TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

ADS	Absolute Difference of Sums
AMP	Asymmetric Motion Partition
AVC	Advanced Video Coding
BMA	Block-Matching Algorithm
BRTRR	Bi-Predictive Refinement Time Reