtained by adding up more intelligent ways of constraining the frame partitioning structures of HEVC. In general, the complexity scaling accuracy of the five algorithms is quite similar and all of them achieve running complexities very close to the targets, even though the last method (CCUPU) provides much larger computational complexity reduction levels than the previous ones. While FDCR, VDCR, MCTDL and CTDE provide computational complexity reductions of up to 40%, the CCUPU method is able to reduce it in up to 80%. In terms of R-D efficiency, CCUPU operates in two regions: a near-lossless encoding, where complexity reductions of up to 50% are achieved at the cost of small BD-rate increases (from 0.03% to 1.28%), and a lossy region, where a complexity reduction of up to 80% is achieved with a BD-rate increase between 3.98% and 22.64%. From the results obtained in chapter 5, it was possible to notice that adding up information related to spatial and temporal correlation to the last complexity scaling algorithms decreased the R-D efficiency losses noticed in the first ones. This lead to the conclusion that if more information from the original video, as well as intermediate information computed during the encoding process, were taken into account in the development of complexity reduction and scaling methods, R-D efficiency losses even smaller than those observed with CCUPU could be achieved.

The third major contribution of this thesis was presented in chapter 6. A set of classification trees obtained through data mining techniques was developed and implemented in the HM encoder to early terminate the exhaustive search for the best frame partitioning structure configuration. The three sets of decision trees were created to make use of intermediate encoding results for early terminating the determination of Coding Trees, PUs and RQTs. They were separately and jointly implemented to provide further complexity reduction levels. On average, a complexity reduction of 65% was achieved when the three early terminations are jointly implemented, with a BD-rate increase of only 1.36%.

However, the early termination methods proposed in chapter 6 provide fixed complexity reductions, differently from the dynamic complexity scaling methods presented in chapter 5. Besides, they do not guarantee that the encoding process is performed within a determined time budget. In order to solve this issue, chapter 7 presents the fourth contribution of this thesis: an encoding time control system that combines the findings of chapter 4, the best-performing complexity scaling method of chapter 5 and the complexity reduction algorithms of chapter 6, aiming at adjusting the encoder operating point whenever necessary so that the encoding time per GOP is kept under a specified target.

## 8.3 Future Work

Several research directions can be thought for the future, departing from the work presented in this thesis. One of the most important and immediate works to be done as a continuation of this research is the implementation of the best proposed methods (for example, the CCUPU algorithm, the decision trees and the encoding time control system) in an optimised encoder developed for real time applications. As previously explained, all the algorithms presented in this thesis were tested in the HM encoder, which was developed during the standardisation process for testing purposes only. However, the proposed methods are expected to provide similar complexity reduction rates in other optimised solutions that employ the RDO process to take decisions.

Another possible solution to be explored in future work is the exploration of parallel computing strategies aiming at the achievement of real-time video coding. It is now common to find handheld devices equipped with multicore general processors and graphics processing units (GPUs), which can be used to increase encoding speed through the divide-and-conquer approach. As explained in chapter 2, the HEVC standard includes a set of parallel processing structures (Dependent Slice Segments, Tiles, WPP) which can be explored to solve such issues. The algorithms proposed in this thesis, mostly based on the management of the HEVC frame partitioning structures, can be extended in future works to support the parallel processing structures, so that computational complexity could be reduced or scaled separately in each core.

All the analyses, algorithms and methods proposed in this thesis focused on the encoding process, which is the most critical issue in terms of computational complexity in HEVC-based codec systems. However, the use of computational resources on the decoder side will also become more important with the introduction of higher spatial resolution, such as ultra-HD. Techniques for computational complexity reduction and scaling on video decoders will need to be further investigated, especially for devices with fewer computational capabilities. Furthermore, the heterogeneity of mobile devices, varying from those with fast multicore processors and GPUs to those with slower single-core processors, will also call for methods that allow scaling computational complexity on the decoder side according to the target platform constraints. In [132], the authors propose a model that allows an encoder to perform different encoding operations, so that the generated bitstream is suitable for a determined receiver platform with a certain constraint in terms of computational resources. However, solutions on this range aiming at HEVC decoders are still absent.

# Appendix A

# Ph.D. Publications

The Ph.D. project presented in this thesis resulted in 16 peer-reviewed submissions (13 accepted and three currently under evaluation), which are listed in the following lines. They include one book chapter, four journal articles and 11 conference papers.

## Book Chapter

I. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Computational Resource Management for Video Coding in Mobile Environments," in *Modeling and Optimization in Science and Technologies (Resource Management in Mobile Computing Environment)*, 1st ed., vol. 3, Berlin, Germany: Springer International Publishing, chapter 24, pp. 515-549, 2014.

## Journal Articles

I. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Fast HEVC Encoding Decisions Using Data Mining". *IEEE Transactions on Circuits and Systems for Video Technology*, submitted: May 4, 2014, revised: July 15, 2014.

II. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Complexity Scalability in Real Time HEVC Encoders". *Journal of Real-Time Image Processing*, January 2014, pp. 1-16, 2014 (online early access).

III. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Performance and Computational Complexity Assessment of High Efficiency Video Encoders”. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, issue 12, pp. 1899-1909, 2012.

IV. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Complexity Control of High Efficiency Video Encoders for Power-Constrained Devices”. *IEEE Transactions on Consumer Electronics*, vol. 57, issue 4, pp. 1866-1874, 2011.

## Conference Papers

I. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Four-step Algorithm for Early Termination in HEVC Inter-frame Prediction based on Decision Trees”. *Visual Communications and Image Processing (VCIP 2014)*, Valleta, Malta, submitted: May 26, 2014.

II. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Classification-Based Early Termination for Coding Tree Structure Decision in HEVC”. *IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2014)*, Marseille, France, submitted: June 13, 2014.

III. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “A Method for Early-Splitting of HEVC Inter Blocks Based on Decision Trees”. *European Signal Processing Conference (EUSIPCO 2014)*, Lisbon, Portugal, 2014.

IV. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Constrained Encoding Structures for Computational Complexity Scalability in HEVC”. *Picture Coding Symposium (PCS 2013)*, San Francisco, USA, 2013.

V. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Computational Complexity Control for HEVC Based on Coding Tree Spatio-Temporal Correlation”. *IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2013)*, Abu Dhabi, UAE, 2013.

VI. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. “Complexity Control of HEVC through Quadtree Depth Estimation”. *IEEE Region 8 Conference (EUROCON 2013)*, Zagreb, Croatia, 2013.

VII. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Maximum Coding Tree Depth Adjustment for Complexity Scalability of HEVC". *Conference on Telecommunications (CONFTELE 2013)*, Castelo Branco, Portugal, 2013.

VIII. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Coding Tree Depth Estimation for Complexity Reduction of HEVC". *Data Compression Conference (DCC 2013)*, Snowbird, USA, 2013.

IX. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Dynamic Tree-Depth Adjustment for Low Power HEVC Encoders". *IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, Seville, Spain, 2012.

X. **CORREA, G.**; ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Motion Compensated Tree Depth Limitation for Complexity Control of HEVC Encoding". *IEEE International Conference on Image Processing (ICIP 2012)*, Orlando, USA, 2012.

XI. **CORREA, G.;** ASSUNCAO, P. A.; AGOSTINI, L. V.; CRUZ, L. A.. "Adaptive Coding Tree for Complexity Control of High Efficiency Video Encoders". *Picture Coding Symposium (PCS 2012)*, Krakow, Poland, 2012.

# Appendix B

# Common Test Conditions and Video Sequences

This appendix presents in section B.1 the *Common Test Conditions* (CTC) document [43], which describes the experimental setup for most of the tests performed in this work. The CTC document also lists the 24 video sequences used in the experiments and presents some of their characteristics, such as the number of frames, the frame rate and the number of bits used to represent each pixel.

Section B.2 presents the spatial resolution for each video sequence, since this information is not included in the CTC document.

Finally, section B.3 shows one frame belonging to each video sequence listed in this appendix.

## B.1 Common Test Conditions

**Joint Collaborative Team on Video Coding (JCT-VC)**
**of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**
10th Meeting: Stockholm, SE, 11 – 20 July 2012

Document: JCTVC-J1100
WG11 Number: m26383

| | |
|---|---|
| *Title:* | **Common test conditions and software reference configurations** |
| *Status:* | Output document |
| *Purpose:* | Information |
| *Author(s) or Contact(s):* | Frank Bossen |
| | Tel: +1 650 496 4742 |
| | Email: bossen@docomoinnovations.com |
| *Source:* | JCT-VC |

## Abstract

This document defines common test conditions and software reference configurations to be used in the context of core experiments (CE) conducted between the 10th and the 11th JCT-VC meetings. These common test conditions are also recommended for use in technical contributions to the 11th JCT-VC meeting, if applicable.

## 1. Introduction

Common test conditions are desirable to conduct experiments in a well-defined environment and ease the comparison of the outcome of experiments.

This document defines 8 test conditions, reflecting a combination of high efficiency and low complexity, and of intra-only, random-access, and low-delay settings:

- Intra, main
- Intra, high efficiency, 10 bit
- Random access, main
- Random access, high efficiency, 10 bit
- Low delay, main
- Low delay, high efficiency, 10 bit
- Low delay, main, P slices only (optional)
- Low delay, high efficiency, P slices only, 10 bit (optional)

A subset of these test conditions might be used for a particular experiment. For example, when testing an intra coding tool, only intra configurations might be used. Also, when testing variations of a tool such as adaptive loop filtering (ALF), low-complexity configurations might be skipped as this tool is not enabled therein.

Version 8.0 of the common software is expected to be used for most experiments. More recent versions are encouraged where applicable. Availability of the software will be announced on the JCT-VC email reflector by August 7, 2012.

A version 8.1 of the common software will be provided roughly 3 weeks after version 8.0. It will additionally include tools that do not affect common conditions.

The following sections define test sequences, quantization parameter values, encoder configuration files, and compile-time options to be used.

People bringing input contributions should provide a set of results as complete as possible that apply to the proposal. Results should be reported using the attached Excel sheets.

## 2. Test sequences

Table 1 defines the set of test sequences to be used for intra, random-access, and low-delay conditions. All frames (as defined by frame count in the table) shall be encoded for all sequences and test cases described below (see Section 4 for definitions of Main and HE10).

Test sequences are available on ftp://hevc@ftp.tnt.uni-hannover.de/testsequences/ (please contact the JCT-VC chairs for login information).

**Table 1: Test sequences**

| Class | Sequence name | Frame count | Frame rate | Bit depth | Intra | Random access | Low-delay |
|---|---|---|---|---|---|---|---|
| A | Traffic | 150 | 30fps | 8 | Main/HE10 | Main/HE10 | |
| A | PeopleOnStreet | 150 | 30fps | 8 | Main/HE10 | Main/HE10 | |
| A | Nebuta | 300 | 60fps | 10 | Main/HE10 | Main/HE10 | |
| A | SteamLocomotive | 300 | 60fps | 10 | Main/HE10 | Main/HE10 | |
| B | Kimono | 240 | 24fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| B | ParkScene | 240 | 24fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| B | Cactus | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| B | BQTerrace | 600 | 60fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| B | BasketballDrive | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| C | RaceHorses1 | 300 | 30fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| C | BQMall | 600 | 60fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| C | PartyScene | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| C | BasketballDrill | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| D | RaceHorses2 | 300 | 30fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| D | BQSquare | 600 | 60fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| D | BlowingBubbles | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| D | BasketballPass | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| E | FourPeople | 600 | 60fps | 8 | Main/HE10 | | Main/HE10 |
| E | Johnny | 600 | 60fps | 8 | Main/HE10 | | Main/HE10 |
| E | KristenAndSara | 600 | 60fps | 8 | Main/HE10 | | Main/HE10 |
| F | BaskeballDrillText | 500 | 50fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| F | ChinaSpeed | 500 | 30fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| F | SlideEditing | 300 | 30fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |
| F | SlideShow | 500 | 20fps | 8 | Main/HE10 | Main/HE10 | Main/HE10 |

Note: When the encoder operates in 8-bit mode (InternalBitDepth=8) for a 10-bit source, each 10-bit source sample x is converted prior to encoding to an 8-bit value (x+2) / 4 clipped to the [0,255] range. Similarly when the encoder operates in 10-bit mode (InternalBitDepth=10) for an 8-bit source, each 8-bit source sample x is converted prior to encoding to a 10-bit value 4*x. This behavior is built into the reference encoder and no external conversion program is required.

## 3. Quantization parameter values

For each video sequence four quantization parameter values are to be used: 22, 27, 32 and 37. These values define the QP values used for the I-frames in a sequence (configuration files further define QP values used for other frames).

## 4. Configuration files

The following sections define encoder configuration files to be used for each test case. Parameters to be changed for each test point are:

- InputFile to reflect the location of the source video sequence on the test system
- FrameRate to reflect the frame rate of a given sequence as per Table 1
- SourceWidth to reflect the width of the source video sequence
- SourceHeight to reflect the height of the source video sequence
- FramesToBeEncoded to reflect the frame count of a given sequence as per Table 1
- IntraPeriod to reflect the intra refresh period in the random access test cases. The intra refresh period is dependent on the frame rate of the source: a value 16 shall be used for sequences with a frame rate equal to 20fps, 24 for 24fps, 32 for 30fps, 48 for 50fps, and 64 for 60fps.
- QP to reflect the quantization parameter values defined in section 3.
- InputBitDepth to reflect the bit depth of a given sequence as per Table 1

These configuration files are provided in the cfg/ folder of version 8.0 of the common software package (available at https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-8.0). There are 8 configurations provided as follows:

- "All Intra – Main" (AI-Main): encoder_intra_main.cfg
- "Random access – Main" (RA-Main): encoder_randomaccess_main.cfg
- "Low-delay B – Main" (LB-Main): encoder_lowdelay_main.cfg
- "Low-delay P – Main" (LP-Main, optional): encoder_lowdelay_P_main.cfg
- "All Intra – High efficiency" (AI-HE10): encoder_intra_he10.cfg
- "Random access – High efficiency" (RA-HE10): encoder_randomaccess_he10.cfg
- "Low-delay B – High efficiency" (LB-HE10): encoder_lowdelay_he10.cfg
- "Low-delay P – High efficiency" (LP-HE10, optional): encoder_lowdelay_P_he10.cfg

Sequence-specific parameters are to be found in the cfg/per-sequence/ folder.

## 5. Compile-time settings

Compile-time settings are defined mostly in the TypeDef.h file located in the source/Lib/TLibCommon folder of the common software. The default settings provided in the source code should be used.

## B.2 Spatial Resolutions

As the CTC document presented in section B.1 does not include the spatial resolution of each video sequence, this information is shown in Table B.1 of this section. Besides the 24 video sequences listed in the CTC, one supplementary video sequence (*Tennis*) was used in the experiments in order to allow tests with high-resolution sequences not used in the training of the decision trees and in the parameter selection presented in section 7.1. The *Tennis* sequence is composed of 240 frames, presents a rate of 24 frames per second and a bit depth of 8 bits per pixel.

Table B.1: Video sequence spatial resolutions.

| **Name** | **Spatial Resolution** |
|---|---|
| *BaskeballDrillText* | 832×480 |
| *BasketballDrill* | 832×480 |
| *BasketballDrive* | 1920×1080 |
| *BasketballPass* | 416×240 |
| *BlowingBubbles* | 416×240 |
| *BQMall* | 832×480 |
| *BQSquare* | 416×240 |
| *BQTerrace* | 1920×1080 |
| *Cactus* | 1920×1080 |
| *ChinaSpeed* | 1024×768 |
| *FourPeople* | 1280×720 |
| *Johnny* | 1280×720 |
| *Kimono* | 1920×1080 |
| *KristenAndSara* | 1280×720 |
| *NebutaFestival* | 2560×1600 |
| *ParkScene* | 1920×1080 |
| *PartyScene* | 832×480 |
| *PeopleOnStreet* | 2560×1600 |
| *RaceHorses1* | 416×240 |
| *RaceHorses2* | 832×480 |
| *SlideEditing* | 1280×720 |
| *SlideShow* | 1280×720 |
| *SteamLocomotive* | 2560×1600 |
| *Tennis* | 1920×1080 |
| *Traffic* | 2560×1600 |

## B.3 Video Sequences

Trying to illustrate the characteristics of the 25 video sequences listed in Table B.1, the frame positioned exactly in the middle of each one is presented in this section. Fig. B.1 to Fig. B.25 show each middle frame, which are all pictured here in the same size (i.e., they are not resized in the same proportions).

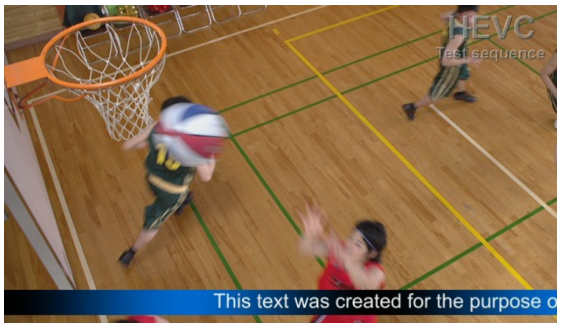


Fig. B.1: *BaskeballDrillText*.

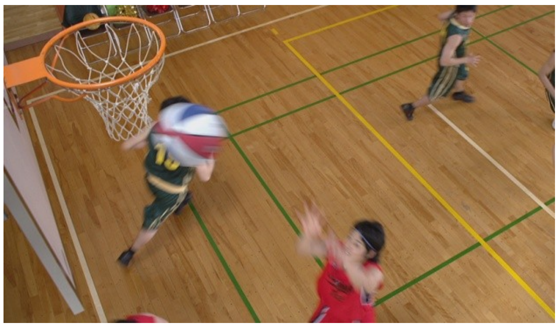


Fig. B.2: *BaskeballDrill*.

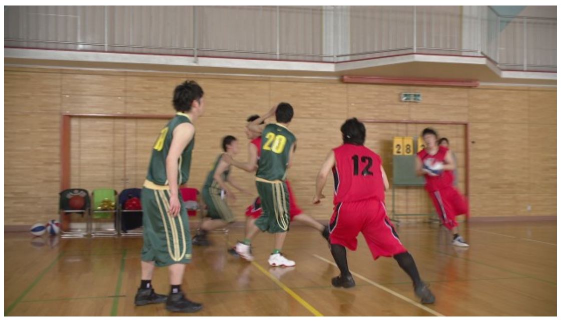


Fig. B.3: *BasketballDrive*.

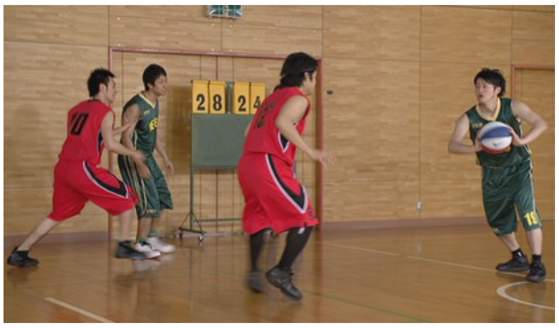


Fig. B.4: *BasketballPass*.

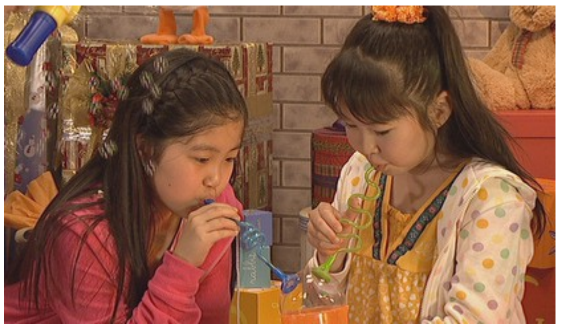


Fig. B.5: *BlowingBubbles*.

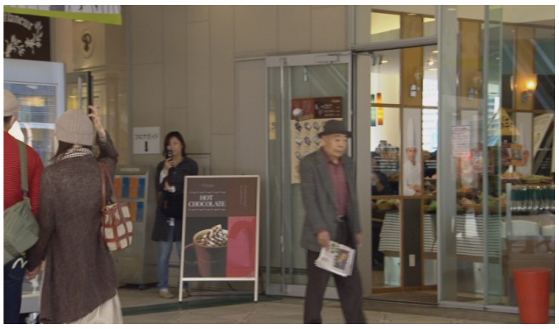


Fig. B.6: *BQMall*.

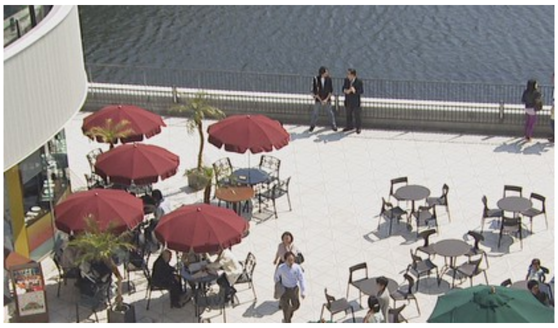


Fig. B.7: *BQSquare*.

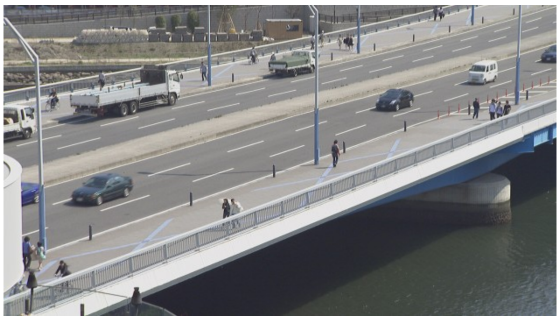


Fig. B.8: *BQTerrace*.

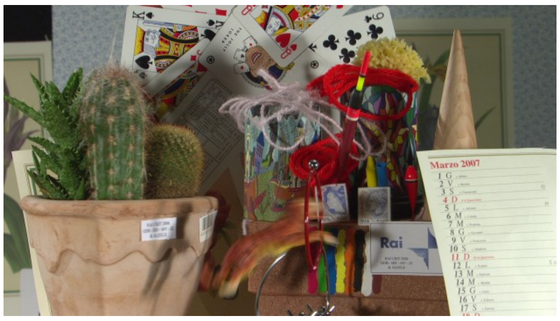


Fig. B.9: *Cactus*.

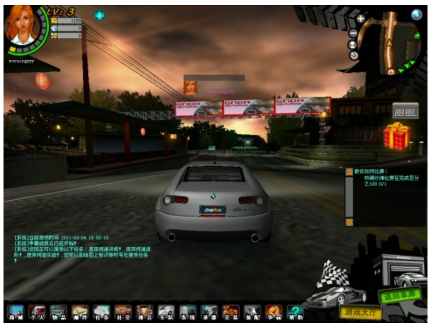
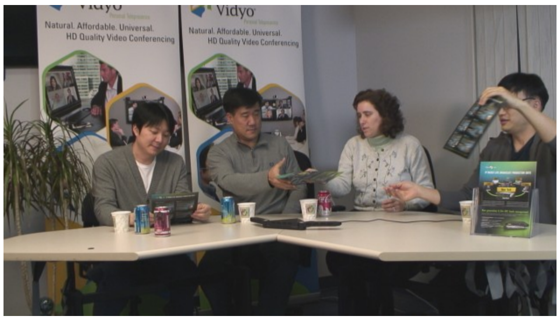


Fig. B.10: *ChinaSpeed*.


Vidyo
Natural. Affordable. Universal.
HD Quality Video Conferencing
Fig. B.11: *FourPeople*.

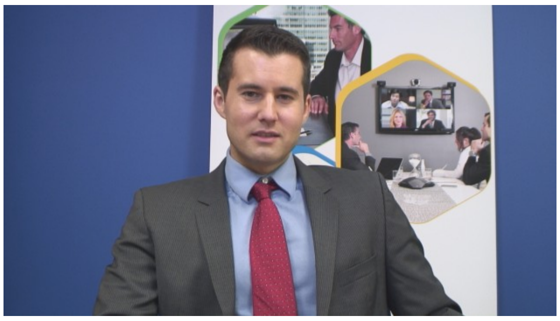


Fig. B.12: *Johnny*.

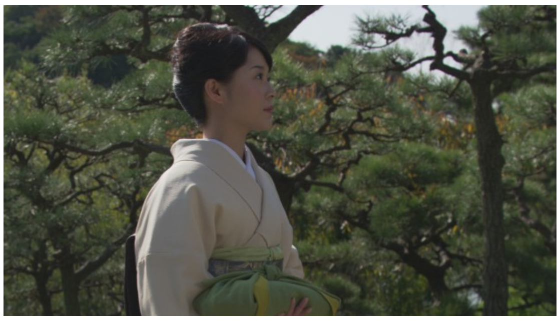
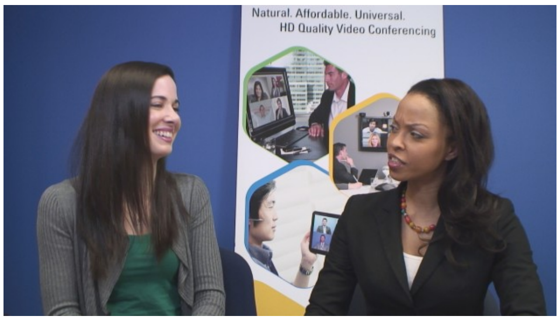


Fig. B.13: *Kimono*.


Natural. Affordable. Universal.
HD Quality Video Conferencing
Fig. B.14: *KristenAndSara*.

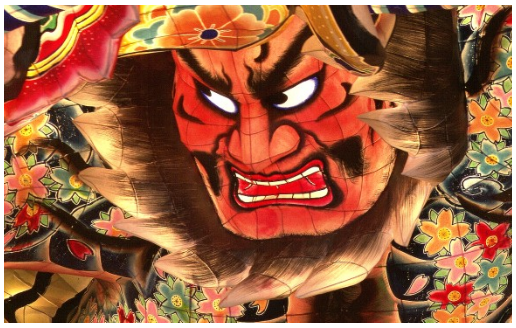


Fig. B.15: *NebutaFestival*.

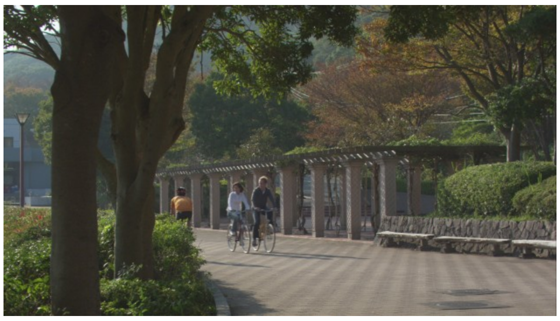


Fig. B.16: *ParkScene*.

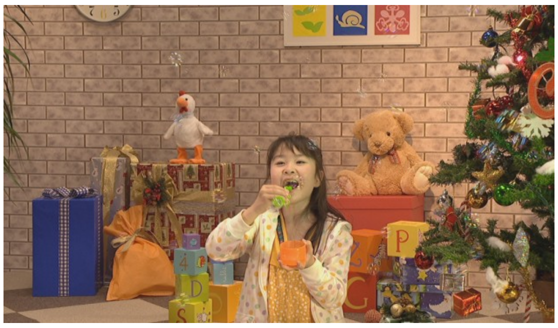


Fig. B.17: *PartyScene*.

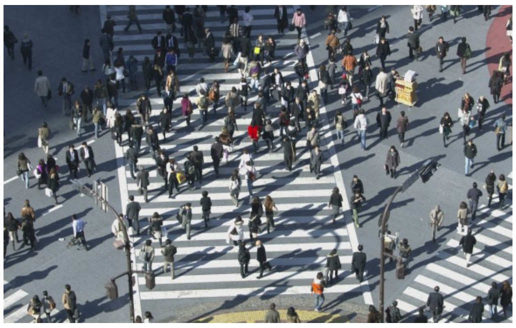


Fig. B.18: *PeopleOnStreet*.

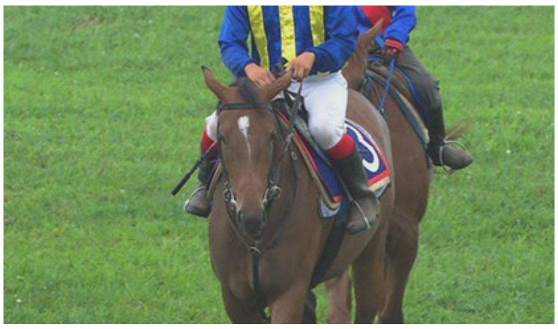


Fig. B.19: *RaceHorses1*.

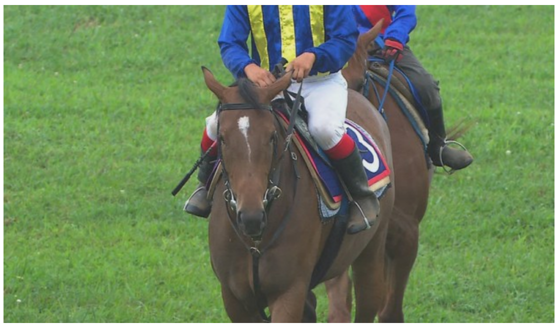
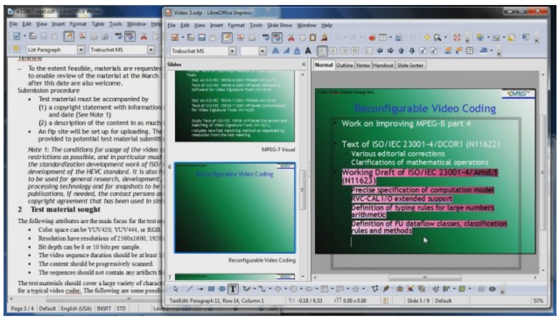
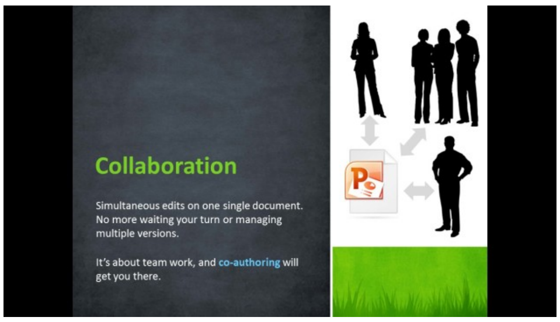


Fig. B.20: *RaceHorses2*.


Reconfigurable Video Coding

- Work on Improving MPEG-B part 4
- Text of ISO/IEC 23001-4/DCOR1 (N11622)
  - Various editorial corrections
  - Clarifications of mathematical operations
- Working Draft of ISO/IEC 23001-4/Amd.1 (N11623)
  - Precise specification of computation model
  - RVC-CAL I/O extended support
  - Definition of typing rules for large numbers arithmetic
  - Definition of FU dataflow classes, classification rules and methods

Fig. B.21: *SlideEditing*.


Collaboration

Simultaneous edits on one single document. No more waiting your turn or managing multiple versions.

It's about team work, and co-authoring will get you there.

Fig. B.22: *SlideShow*.

Fig. B.23: *SteamLocomotive*.

Fig. B.24: *Tennis*.

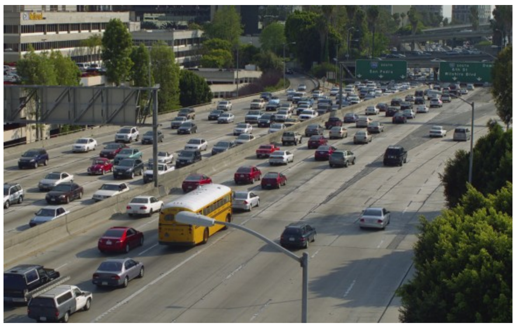


Fig. B.25: *Traffic*.

# Appendix C

# Obtained Decision Trees

This appendix presents the decision trees obtained with the methodology described in chapter 6. The graphic representation of each tree, obtained with the WEKA tool [127], is presented in sections C.1, C.2 and C.3 for the Coding Tree early termination, the PU early termination and the RQT early termination, respectively.

## C.1 Decision Trees for Coding Tree Early Termination

As explained in section 6.3, three decision trees were trained and implemented for the Coding Tree early termination, one for each CU size that allows splitting into smaller CUs (i.e., 16×16, 32×32 and 64×64). The three trees are presented from Fig. C.1 to Fig. C.3, where *C* and *T* correspond to the decisions of continuing and terminating the CU splitting process, respectively.

## C.2 Decision Trees for PU Early Termination

The four decision trees introduced in section 6.4 for the PU early termination are presented from Fig. C.4 to Fig. C.7, one for each CU size possible (i.e., 8×8, 16×16, 32×32 and 64×64). In the figures, *C* and *T* correspond to the decisions of continuing and terminating the process of choosing the best PU splitting mode, respectively.

## C.3 Decision Trees for RQT Early Termination

Two decision trees for the RQT early termination were trained and implemented, as explained in section 6.5. Fig. C.8 presents the decision tree obtained for 16×16 TUs and Fig. C.9 shows the decision tree for 32×32 TUs. In both figures, *C* and *T* correspond to the decisions of continuing and terminating the TU splitting process, respectively.

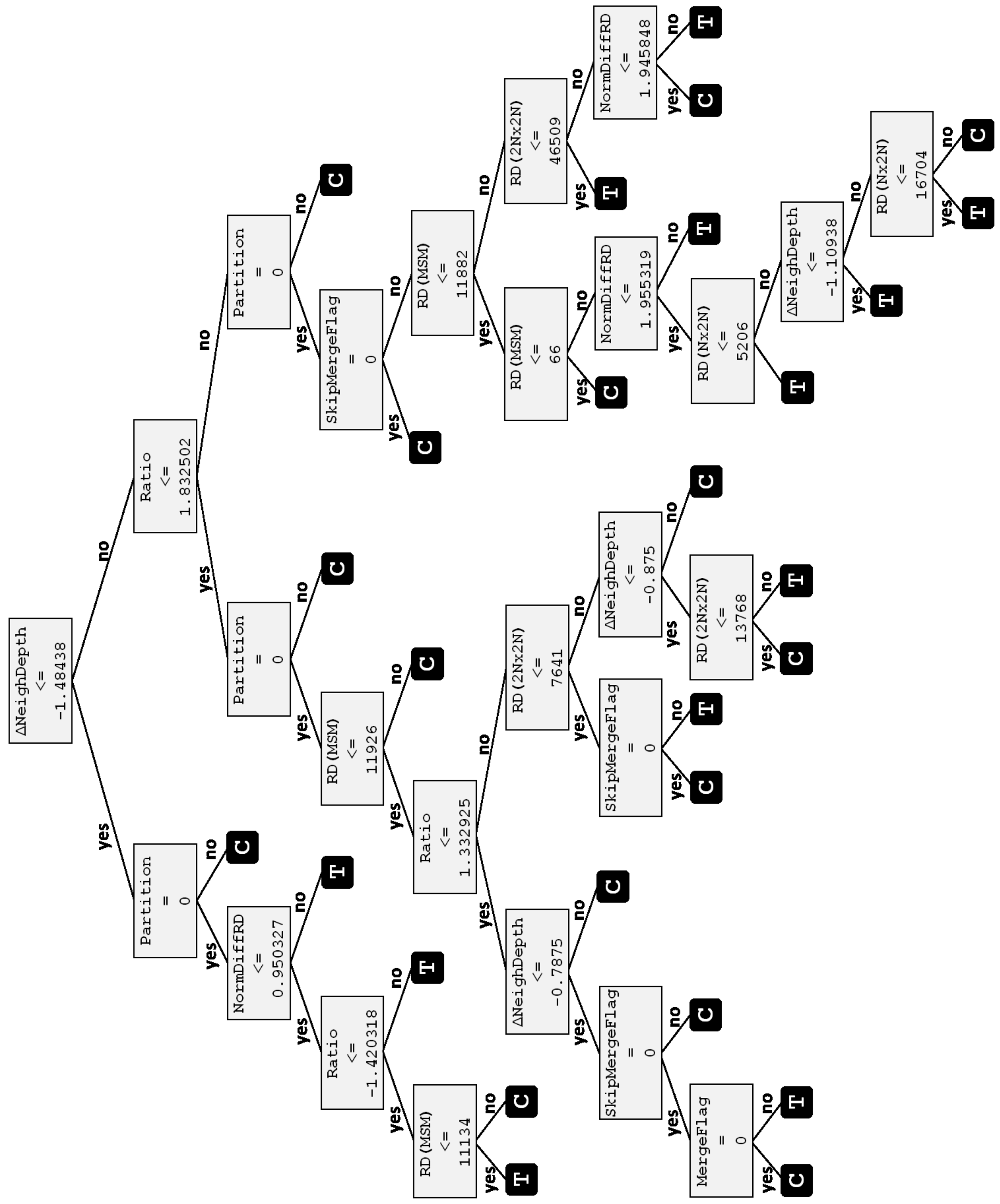


Fig. C.1: Coding Tree early termination decision tree for 16×16 CUs.

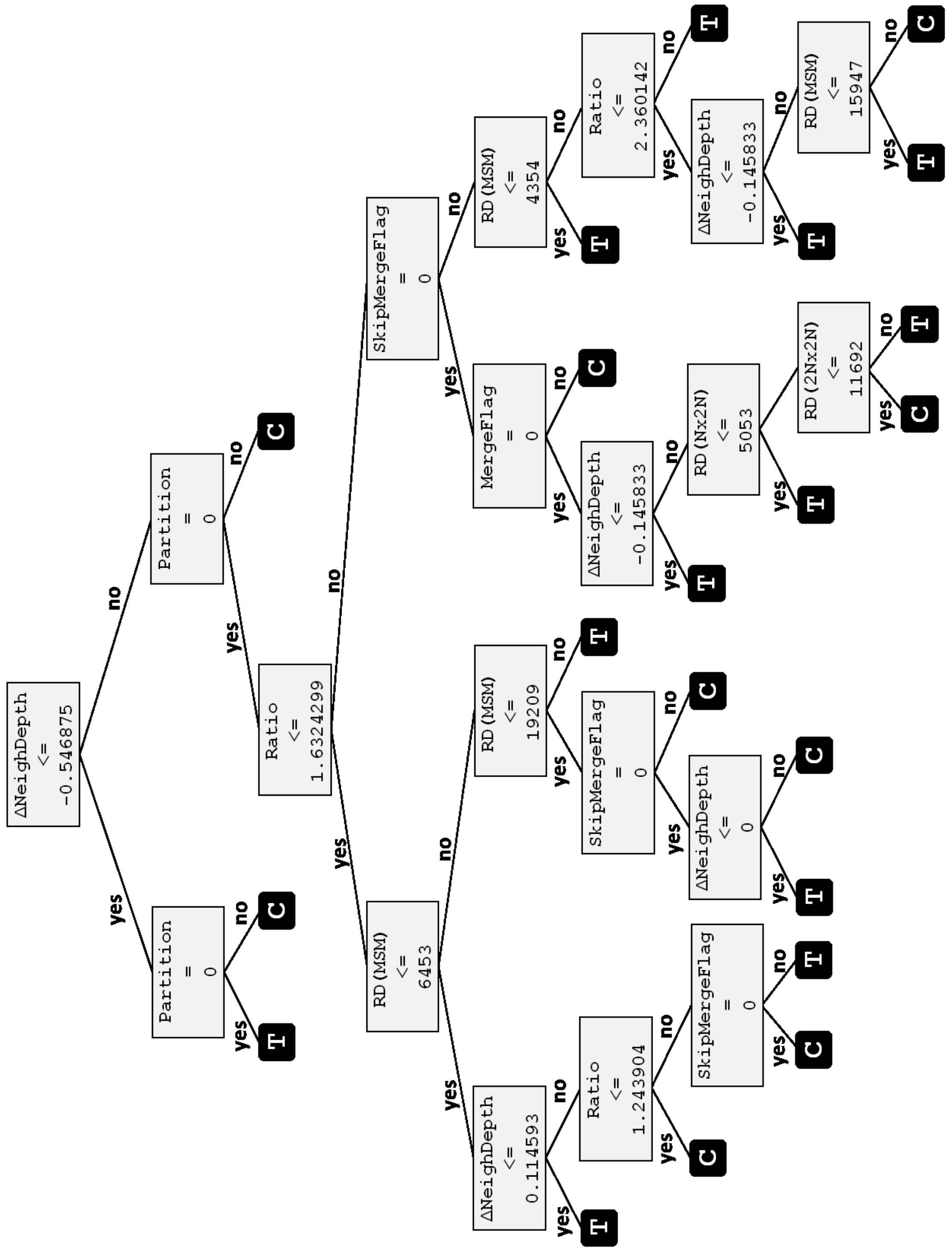


Fig. C.2: Coding Tree early termination decision tree for 32×32 CUs.

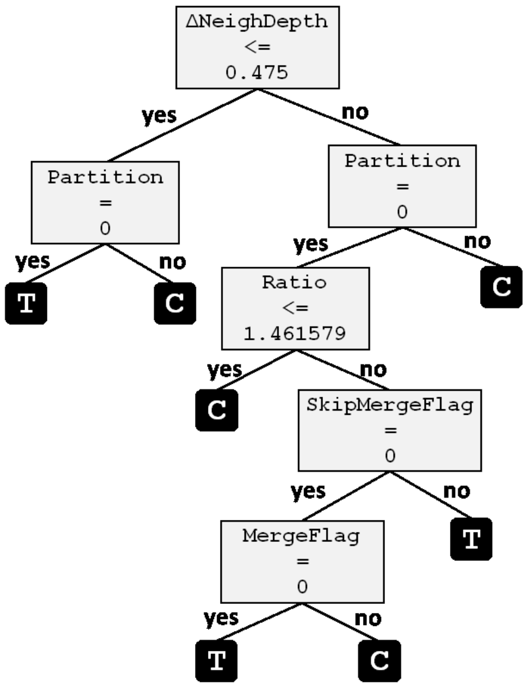
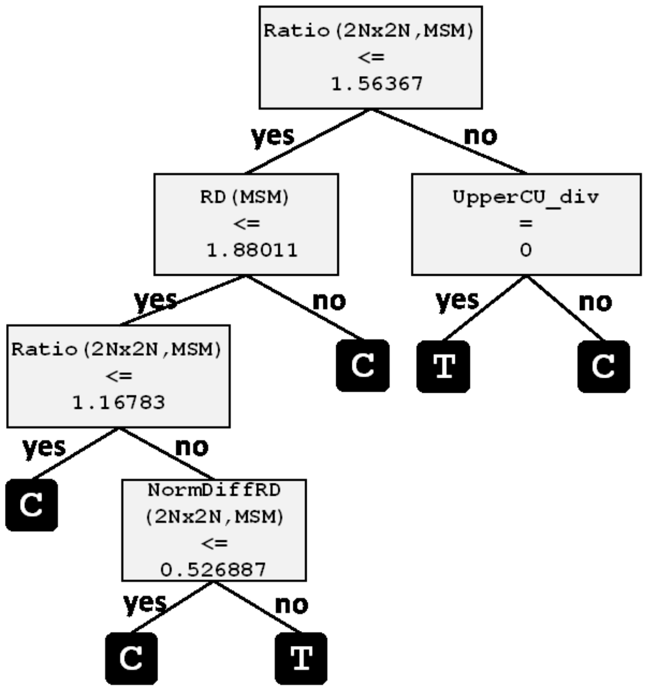


Fig. C.3: Coding Tree early termination decision tree for 64×64 CUs.



Ratio(2Nx2N,MSM) <= 1.56367
yes
no
RD(MSM) <= 1.88011
UpperCU_div = 0
yes
no
yes
no
Ratio(2Nx2N,MSM) <= 1.16783
C
T
C
yes
no
C
NormDiffRD (2Nx2N,MSM) <= 0.526887
yes
no
C
T

Fig. C.4: PU early termination decision tree for 8×8 CUs.

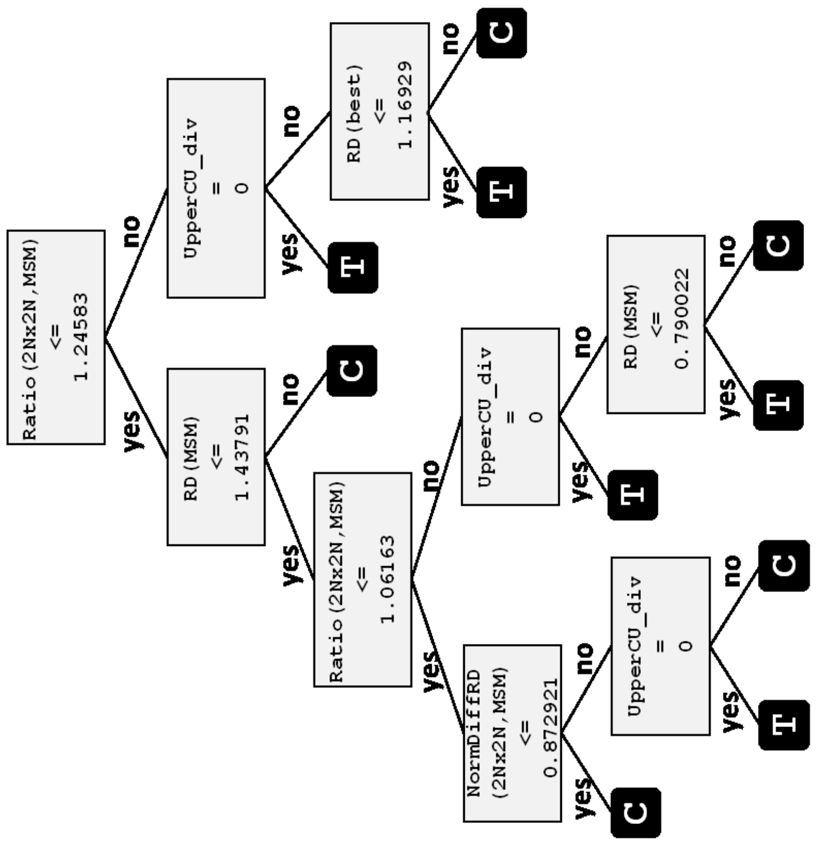


Fig. C.5: PU early termination decision tree for 16×16 CUs.

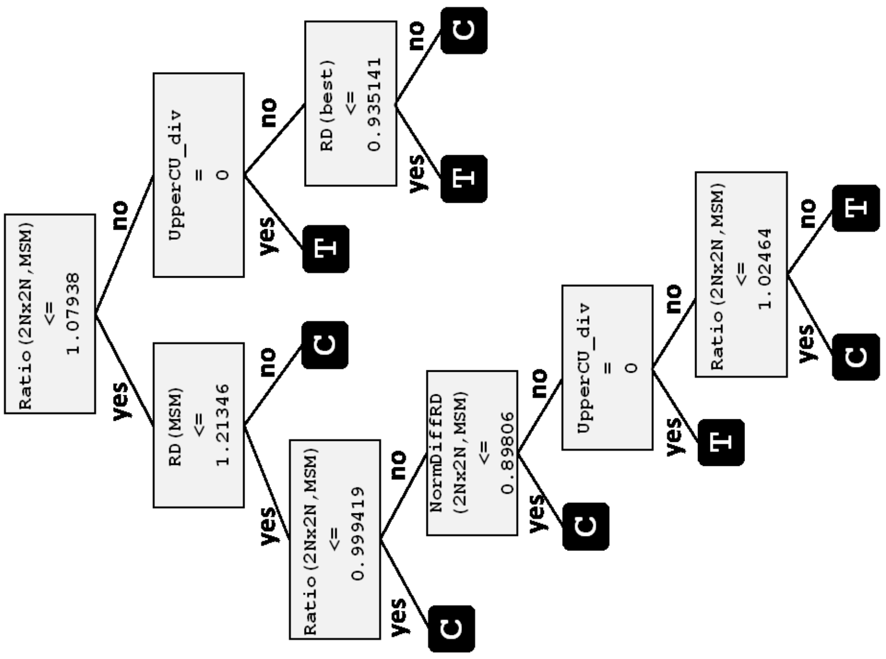


Fig. C.6: PU early termination decision tree for 32×32 CUs.

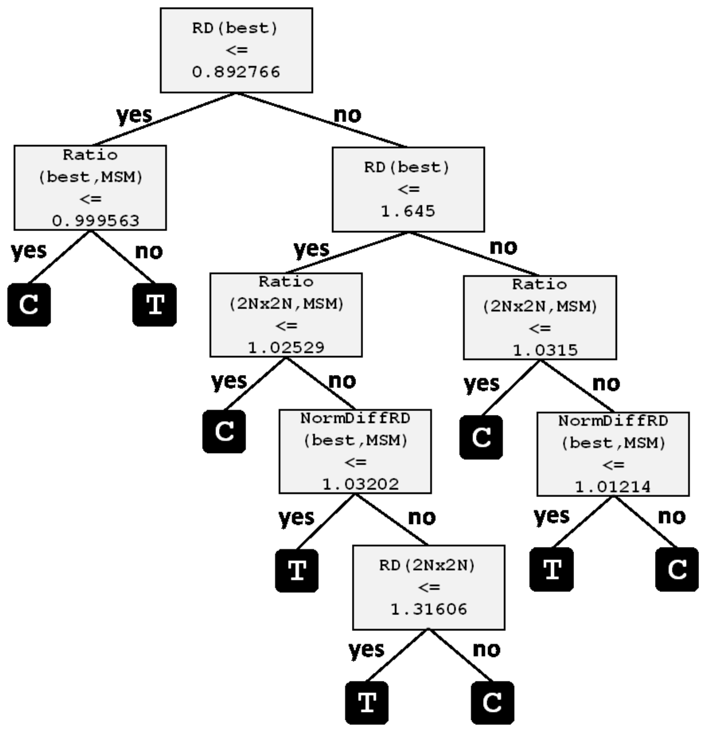

RD(best) <= 0.892766
yes
no
Ratio (best,MSM) <= 0.999563
RD(best) <= 1.645
C
T
Ratio (2Nx2N,MSM) <= 1.02529
Ratio (2Nx2N,MSM) <= 1.0315
NormDiffRD (best,MSM) <= 1.03202
NormDiffRD (best,MSM) <= 1.01214
RD(2Nx2N) <= 1.31606

Fig. C.7: PU early termination decision tree for 64×64 CUs.

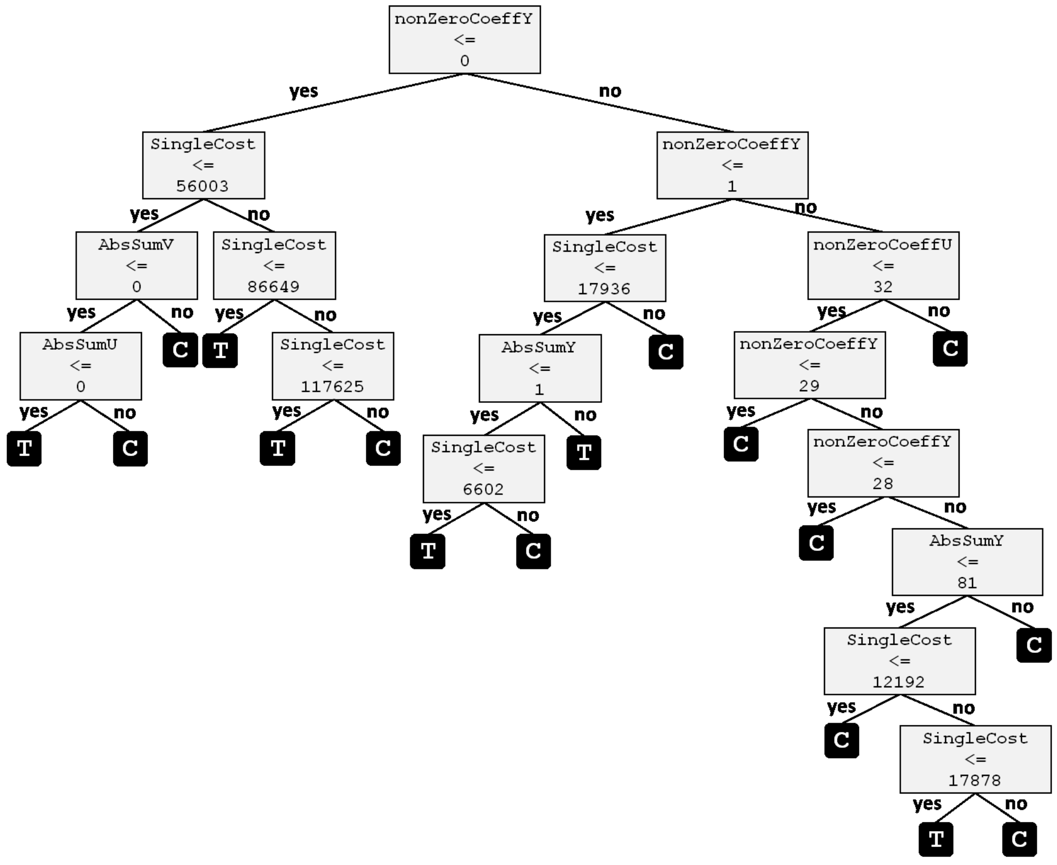


Fig. C.8: RQT early termination decision tree for 16×16 TUs.

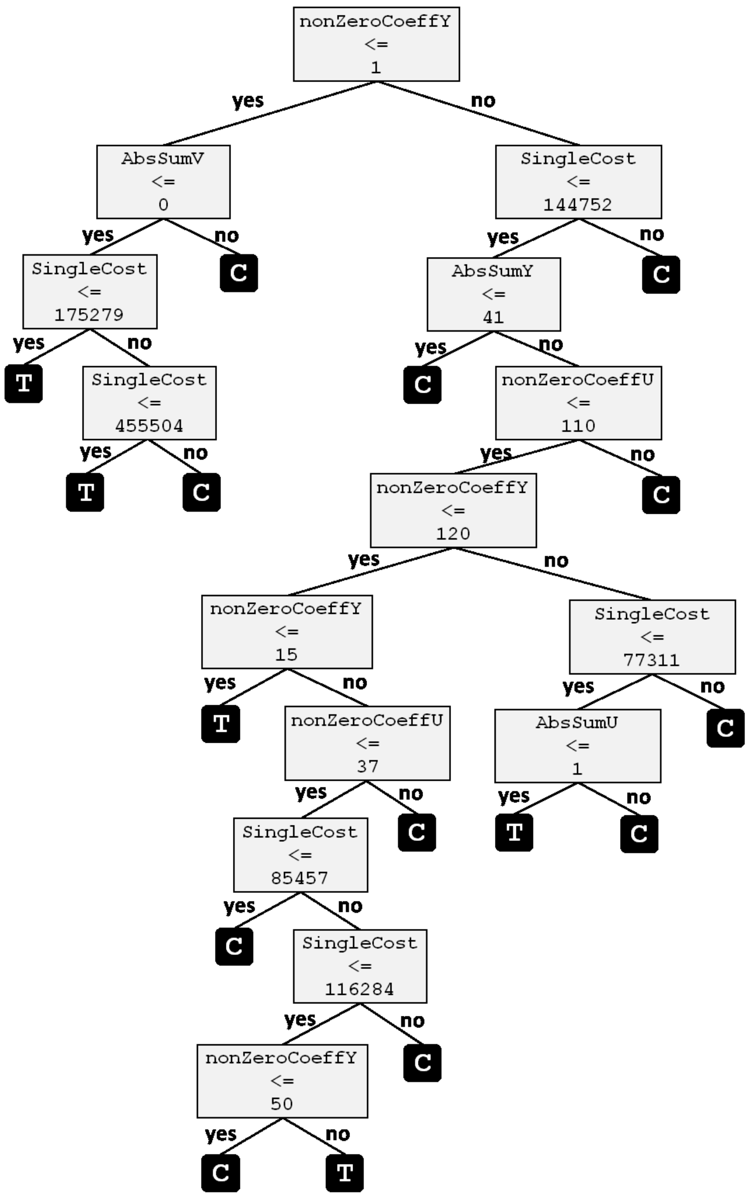


Fig. C.9: RQT early termination decision tree for 32×32 TUs.

# Appendix D

# Encoder Configurations Tested in the R-D-C Analysis

In section 7.1 it was explained that the encoding configurations considered in the R-D-C analysis were created by modifying the value of each parameter, one at a time, so that every parameter value could be tested with all possible values of the remaining ones, totalising 240 encoding configurations. As described in that section, the R-D efficiency and the computational complexity associated to each configuration was assessed with four high-resolution video sequences, QPs 22, 27, 32, 37, and the *Random Access* temporal configuration, totalising 3,840 encodings. Average BD-rate, BD-PSNR and computational complexity reduction for each configuration, using the unmodified encoder as reference (configuration 1), were calculated. Each configuration tested and their respective results are presented in this appendix, in Table D.1, since only those corresponding to the points that compose the Pareto frontier were presented in Table 7.2 of chapter 7.

Table D.2 shows the look-up table (LUT) used to determine the encoding configuration that best suits a given $R_{T(i-1)}$ ratio between the target time ($T_T$) and the weighted average encoding time ($T_{W(i-1)}$) of the last two GOPs, as explained in section 7.2.1 of chapter 7 (see (Eq. 55)). The encoding configuration $p_{(i-1)}$ used in the previous GOP is used to select a line in the LUT where the closest value to $R_{T(i-1)}$ is searched. Once it is found, the index indicated by the column where the found value belongs is chosen as the new encoding configuration $p_{(i)}$ to be used in the current GOP *i*.

Table D.1: Parameters, computational complexity, BD-PSNR and BD-rate for the 240 encoder configurations considered in the R-D-C analysis.

| **Config.** | ***SR*** | ***BPR*** | ***HME*** | ***CTET*** | ***PUET*** | ***RQTET*** | **Normal. Complex.** | **BD-PSNR (dB)** | **BD-rate (%)** |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 64 | 4 | on | off | off | off | 1.000 | 0.000 | 0.000 |
| 2 | 32 | 4 | on | off | off | off | 0.969 | 0.000 | 0.016 |
| 3 | 16 | 4 | on | off | off | off | 0.958 | -0.014 | 0.426 |
| 4 | 8 | 4 | on | off | off | off | 0.950 | -0.043 | 1.276 |
| 5 | 4 | 4 | on | off | off | off | 0.945 | -0.082 | 2.469 |
| 6 | 64 | 2 | on | off | off | off | 0.973 | -0.001 | 0.038 |
| 7 | 32 | 2 | on | off | off | off | 0.941 | -0.004 | 0.150 |
| 8 | 16 | 2 | on | off | off | off | 0.930 | -0.015 | 0.426 |
| 9 | 8 | 2 | on | off | off | off | 0.923 | -0.049 | 1.498 |
| 10 | 4 | 2 | on | off | off | off | 0.917 | -0.096 | 2.845 |
| 11 | 64 | 1 | on | off | off | off | 0.964 | -0.002 | 0.089 |
| 12 | 32 | 1 | on | off | off | off | 0.933 | -0.006 | 0.221 |
| 13 | 16 | 1 | on | off | off | off | 0.922 | -0.019 | 0.588 |
| 14 | 8 | 1 | on | off | off | off | 0.915 | -0.048 | 1.481 |
| 15 | 4 | 1 | on | off | off | off | 0.910 | -0.104 | 3.093 |
| 16 | 64 | 4 | off | off | off | off | 0.899 | -0.018 | 0.595 |
| 17 | 32 | 4 | off | off | off | off | 0.868 | -0.019 | 0.637 |
| 18 | 16 | 4 | off | off | off | off | 0.857 | -0.029 | 0.886 |
| 19 | 8 | 4 | off | off | off | off | 0.849 | -0.056 | 1.727 |
| 20 | 4 | 4 | off | off | off | off | 0.845 | -0.100 | 3.045 |
| 21 | 64 | 2 | off | off | off | off | 0.872 | -0.019 | 0.648 |
| 22 | 32 | 2 | off | off | off | off | 0.841 | -0.019 | 0.651 |
| 23 | 16 | 2 | off | off | off | off | 0.829 | -0.031 | 0.990 |
| 24 | 8 | 2 | off | off | off | off | 0.822 | -0.059 | 1.839 |
| 25 | 4 | 2 | off | off | off | off | 0.817 | -0.112 | 3.386 |
| 26 | 64 | 1 | off | off | off | off | 0.863 | -0.020 | 0.697 |
| 27 | 32 | 1 | off | off | off | off | 0.832 | -0.021 | 0.735 |
| 28 | 16 | 1 | off | off | off | off | 0.821 | -0.033 | 1.067 |
| 29 | 8 | 1 | off | off | off | off | 0.814 | -0.062 | 1.956 |
| 30 | 4 | 1 | off | off | off | off | 0.809 | -0.120 | 3.690 |
| 31 | 64 | 4 | on | on | off | off | 0.723 | -0.002 | 0.086 |
| 32 | 32 | 4 | on | on | off | off | 0.698 | -0.003 | 0.124 |
| 33 | 16 | 4 | on | on | off | off | 0.688 | -0.017 | 0.527 |
| 34 | 8 | 4 | on | on | off | off | 0.683 | -0.045 | 1.337 |
| 35 | 4 | 4 | on | on | off | off | 0.680 | -0.085 | 2.495 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 36 | 64 | 2 | on | on | off | off | 0.706 | -0.004 | 0.145 |
| 37 | 32 | 2 | on | on | off | off | 0.680 | -0.007 | 0.239 |
| 38 | 16 | 2 | on | on | off | off | 0.669 | -0.017 | 0.550 |
| 39 | 8 | 2 | on | on | off | off | 0.664 | -0.047 | 1.457 |
| 40 | 4 | 2 | on | on | off | off | 0.662 | -0.102 | 3.051 |
| 41 | 64 | 1 | on | on | off | off | 0.700 | -0.007 | 0.255 |
| 42 | 32 | 1 | on | on | off | off | 0.673 | -0.006 | 0.256 |
| 43 | 16 | 1 | on | on | off | off | 0.666 | -0.021 | 0.692 |
| 44 | 8 | 1 | on | on | off | off | 0.659 | -0.052 | 1.633 |
| 45 | 4 | 1 | on | on | off | off | 0.657 | -0.108 | 3.211 |
| 46 | 64 | 4 | off | on | off | off | 0.660 | -0.020 | 0.678 |
| 47 | 32 | 4 | off | on | off | off | 0.634 | -0.019 | 0.674 |
| 48 | 16 | 4 | off | on | off | off | 0.625 | -0.032 | 1.018 |
| 49 | 8 | 4 | off | on | off | off | 0.618 | -0.059 | 1.846 |
| 50 | 4 | 4 | off | on | off | off | 0.616 | -0.104 | 3.175 |
| 51 | 64 | 2 | off | on | off | off | 0.643 | -0.021 | 0.730 |
| 52 | 32 | 2 | off | on | off | off | 0.616 | -0.020 | 0.715 |
| 53 | 16 | 2 | off | on | off | off | 0.607 | -0.034 | 1.113 |
| 54 | 8 | 2 | off | on | off | off | 0.601 | -0.063 | 1.979 |
| 55 | 4 | 2 | off | on | off | off | 0.598 | -0.112 | 3.392 |
| 56 | 64 | 1 | off | on | off | off | 0.637 | -0.022 | 0.761 |
| 57 | 32 | 1 | off | on | off | off | 0.610 | -0.022 | 0.796 |
| 58 | 16 | 1 | off | on | off | off | 0.601 | -0.035 | 1.159 |
| 59 | 8 | 1 | off | on | off | off | 0.596 | -0.065 | 2.062 |
| 60 | 4 | 1 | off | on | off | off | 0.594 | -0.122 | 3.833 |
| 61 | 64 | 4 | on | off | on | off | 0.586 | -0.018 | 0.572 |
| 62 | 32 | 4 | on | off | on | off | 0.569 | -0.019 | 0.603 |
| 63 | 16 | 4 | on | off | on | off | 0.563 | -0.029 | 0.847 |
| 64 | 8 | 4 | on | off | on | off | 0.559 | -0.059 | 1.782 |
| 65 | 4 | 4 | on | off | on | off | 0.558 | -0.099 | 2.946 |
| 66 | 64 | 2 | on | off | on | off | 0.572 | -0.019 | 0.625 |
| 67 | 32 | 2 | on | off | on | off | 0.554 | -0.019 | 0.620 |
| 68 | 16 | 2 | on | off | on | off | 0.548 | -0.030 | 0.923 |
| 69 | 8 | 2 | on | off | on | off | 0.545 | -0.061 | 1.916 |
| 70 | 4 | 2 | on | off | on | off | 0.545 | -0.115 | 3.470 |
| 71 | 64 | 1 | on | off | on | off | 0.567 | -0.019 | 0.611 |
| 72 | 32 | 1 | on | off | on | off | 0.551 | -0.018 | 0.600 |
| 73 | 16 | 1 | on | off | on | off | 0.544 | -0.034 | 1.044 |
| 74 | 8 | 1 | on | off | on | off | 0.541 | -0.065 | 1.985 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 75 | 4 | 1 | on | off | on | off | 0.540 | -0.118 | 3.534 |
| 76 | 64 | 4 | off | off | on | off | 0.538 | -0.032 | 1.001 |
| 77 | 32 | 4 | off | off | on | off | 0.521 | -0.033 | 1.074 |
| 78 | 16 | 4 | off | off | on | off | 0.514 | -0.044 | 1.392 |
| 79 | 8 | 4 | off | off | on | off | 0.511 | -0.071 | 2.205 |
| 80 | 4 | 4 | off | off | on | off | 0.511 | -0.115 | 3.484 |
| 81 | 64 | 2 | off | off | on | off | 0.525 | -0.035 | 1.122 |
| 82 | 32 | 2 | off | off | on | off | 0.507 | -0.034 | 1.094 |
| 83 | 16 | 2 | off | off | on | off | 0.501 | -0.048 | 1.545 |
| 84 | 8 | 2 | off | off | on | off | 0.497 | -0.075 | 2.322 |
| 85 | 4 | 2 | off | off | on | off | 0.496 | -0.124 | 3.757 |
| 86 | 64 | 1 | off | off | on | off | 0.520 | -0.035 | 1.171 |
| 87 | 32 | 1 | off | off | on | off | 0.503 | -0.037 | 1.242 |
| 88 | 16 | 1 | off | off | on | off | 0.497 | -0.047 | 1.507 |
| 89 | 8 | 1 | off | off | on | off | 0.493 | -0.077 | 2.446 |
| 90 | 4 | 1 | off | off | on | off | 0.493 | -0.134 | 4.179 |
| 91 | 64 | 4 | on | on | on | off | 0.481 | -0.026 | 0.823 |
| 92 | 32 | 4 | on | on | on | off | 0.465 | -0.027 | 0.865 |
| 93 | 16 | 4 | on | on | on | off | 0.460 | -0.039 | 1.186 |
| 94 | 8 | 4 | on | on | on | off | 0.456 | -0.066 | 2.013 |
| 95 | 4 | 4 | on | on | on | off | 0.455 | -0.108 | 3.305 |
| 96 | 64 | 2 | on | on | on | off | 0.471 | -0.026 | 0.832 |
| 97 | 32 | 2 | on | on | on | off | 0.455 | -0.027 | 0.879 |
| 98 | 16 | 2 | on | on | on | off | 0.449 | -0.042 | 1.291 |
| 99 | 8 | 2 | on | on | on | off | 0.447 | -0.071 | 2.219 |
| 100 | 4 | 2 | on | on | on | off | 0.446 | -0.122 | 3.708 |
| 101 | 64 | 1 | on | on | on | off | 0.468 | -0.029 | 0.958 |
| 102 | 32 | 1 | on | on | on | off | 0.451 | -0.029 | 0.939 |
| 103 | 16 | 1 | on | on | on | off | 0.447 | -0.041 | 1.294 |
| 104 | 8 | 1 | on | on | on | off | 0.443 | -0.073 | 2.305 |
| 105 | 4 | 1 | on | on | on | off | 0.443 | -0.128 | 3.910 |
| 106 | 64 | 4 | off | on | on | off | 0.445 | -0.041 | 1.330 |
| 107 | 32 | 4 | off | on | on | off | 0.428 | -0.041 | 1.339 |
| 108 | 16 | 4 | off | on | on | off | 0.422 | -0.054 | 1.698 |
| 109 | 8 | 4 | off | on | on | off | 0.419 | -0.081 | 2.553 |
| 110 | 4 | 4 | off | on | on | off | 0.419 | -0.123 | 3.746 |
| 111 | 64 | 2 | off | on | on | off | 0.434 | -0.043 | 1.408 |
| 112 | 32 | 2 | off | on | on | off | 0.419 | -0.044 | 1.422 |
| 113 | 16 | 2 | off | on | on | off | 0.412 | -0.056 | 1.821 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 114 | 8 | 2 | off | on | on | off | 0.409 | -0.087 | 2.700 |
| 115 | 4 | 2 | off | on | on | off | 0.409 | -0.136 | 4.217 |
| 116 | 64 | 1 | off | on | on | off | 0.432 | -0.045 | 1.460 |
| 117 | 32 | 1 | off | on | on | off | 0.415 | -0.044 | 1.443 |
| 118 | 16 | 1 | off | on | on | off | 0.409 | -0.057 | 1.828 |
| 119 | 8 | 1 | off | on | on | off | 0.405 | -0.088 | 2.795 |
| 120 | 4 | 1 | off | on | on | off | 0.405 | -0.142 | 4.460 |
| 121 | 64 | 4 | on | off | off | on | 0.925 | -0.006 | 0.187 |
| 122 | 32 | 4 | on | off | off | on | 0.894 | -0.008 | 0.275 |
| 123 | 16 | 4 | on | off | off | on | 0.884 | -0.018 | 0.527 |
| 124 | 8 | 4 | on | off | off | on | 0.877 | -0.045 | 1.307 |
| 125 | 4 | 4 | on | off | off | on | 0.872 | -0.087 | 2.592 |
| 126 | 64 | 2 | on | off | off | on | 0.897 | -0.008 | 0.252 |
| 127 | 32 | 2 | on | off | off | on | 0.867 | -0.009 | 0.312 |
| 128 | 16 | 2 | on | off | off | on | 0.856 | -0.021 | 0.649 |
| 129 | 8 | 2 | on | off | off | on | 0.849 | -0.050 | 1.514 |
| 130 | 4 | 2 | on | off | off | on | 0.844 | -0.101 | 3.052 |
| 131 | 64 | 1 | on | off | off | on | 0.890 | -0.008 | 0.291 |
| 132 | 32 | 1 | on | off | off | on | 0.859 | -0.009 | 0.350 |
| 133 | 16 | 1 | on | off | off | on | 0.848 | -0.022 | 0.690 |
| 134 | 8 | 1 | on | off | off | on | 0.841 | -0.054 | 1.650 |
| 135 | 4 | 1 | on | off | off | on | 0.837 | -0.109 | 3.331 |
| 136 | 64 | 4 | off | off | off | on | 0.824 | -0.022 | 0.722 |
| 137 | 32 | 4 | off | off | off | on | 0.793 | -0.023 | 0.770 |
| 138 | 16 | 4 | off | off | off | on | 0.782 | -0.037 | 1.198 |
| 139 | 8 | 4 | off | off | off | on | 0.775 | -0.064 | 1.972 |
| 140 | 4 | 4 | off | off | off | on | 0.771 | -0.107 | 3.286 |
| 141 | 64 | 2 | off | off | off | on | 0.797 | -0.023 | 0.762 |
| 142 | 32 | 2 | off | off | off | on | 0.766 | -0.024 | 0.796 |
| 143 | 16 | 2 | off | off | off | on | 0.754 | -0.037 | 1.250 |
| 144 | 8 | 2 | off | off | off | on | 0.748 | -0.067 | 2.131 |
| 145 | 4 | 2 | off | off | off | on | 0.744 | -0.118 | 3.669 |
| 146 | 64 | 1 | off | off | off | on | 0.788 | -0.025 | 0.835 |
| 147 | 32 | 1 | off | off | off | on | 0.758 | -0.026 | 0.898 |
| 148 | 16 | 1 | off | off | off | on | 0.746 | -0.040 | 1.340 |
| 149 | 8 | 1 | off | off | off | on | 0.740 | -0.069 | 2.215 |
| 150 | 4 | 1 | off | off | off | on | 0.736 | -0.123 | 3.787 |
| 151 | 64 | 4 | on | on | off | on | 0.674 | -0.009 | 0.283 |
| 152 | 32 | 4 | on | on | off | on | 0.648 | -0.011 | 0.369 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 153 | 16 | 4 | on | on | off | on | 0.638 | -0.022 | 0.716 |
| 154 | 8 | 4 | on | on | off | on | 0.632 | -0.051 | 1.539 |
| 155 | 4 | 4 | on | on | off | on | 0.631 | -0.093 | 2.747 |
| 156 | 64 | 2 | on | on | off | on | 0.656 | -0.010 | 0.338 |
| 157 | 32 | 2 | on | on | off | on | 0.628 | -0.012 | 0.424 |
| 158 | 16 | 2 | on | on | off | on | 0.620 | -0.024 | 0.745 |
| 159 | 8 | 2 | on | on | off | on | 0.614 | -0.058 | 1.806 |
| 160 | 4 | 2 | on | on | off | on | 0.613 | -0.105 | 3.148 |
| 161 | 64 | 1 | on | on | off | on | 0.650 | -0.012 | 0.415 |
| 162 | 32 | 1 | on | on | off | on | 0.625 | -0.014 | 0.482 |
| 163 | 16 | 1 | on | on | off | on | 0.615 | -0.026 | 0.871 |
| 164 | 8 | 1 | on | on | off | on | 0.609 | -0.055 | 1.747 |
| 165 | 4 | 1 | on | on | off | on | 0.608 | -0.115 | 3.493 |
| 166 | 64 | 4 | off | on | off | on | 0.610 | -0.025 | 0.849 |
| 167 | 32 | 4 | off | on | off | on | 0.584 | -0.027 | 0.924 |
| 168 | 16 | 4 | off | on | off | on | 0.574 | -0.036 | 1.169 |
| 169 | 8 | 4 | off | on | off | on | 0.568 | -0.066 | 2.062 |
| 170 | 4 | 4 | off | on | off | on | 0.566 | -0.109 | 3.307 |
| 171 | 64 | 2 | off | on | off | on | 0.591 | -0.026 | 0.900 |
| 172 | 32 | 2 | off | on | off | on | 0.565 | -0.028 | 0.954 |
| 173 | 16 | 2 | off | on | off | on | 0.556 | -0.040 | 1.329 |
| 174 | 8 | 2 | off | on | off | on | 0.550 | -0.068 | 2.149 |
| 175 | 4 | 2 | off | on | off | on | 0.550 | -0.121 | 3.662 |
| 176 | 64 | 1 | off | on | off | on | 0.586 | -0.028 | 0.944 |
| 177 | 32 | 1 | off | on | off | on | 0.560 | -0.027 | 0.939 |
| 178 | 16 | 1 | off | on | off | on | 0.551 | -0.042 | 1.402 |
| 179 | 8 | 1 | off | on | off | on | 0.545 | -0.071 | 2.292 |
| 180 | 4 | 1 | off | on | off | on | 0.544 | -0.129 | 3.993 |
| 181 | 64 | 4 | on | off | on | on | 0.539 | -0.022 | 0.659 |
| 182 | 32 | 4 | on | off | on | on | 0.522 | -0.024 | 0.737 |
| 183 | 16 | 4 | on | off | on | on | 0.516 | -0.036 | 1.054 |
| 184 | 8 | 4 | on | off | on | on | 0.512 | -0.066 | 1.979 |
| 185 | 4 | 4 | on | off | on | on | 0.512 | -0.104 | 3.174 |
| 186 | 64 | 2 | on | off | on | on | 0.525 | -0.024 | 0.770 |
| 187 | 32 | 2 | on | off | on | on | 0.508 | -0.023 | 0.734 |
| 188 | 16 | 2 | on | off | on | on | 0.502 | -0.038 | 1.142 |
| 189 | 8 | 2 | on | off | on | on | 0.499 | -0.070 | 2.163 |
| 190 | 4 | 2 | on | off | on | on | 0.498 | -0.117 | 3.472 |
| 191 | 64 | 1 | on | off | on | on | 0.521 | -0.026 | 0.857 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 192 | 32 | 1 | on | off | on | on | 0.504 | -0.026 | 0.844 |
| 193 | 16 | 1 | on | off | on | on | 0.497 | -0.038 | 1.127 |
| 194 | 8 | 1 | on | off | on | on | 0.494 | -0.067 | 2.082 |
| 195 | 4 | 1 | on | off | on | on | 0.494 | -0.125 | 3.799 |
| 196 | 64 | 4 | off | off | on | on | 0.491 | -0.038 | 1.256 |
| 197 | 32 | 4 | off | off | on | on | 0.474 | -0.039 | 1.277 |
| 198 | 16 | 4 | off | off | on | on | 0.468 | -0.052 | 1.673 |
| 199 | 8 | 4 | off | off | on | on | 0.465 | -0.078 | 2.451 |
| 200 | 4 | 4 | off | off | on | on | 0.464 | -0.123 | 3.742 |
| 201 | 64 | 2 | off | off | on | on | 0.477 | -0.040 | 1.297 |
| 202 | 32 | 2 | off | off | on | on | 0.460 | -0.039 | 1.278 |
| 203 | 16 | 2 | off | off | on | on | 0.454 | -0.053 | 1.676 |
| 204 | 8 | 2 | off | off | on | on | 0.451 | -0.082 | 2.582 |
| 205 | 4 | 2 | off | off | on | on | 0.450 | -0.133 | 4.072 |
| 206 | 64 | 1 | off | off | on | on | 0.473 | -0.043 | 1.387 |
| 207 | 32 | 1 | off | off | on | on | 0.456 | -0.041 | 1.349 |
| 208 | 16 | 1 | off | off | on | on | 0.450 | -0.053 | 1.711 |
| 209 | 8 | 1 | off | off | on | on | 0.447 | -0.083 | 2.632 |
| 210 | 4 | 1 | off | off | on | on | 0.446 | -0.141 | 4.372 |
| 211 | 64 | 4 | on | on | on | on | 0.447 | -0.031 | 0.969 |
| 212 | 32 | 4 | on | on | on | on | 0.431 | -0.032 | 1.018 |
| 213 | 16 | 4 | on | on | on | on | 0.425 | -0.045 | 1.391 |
| 214 | 8 | 4 | on | on | on | on | 0.422 | -0.072 | 2.214 |
| 215 | 4 | 4 | on | on | on | on | 0.422 | -0.112 | 3.432 |
| 216 | 64 | 2 | on | on | on | on | 0.437 | -0.034 | 1.092 |
| 217 | 32 | 2 | on | on | on | on | 0.420 | -0.034 | 1.091 |
| 218 | 16 | 2 | on | on | on | on | 0.415 | -0.045 | 1.425 |
| 219 | 8 | 2 | on | on | on | on | 0.412 | -0.076 | 2.366 |
| 220 | 4 | 2 | on | on | on | on | 0.413 | -0.125 | 3.762 |
| 221 | 64 | 1 | on | on | on | on | 0.434 | -0.036 | 1.187 |
| 222 | 32 | 1 | on | on | on | on | 0.417 | -0.034 | 1.133 |
| 223 | 16 | 1 | on | on | on | on | 0.412 | -0.047 | 1.472 |
| 224 | 8 | 1 | on | on | on | on | 0.410 | -0.078 | 2.439 |
| 225 | 4 | 1 | on | on | on | on | 0.409 | -0.135 | 4.151 |
| 226 | 64 | 4 | off | on | on | on | 0.411 | -0.048 | 1.566 |
| 227 | 32 | 4 | off | on | on | on | 0.394 | -0.049 | 1.612 |
| 228 | 16 | 4 | off | on | on | on | 0.389 | -0.059 | 1.890 |
| 229 | 8 | 4 | off | on | on | on | 0.385 | -0.087 | 2.734 |
| 230 | 4 | 4 | off | on | on | on | 0.385 | -0.130 | 3.998 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 231 | 64 | 2 | off | on | on | on | 0.400 | -0.048 | 1.563 |
| 232 | 32 | 2 | off | on | on | on | 0.384 | -0.049 | 1.601 |
| 233 | 16 | 2 | off | on | on | on | 0.379 | -0.061 | 1.956 |
| 234 | 8 | 2 | off | on | on | on | 0.375 | -0.091 | 2.851 |
| 235 | 4 | 2 | off | on | on | on | 0.374 | -0.140 | 4.326 |
| 236 | 64 | 1 | off | on | on | on | 0.397 | -0.050 | 1.654 |
| 237 | 32 | 1 | off | on | on | on | 0.380 | -0.049 | 1.616 |
| 238 | 16 | 1 | off | on | on | on | 0.375 | -0.062 | 2.032 |
| 239 | 8 | 1 | off | on | on | on | 0.372 | -0.093 | 2.923 |
| 240 | 4 | 1 | off | on | on | on | 0.372 | -0.148 | 4.585 |

Table D.2: LUT used for determination of the new $p_{(i)}$ encoding configuration.

| New Configuration ($p_{(i)}$) | | | | | | | | | | | | | | | | | | | | Previous Configuration ($p_{(i-1)}$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **21** | **20** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | |
| 0.581 | 0.596 | 0.618 | 0.624 | 0.629 | 0.643 | 0.651 | 0.665 | 0.703 | 0.746 | 0.762 | 0.811 | 0.864 | 0.869 | 0.896 | 0.907 | 0.931 | 0.941 | 0.965 | **1.000** | **2** |
| 0.602 | 0.617 | 0.640 | 0.646 | 0.652 | 0.666 | 0.675 | 0.689 | 0.728 | 0.772 | 0.789 | 0.840 | 0.895 | 0.900 | 0.928 | 0.940 | 0.964 | 0.974 | **1.000** | 1.036 | **3** |
| 0.618 | 0.634 | 0.657 | 0.663 | 0.669 | 0.684 | 0.693 | 0.707 | 0.747 | 0.793 | 0.810 | 0.862 | 0.919 | 0.924 | 0.953 | 0.965 | 0.990 | **1.000** | 1.026 | 1.063 | **4** |
| 0.624 | 0.640 | 0.664 | 0.670 | 0.676 | 0.691 | 0.700 | 0.715 | 0.755 | 0.801 | 0.819 | 0.871 | 0.929 | 0.933 | 0.963 | 0.975 | **1.000** | 1.010 | 1.037 | 1.074 | **5** |
| 0.640 | 0.657 | 0.681 | 0.688 | 0.694 | 0.709 | 0.718 | 0.733 | 0.774 | 0.822 | 0.840 | 0.893 | 0.953 | 0.957 | 0.988 | **1.000** | 1.026 | 1.037 | 1.064 | 1.102 | **6** |
| 0.648 | 0.665 | 0.690 | 0.696 | 0.702 | 0.718 | 0.727 | 0.742 | 0.784 | 0.832 | 0.850 | 0.904 | 0.965 | 0.969 | **1.000** | 1.012 | 1.039 | 1.049 | 1.077 | 1.116 | **7** |
| 0.669 | 0.686 | 0.712 | 0.718 | 0.725 | 0.740 | 0.750 | 0.766 | 0.809 | 0.858 | 0.877 | 0.933 | 0.995 | **1.000** | 1.032 | 1.045 | 1.072 | 1.083 | 1.111 | 1.151 | **8** |
| 0.672 | 0.690 | 0.715 | 0.722 | 0.728 | 0.744 | 0.754 | 0.770 | 0.813 | 0.862 | 0.882 | 0.938 | **1.000** | 1.005 | 1.037 | 1.050 | 1.077 | 1.088 | 1.117 | 1.157 | **9** |
| 0.717 | 0.735 | 0.763 | 0.770 | 0.776 | 0.794 | 0.804 | 0.821 | 0.867 | 0.920 | 0.940 | **1.000** | 1.067 | 1.072 | 1.106 | 1.119 | 1.148 | 1.160 | 1.191 | 1.234 | **10** |
| 0.762 | 0.782 | 0.811 | 0.819 | 0.826 | 0.844 | 0.855 | 0.873 | 0.922 | 0.978 | **1.000** | 1.064 | 1.134 | 1.140 | 1.176 | 1.191 | 1.221 | 1.234 | 1.267 | 1.312 | **11** |
| 0.779 | 0.800 | 0.829 | 0.837 | 0.844 | 0.863 | 0.874 | 0.892 | 0.942 | **1.000** | 1.022 | 1.087 | 1.160 | 1.165 | 1.202 | 1.217 | 1.249 | 1.262 | 1.295 | 1.341 | **12** |
| 0.827 | 0.848 | 0.880 | 0.888 | 0.896 | 0.915 | 0.927 | 0.947 | **1.000** | 1.061 | 1.085 | 1.154 | 1.230 | 1.236 | 1.276 | 1.291 | 1.325 | 1.339 | 1.374 | 1.423 | **13** |
| 0.873 | 0.896 | 0.929 | 0.938 | 0.946 | 0.967 | 0.979 | **1.000** | 1.056 | 1.121 | 1.146 | 1.218 | 1.299 | 1.306 | 1.347 | 1.364 | 1.399 | 1.414 | 1.451 | 1.503 | **14** |
| 0.892 | 0.915 | 0.949 | 0.958 | 0.966 | 0.987 | **1.000** | 1.021 | 1.079 | 1.144 | 1.170 | 1.244 | 1.327 | 1.333 | 1.376 | 1.393 | 1.429 | 1.444 | 1.482 | 1.535 | **15** |
| 0.903 | 0.927 | 0.961 | 0.970 | 0.978 | **1.000** | 1.013 | 1.034 | 1.092 | 1.159 | 1.185 | 1.260 | 1.344 | 1.351 | 1.394 | 1.411 | 1.447 | 1.462 | 1.501 | 1.555 | **16** |
| 0.923 | 0.947 | 0.982 | 0.991 | **1.000** | 1.022 | 1.035 | 1.057 | 1.116 | 1.185 | 1.211 | 1.288 | 1.374 | 1.380 | 1.424 | 1.442 | 1.479 | 1.495 | 1.534 | 1.589 | **17** |
| 0.931 | 0.956 | 0.991 | **1.000** | 1.009 | 1.031 | 1.044 | 1.067 | 1.126 | 1.195 | 1.222 | 1.299 | 1.386 | 1.392 | 1.437 | 1.455 | 1.492 | 1.508 | 1.548 | 1.603 | **18** |
| 0.940 | 0.964 | **1.000** | 1.009 | 1.018 | 1.040 | 1.054 | 1.076 | 1.136 | 1.206 | 1.233 | 1.311 | 1.398 | 1.405 | 1.450 | 1.468 | 1.506 | 1.521 | 1.562 | 1.617 | **19** |
| 0.974 | **1.000** | 1.037 | 1.046 | 1.056 | 1.079 | 1.093 | 1.116 | 1.179 | 1.251 | 1.278 | 1.360 | 1.450 | 1.457 | 1.503 | 1.522 | 1.561 | 1.578 | 1.619 | 1.677 | **20** |
| **1.000** | 1.026 | 1.064 | 1.074 | 1.083 | 1.107 | 1.121 | 1.145 | 1.210 | 1.283 | 1.312 | 1.395 | 1.488 | 1.495 | 1.543 | 1.562 | 1.602 | 1.619 | 1.662 | 1.721 | **21** |
| 1.007 | 1.034 | 1.072 | 1.082 | 1.091 | 1.115 | 1.129 | 1.153 | 1.218 | 1.293 | 1.321 | 1.405 | 1.499 | 1.506 | 1.554 | 1.573 | 1.614 | 1.631 | 1.674 | 1.734 | **22** |
| 1.012 | 1.039 | 1.077 | 1.087 | 1.096 | 1.120 | 1.135 | 1.159 | 1.224 | 1.299 | 1.328 | 1.412 | 1.506 | 1.513 | 1.561 | 1.581 | 1.622 | 1.639 | 1.682 | 1.742 | **23** |
| 1.019 | 1.046 | 1.085 | 1.095 | 1.104 | 1.129 | 1.143 | 1.167 | 1.233 | 1.308 | 1.337 | 1.422 | 1.517 | 1.524 | 1.573 | 1.592 | 1.633 | 1.650 | 1.694 | 1.755 | **24** |
| 1.050 | 1.078 | 1.118 | 1.128 | 1.138 | 1.163 | 1.178 | 1.203 | 1.270 | 1.348 | 1.378 | 1.465 | 1.563 | 1.570 | 1.620 | 1.640 | 1.683 | 1.700 | 1.745 | 1.808 | **25** |
| 1.094 | 1.122 | 1.164 | 1.174 | 1.185 | 1.211 | 1.227 | 1.253 | 1.323 | 1.404 | 1.435 | 1.526 | 1.628 | 1.635 | 1.688 | 1.708 | 1.753 | 1.771 | 1.818 | 1.883 | **26** |
| 1.105 | 1.134 | 1.176 | 1.187 | 1.197 | 1.224 | 1.239 | 1.266 | 1.337 | 1.418 | 1.450 | 1.542 | 1.645 | 1.653 | 1.705 | 1.726 | 1.771 | 1.789 | 1.837 | 1.903 | **27** |

| | | 27 | 26 | 25 | 24 | 23 | 22 |
|---|---|---|---|---|---|---|---|
| **Previous Configuration ($p_{(i-1)}$)** | **2** | 0.526 | 0.531 | 0.553 | 0.570 | 0.574 | 0.577 |
| | **3** | 0.544 | 0.550 | 0.573 | 0.590 | 0.595 | 0.597 |
| | **4** | 0.559 | 0.565 | 0.588 | 0.606 | 0.610 | 0.613 |
| | **5** | 0.565 | 0.571 | 0.594 | 0.612 | 0.617 | 0.620 |
| | **6** | 0.579 | 0.585 | 0.610 | 0.628 | 0.633 | 0.636 |
| | **7** | 0.586 | 0.593 | 0.617 | 0.636 | 0.640 | 0.644 |
| | **8** | 0.605 | 0.611 | 0.637 | 0.656 | 0.661 | 0.664 |
| | **9** | 0.608 | 0.614 | 0.640 | 0.659 | 0.664 | 0.667 |
| | **10** | 0.648 | 0.655 | 0.683 | 0.703 | 0.708 | 0.712 |
| | **11** | 0.690 | 0.697 | 0.726 | 0.748 | 0.753 | 0.757 |
| | **12** | 0.705 | 0.712 | 0.742 | 0.764 | 0.770 | 0.774 |
| | **13** | 0.748 | 0.756 | 0.787 | 0.811 | 0.817 | 0.821 |
| | **14** | 0.790 | 0.798 | 0.832 | 0.857 | 0.863 | 0.867 |
| | **15** | 0.807 | 0.815 | 0.849 | 0.875 | 0.881 | 0.885 |
| | **16** | 0.817 | 0.826 | 0.860 | 0.886 | 0.892 | 0.897 |
| | **17** | 0.835 | 0.844 | 0.879 | 0.905 | 0.912 | 0.916 |
| | **18** | 0.843 | 0.851 | 0.887 | 0.914 | 0.920 | 0.925 |
| | **19** | 0.850 | 0.859 | 0.895 | 0.922 | 0.928 | 0.933 |
| | **20** | 0.882 | 0.891 | 0.928 | 0.956 | 0.963 | 0.968 |
| | **21** | 0.905 | 0.914 | 0.952 | 0.981 | 0.988 | 0.993 |
| | **22** | 0.911 | 0.921 | 0.959 | 0.988 | 0.995 | **1.000** |
| | **23** | 0.916 | 0.925 | 0.964 | 0.993 | **1.000** | 1.005 |
| | **24** | 0.922 | 0.932 | 0.971 | **1.000** | 1.007 | 1.012 |
| | **25** | 0.950 | 0.960 | **1.000** | 1.030 | 1.038 | 1.043 |
| | **26** | 0.990 | **1.000** | 1.042 | 1.073 | 1.081 | 1.086 |
| | **27** | **1.000** | 1.011 | 1.053 | 1.084 | 1.092 | 1.097 |

# References

[1] W. Kim, J. You, and J. Jeong, "Complexity control strategy for real-time H.264/AVC encoder," *IEEE Transactions on Consumer Electronics,* vol. 56, pp. 1137-1143, 2010.

[2] S. B. Solak and F. Labeau, "Complexity scalable video encoding for power-aware applications," in *2010 International Green Computing Conference*, 2010, pp. 443-449.

[3] L. Xiaoan, W. Yao, and E. Erkip, "Power efficient H.263 video transmission over wireless channels," in *IEEE International Conference on Image Processing*, 2002, pp. 533-536.

[4] P. Agrawal, C. Jyh-Cheng, S. Kishore, P. Ramanathan, and K. Sivalingam, "Battery power sensitive video processing in wireless networks," in *9th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1998, pp. 116-120.

[5] A. K. Katsaggelos, Z. Fan, Y. Eisenberg, and R. Berry, "Energy-efficient wireless video coding and delivery," *IEEE Wireless Communications,* vol. 12, pp. 24-30, 2005.

[6] G. J. Sullivan, J. Ohm, H. Woo-Jin, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1649-1668, 2012.

[7] International Telecommunication Union, "ITU-T Recommendation H.264 (05/2003): advanced video coding for generic audiovisual services," 2003.

[8] C. S. Kannangara, I. E. Richardson, and A. J. Miller, "Computational Complexity Management of a Real-Time H.264/AVC Encoder," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 18, pp. 1191-1200, 2008.

[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, pp. 560-576, 2003.

[10] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Performance and Computational Complexity Assessment of High Efficiency Video Encoders," *IEEE*

*Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1899-1909, 2012.

[11] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Complexity Control of High Efficiency Video Encoders for Power-Constrained Devices," *IEEE Transactions on Consumer Electronics,* vol. 57, pp. 1866-1874, 2011.

[12] G. Correa, P. Assuncao, L. Agostini, and L. A. Da Silva Cruz, "Adaptive coding tree for complexity control of high efficiency video encoders," in *2012 Picture Coding Symposium*, 2012, pp. 425-428.

[13] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Dynamic tree-depth adjustment for low power HEVC encoders," in *19th IEEE International Conference on Electronics, Circuits and Systems*, 2012, pp. 564-567.

[14] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Motion compensated tree depth limitation for complexity control of HEVC encoding," in *2012 IEEE International Conference on Image Processing*, 2012, pp. 217-220.

[15] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Complexity control of HEVC through quadtree depth estimation," in *2013 IEEE EUROCON*, 2013, pp. 81-86.

[16] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Coding Tree Depth Estimation for Complexity Reduction of HEVC," in *2013 Data Compression Conference*, 2013, pp. 43-52.

[17] G. Correa, P. Assuncao, L. A. Da Silva Cruz, and L. Agostini, "Constrained Encoding Structures for Computational Complexity Scalability in HEVC," in *2013 Picture Coding Symposium*, 2013.

[18] G. Correa, P. Assuncao, L. Agostini, and L. Silva Cruz, "Complexity scalability for real-time HEVC encoders," *Journal of Real-Time Image Processing,* pp. 1-16, 2014, online early access.

[19] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Classification-Based Early Termination for Coding Tree Structure Decision in HEVC," in *20th IEEE International Conference on Electronics, Circuits, and Systems*, submitted: June 13, 2014.

[20] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Fast HEVC Encoding Decisions Using Data Mining," *IEEE Transactions on Circuits and Systems for Video Technology,* pp. 1-14, submitted: May 4, 2014, revised: July 15, 2014.

[21] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "A Method for Early-Splitting of HEVC Inter Blocks Based on Decision Trees," in *2014 European Signal Processing Conference*, 2014, accepted for publication.

[22] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Four-step Algorithm for Early Termination in HEVC Inter-frame Prediction based on Decision Trees," in *Visual Communications and Image Processing*, submitted: May 26, 2014.

[23] I. E. Richardson, *Video Codec Design: Developing Image and Video Compression Systems*. Chichester: John Wiley and Sons, 2002.

[24] I. E. Richardson, *H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia*. Chichester: John Wiley and Sons, 2003.

[25] M. Ghanbari, *Standard Codecs: Image Compression to Advanced Video Coding*. UK: The Institute of Electrical Engineers, 2003.

[26] Y. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms and Standards*. Boca Raton: CRC Press, 1999.

[27] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," doc. VCEG-M33, 2001.

[28] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine,* vol. 15, pp. 74-90, 1998.

[29] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine,* vol. 15, pp. 23-50, 1998.

[30] ISO/IEC-JCT1/SC29/WG11, "High Efficiency Video Coding (HEVC) Test Model 13 (HM 13) Encoder Description," doc. JCTVC-O1002, 2013.

[31] J. Lainema, F. Bossen, H. Woo-Jin, M. Junghye, and K. Ugur, "Intra Coding of the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1792-1801, 2012.

[32] ISO/IEC-JCT1/SC29/WG11, "Encoder improvement of unified intra prediction," doc. JCTVC-C207, 2010.

[33] X. Xu and Y. He, "Comments on Motion Estimation Algorithms in Current JM Software," doc. JVT-Q089, 2005.

[34] N. Purnachand, L. N. Alves, and A. Navarro, "Improvements to TZ search motion estimation algorithm for multiview video coding," in *2012 19th International Conference on Systems, Signals and Image Processing*, 2012, pp. 388-391.

[35] P. Helle*, et al.*, "Block Merging for Quadtree-Based Partitioning in HEVC," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1720-1731, 2012.

[36] ISO/IEC-JTC1/SC29/WG11, "Samsung's Response to the Call for Proposals on Video Compression Technology," doc. JCTVC-A124, 2010.

[37] G. RyeongHee and L. Yung-Lyul, "N-level quantization in HEVC," in *2012 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2012, pp. 1-5.

[38] Y. Y. M. Karczewicz, I. Chong, "Rate distortion optimized quantization," doc VCEG-AH21, 2008.

[39] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 13, pp. 620-636, 2003.

[40] A. Norkin*, et al.*, "HEVC Deblocking Filter," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1746-1754, 2012.

[41] F. Chih-Ming, *et al.*, "Sample Adaptive Offset in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1755-1764, 2012.

[42] ISO/IEC-JCT1/SC29/WG11, "High Efficiency Video Coding (HEVC) text specification draft 10," doc. JCTVC-L1003, 2013.

[43] ISO/IEC-JCT1/SC29/WG11, "Common test conditions and software reference configurations," doc. JCTVC-J1100, 2012.

[44] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1685-1696, 2012.

[45] M. Mattavelli and S. Brunetton, "Implementing real-time video decoding on multimedia processors by complexity prediction techniques," *IEEE Transactions on Consumer Electronics,* vol. 44, pp. 760-767, 1998.

[46] J. Valentim, P. Nunes, and F. Pereira, "Evaluating MPEG-4 video decoding complexity for an alternative video complexity verifier model," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 12, pp. 1034-1044, 2002.

[47] M. van der Schaar and Y. Andreopoulos, "Rate-distortion-complexity modeling for network and receiver aware adaptation," *IEEE Transactions on Multimedia,* vol. 7, pp. 471-479, 2005.

[48] Y. Wang and C. Shih-Fu, "Complexity Adaptive H.264 Encoding for Light Weight Streams," in *2006 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006, p. II.

[49] L. Szu-Wei and C. C. J. Kuo, "Motion Compensation Complexity Model for Decoder-Friendly H.264 System Design," in *2007 IEEE Workshop on Multimedia Signal Processing*, 2007, pp. 119-122.

[50] C. E. Rhee, J. S. Jung, and H. J. Lee, "A Real-Time H.264/AVC Encoder With Complexity-Aware Time Allocation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 1848-1862, 2010.

[51] H. Zhihai, L. Yongfang, C. Lulin, I. Ahmad, and W. Dapeng, "Power-rate-distortion analysis for wireless video communication under energy constraints," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 15, pp. 645-658, 2005.

[52] L. Xiang, M. Wien, and J. R. Ohm, "Rate-Complexity-Distortion Optimization for Hybrid Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 21, pp. 957-970, 2011.

[53] M. Zhan, H. Hao, and W. Yao, "On Complexity Modeling of H.264/AVC Video Decoding and Its Application for Energy Efficient Decoding," *IEEE Transactions on Multimedia,* vol. 13, pp. 1240-1255, 2011.

[54] Ray Garcia, Damian Ruiz Coll, Hari Kalva, and G. Fernandez-Escribano, "HEVC Decision Optimization for Low Bandwidth in Video Conferencing Applications in

Mobile Environments " in *2013 IEEE International Conference on Multimedia and Expo*, 2013.

[55] A. Yong-Jo, H. Tae-Jin, S. Dong-Gyu, and H. Woo-Jin, "Complexity model based load-balancing algorithm for parallel tools of HEVC," in *2013 Visual Communications and Image Processing*, 2013, pp. 1-5.

[56] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *National Telecommunications Conference*, 1981, pp. G5.3.1-G5.3.5.

[57] L. Lurng-Kuo and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 6, pp. 419-422, 1996.

[58] L. Renxiang, Z. Bing, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 4, pp. 438-442, 1994.

[59] T. Jo Yew, R. Surendra, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 8, pp. 369-377, 1998.

[60] Z. Shan and M. Kai-Kuang, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing,* vol. 9, pp. 287-290, 2000.

[61] Z. Ce, L. Xiao, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 12, pp. 349-355, 2002.

[62] B. Xuan-Quang and T. Yap-Peng, "Adaptive dual-cross search algorithm for block-matching motion estimation," *IEEE Transactions on Consumer Electronics,* vol. 50, pp. 766-775, 2004.

[63] Y.-L. S. Lin, C.-Y. Kao, H.-C. Kuo, and J.-W. Chen, *VLSI Design for Video Coding: H.264/AVC Encoding from Standard Specification to Chip*. New York: Springer Publishing Company, 2010.

[64] S. Eckart and C. E. Fogg, "ISO-IEC MPEG-2 software video codec," in *SPIE Conf. Visual Communications and Image Processing*, 1995, pp. 100-109.

[65] C. Chok-Kwan and P. Lai-Man, "Normalized partial distortion search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 10, pp. 417-422, 2000.

[66] K. Lengwehasarit and A. Ortega, "Probabilistic partial-distance fast matching algorithms for motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 11, pp. 139-152, 2001.

[67] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing,* vol. 4, pp. 105-107, 1995.

[68] M. Moecke and R. Seara, "Sorting Rates in Video Encoding Process for Complexity Reduction," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 88-101, 2010.

[69] S. Saponara, M. Casula, F. Rovati, D. Alfonso, and L. Fanucci, "Dynamic control of motion estimation search parameters for low complex H.264 video coding," *IEEE Transactions on Consumer Electronics,* vol. 52, pp. 232-239, 2006.

[70] M. Bystrom, I. Richardson, and Y. Zhao, "Efficient mode selection for H.264 complexity reduction in a Bayesian framework," *Signal Processing: Image Communication,* vol. 23, pp. 71-86, 2008.

[71] L.-J. Pan and H. Yo-Sung, "A Fast Mode Decision Algorithm for H.264/AVC Intra Prediction," in *2007 IEEE Workshop on Signal Processing Systems*, 2007, pp. 704-709.

[72] C. Chun-Hao*, et al.*, "A Quality Scalable H.264/AVC Baseline Intra Encoder for High Definition Video Applicaitons," in *2007 IEEE Workshop on Signal Processing Systems*, 2007, pp. 521-526.

[73] L. De-Wei, K. Chun-Wei, C. Chao-Chung, L. Yu-Kun, and C. Tian-Sheuan, "A 61MHz 72K Gates 1280x720 30FPS H.264 Intra Encoder," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2007, pp. 801-804.

[74] T. An-Chao, W. Jhing-Fa, Y. Jar-Ferr, and L. Wei-Guang, "Effective Subblock-Based and Pixel-Based Fast Direction Detections for H.264 Intra Prediction," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 18, pp. 975-982, 2008.

[75] L. Yu-Ming, S. Yu-Ting, and L. Yinyi, "SATD-Based Intra Mode Decision for H.264/AVC Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 463-469, 2010.

[76] K. Hyungjoon and Y. Altunhasak, "Low-complexity macroblock mode selection for H.264-AVC encoders," in *2004 International Conference on Image Processing*, 2004, pp. 765-768.

[77] F. Wang, Y. Fan, Y. Lan, and W. Liu, "Fast intra mode decision algorithm in H.264/AVC using characteristics of transformed coefficients," in *2008 International Conference on Visual Information Engineering*, 2008, pp. 245-249.

[78] L. Yu-Ming, W. Jyun-De, and L. Yinyi, "An improved SATD-based intra mode decision algorithm for H.264/AVC," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 1029-1032.

[79] G. Correa*, et al.*, "Homogeneity and distortion-based intra mode decision architecture for H.264/AVC," in *2010 IEEE International Conference on Electronics, Circuits, and Systems*, 2010, pp. 591-594.

[80] G. Correa, D. Palomino, C. Diniz, L. Agostini, and S. Bampi, "SHBS: A heuristic for fast inter mode decision of H.264/AVC standard targeting VLSI design," in *2011 IEEE International Conference on Multimedia and Expo*, 2011, pp. 1-4.

[81] G. Correa, D. Palomino, C. Diniz, S. Bampi, and L. Agostini, "Low-Complexity Hierarchical Mode Decision Algorithms Targeting VLSI Architecture Design for the H.264/AVC Video Encoder," *VLSI Design,* vol. 2012, pp. 1-20, 2012.

[82] L. Jeyun and J. Byeungwoo, "Fast mode decision for H.264," in *2004 IEEE International Conference on Multimedia and Expo*, 2004, pp. 1131-1134.

[83] D. Wu*, et al.*, "Fast intermode decision in H.264/AVC video coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 15, pp. 953-958, 2005.

[84] S. Liquan, L. Zhi, Z. Zhaoyang, and S. Xuli, "Fast Inter Mode Decision Using Spatial Property of Motion Field," *IEEE Transactions on Multimedia,* vol. 10, pp. 1208-1214, 2008.

[85] C. Seunghyun and K. Munchurl, "Fast CU Splitting and Pruning for Suboptimal CU Partitioning in HEVC Intra Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 23, pp. 1555-1564, 2013.

[86] M. B. Cassa, M. Naccari, and F. Pereira, "Fast rate distortion optimization for the emerging HEVC standard," in *2012 Picture Coding Symposium*, 2012, pp. 493-496.

[87] L. Jong-Hyeok, P. Chan-Seob, and K. Byung-Gyu, "Fast coding algorithm based on adaptive coding depth range selection for HEVC," in *2012 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2012, pp. 31-33.

[88] Kalyan Goswami, Byung-Gyu Kim, Dong-San Jun, Soon-Heung Jung, and J. S. Choi, "Early Coding Unit (CU) Splitting Termination Algorithm for High Efficiency Video Coding (HEVC)," *Electronics and Telecommunications Research Institute Journal,* 2014, accepted for publication.

[89] X. Jian, L. Hongliang, W. Qingbo, and M. Fanman, "A Fast HEVC Inter CU Selection Method Based on Pyramid Motion Divergence," *IEEE Transactions on Multimedia,* vol. 16, pp. 559-564, 2014.

[90] S.-C. Tai, C.-Y. Chang, B.-J. Chen, and J.-F. Hu, "Speeding Up the Decisions of Quad-Tree Structures and Coding Modes for HEVC Coding Units," in *Advances in Intelligent Systems and Applications - Volume 2*, edited by J.-S. Pan*, et al.*, Berlin: Springer, 2013, pp. 393-401.

[91] Y. Zhang, H. Wang, and Z. Li, "Fast Coding Unit Depth Decision Algorithm for Interframe Coding in HEVC," in *2013 Data Compression Conference*, 2013, pp. 53-62.

[92] L. Jie, S. Lei, T. Ikenaga, and S. Sakaida, "Content Based Hierarchical Fast Coding Unit Decision Algorithm for HEVC," in *2011 International Conference on Mutimedia and Signal Processing*, 2011, pp. 56-59.

[93] J. Vanne, M. Viitanen, and T. Hamalainen, "Efficient Mode Decision Schemes for HEVC Inter Prediction," *IEEE Transactions on Circuits and Systems for Video Technology,* 2014, accepted for publication.

[94] F. Sampaio, S. Bampi, M. Grellert, L. Agostini, and J. Mattos, "Motion Vectors Merging: Low Complexity Prediction Unit Decision Heuristic for the Inter-

prediction of HEVC Encoders," in *2012 IEEE International Conference on Multimedia and Expo*, 2012, pp. 657-662.

[95] T. Guifen and S. Goto, "Content adaptive prediction unit size decision algorithm for HEVC intra coding," in *2012 Picture Coding Symposium*, 2012, pp. 405-408.

[96] K. Jaehwan, Y. Jungyoup, W. Kwanghyun, and J. Byeungwoo, "Early determination of mode decision for HEVC," in *2012 Picture Coding Symposium*, 2012, pp. 449-452.

[97] M. U. K. Khan, M. Shafique, and J. Henkel, "An Adaptive Complexity Reduction Scheme with Fast Prediction Unit Decision for HEVC Intra Encoding," in *IEEE International Conference on Image Processing*, 2013.

[98] Y. Shi, Z. Gao, and X. Zhang, "Early TU Split Termination in HEVC Based on Quasi-Zero-Block," in *3rd International Conference on Electric and Electronics*, 2013, pp. 450-454.

[99] C. Kiho and E. S. Jang, "Early TU decision method for fast video encoding in high efficiency video coding," *IET Electronics Letters,* vol. 48, pp. 689-691, 2012.

[100] H. Wei-Jhe and H. Hsueh-Ming, "Fast coding unit decision algorithm for HEVC," in *2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2013, pp. 1-5.

[101] S. Liquan, L. Zhi, Z. Xinpeng, Z. Wenqiang, and Z. Zhaoyang, "An Effective CU Size Decision Method for HEVC Encoders," *IEEE Transactions on Multimedia,* vol. 15, pp. 465-470, 2013.

[102] S. Xiaolin, Y. Lu, and C. Jie, "Fast coding unit size selection for HEVC based on Bayesian decision rule," in *2012 Picture Coding Symposium*, 2012, pp. 453-456.

[103] T. L. da Silva, L. V. Agostini, and L. A. da Silva Cruz, "Fast HEVC intra prediction mode decision based on EDGE direction information," in *20th European Signal Processing Conference,* 2012, pp. 1214-1218.

[104] P. Jain, A. Laffely, W. Burleson, R. Tessier, and D. Goeckel, "Dynamically Parameterized Algorithms and Architectures to Exploit Signal Variations," *The Journal of VLSI Signal Processing,* vol. 36, pp. 27-40, 2004.

[105] L. Weiyao, K. Panusopone, D. M. Baylon, and S. Ming-Ting, "A Computation Control Motion Estimation Method for Complexity-Scalable Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 1533-1543, 2010.

[106] E. Akyol, D. Mukherjee, and L. Yuxin, "Complexity Control for Real-Time Video Coding," in *2007 IEEE International Conference on Image Processing*, 2007, pp. 77-80.

[107] T. A. da Fonseca and R. L. de Queiroz, "Macroblock sampling and mode ranking for complexity scalability in mobile H.264 video coding," in *2009 IEEE International Conference on Image Processing*, 2009, pp. 3753-3756.

[108] S. Li, L. Yan, W. Feng, L. Shipeng, and G. Wen, "Complexity-Constrained H.264 Video Encoding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 19, pp. 477-490, 2009.

[109] L. Xiang, M. Wien, and J. R. Ohm, "Medium-granularity computational complexity control for H.264/AVC," in *2010 Picture Coding Symposium*, 2010, pp. 214-217.

[110] C. S. Kannangara, I. E. Richardson, M. Bystrom, and Z. Yafan, "Complexity Control of H.264/AVC Based on Mode-Conditional Cost Probability Distributions," *IEEE Transactions on Multimedia,* vol. 11, pp. 433-442, 2009.

[111] X. Li, Y. Cui, and Y. Xue, "Towards an Automatic Parameter-Tuning Framework for Cost Optimization on Video Encoding Cloud," *International Journal of Digital Multimedia Broadcasting,* vol. 2012, pp. 1-11, 2012.

[112] R. Vanam, E. Riskin, R. Ladner, and S. Hemami, "Fast algorithms for designing nearly optimal lookup tables for complexity control of the H.264 encoder," *Signal, Image and Video Processing*, vol. 7, pp. 991-1003, 2013.

[113] L. Breiman, *Classification and regression trees*. Monterey: Chapman & Hall, 1984.

[114] Z. Sun, C. Xi, and H. Zhihai, "Adaptive Critic Design for Energy Minimization of Portable Video Communication Devices," in *2008 International Conference on Computer Communications and Networks*, 2008, pp. 1-5.

[115] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks,* vol. 8, pp. 997-1007, 1997.

[116] I. R. Ismaeil, A. Docef, F. Kossentini, and R. K. Ward, "A computation-distortion optimized framework for efficient DCT-based video coding," *IEEE Transactions on Multimedia,* vol. 3, pp. 298-310, 2001.

[117] L. Yongfang and I. Ahmad, "Power and Distortion Optimization for Pervasive Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 19, pp. 1436-1447, 2009.

[118] H. Zhihai, C. Wenye, and C. Xi, "Energy Minimization of Portable Video Communication Devices Based on Power-Rate-Distortion Optimization," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 18, pp. 596-608, 2008.

[119] W. Ji, J. Liu, M. Chen, and Y. Chen, "Power-efficient video encoding on resource-limited systems: A game-theoretic approach," *Future Generation Computer Systems,* vol. 28, pp. 427-436, 2012.

[120] T. A. da Fonseca and R. L. de Queiroz, "Complexity-Constrained Rate-Distortion Optimization for H.264/AVC Video Coding," in *2011 IEEE International Symposium on Circuits and Systems*, 2011, pp. 2909-2912.

[121] A. Ukhanova, S. Milani, and S. Forchhammer, "Game-Theoretic Rate-Distortion-Complexity Optimization for HEVC," in *2013 IEEE International Conference on Image Processing*, 2013, pp. 1995-1999.

[122] T. Zhao, Z. Wang, and S. Kwong, "Flexible Mode Selection and Complexity Allocation in High Efficiency Video Coding," *IEEE Journal of Selected Topics in Signal Processing,* vol. 7, pp. 1135-1144, 2013.

[123] ISO/IEC-JCT1/SC29/WG11, "High Efficiency Video Coding (HEVC) Test Model 7 (HM 7) Encoder Description," doc. JCTVC-I1002, 2012.

[124] Intel Corporation, *VTune™ Amplifier XE*, available at: <http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/> , August 2014.

[125] ISO/IEC-JCT1/SC29/WG11, "JCT-VC AHG report: complexity assessment (AHG 12)," doc. JCTVC-G012, 2011.

[126] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo: Morgan Kaufmann Publishers, 1993.

[127] M. Hall*, et al.*, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, pp. 10-18, 2009.

[128] N. Japkowicz, "The Class Imbalance Problem: Significance and Strategies," in *2000 International Conference on Artificial Intelligence*, 2000, pp. 111-117.

[129] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New Jersey: Wiley-Interscience, 2006.

[130] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79-86, 1951.

[131] X. Shen and L. Yu, "CU splitting early termination based on weighted SVM," *EURASIP Journal on Image and Video Processing,* vol. 2013, p. 4, 2013.

[132] L. Szu-Wei and C. C. J. Kuo, "Complexity Modeling of Spatial and Temporal Compensations in H.264/AVC Decoding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, pp. 706-720, 2010.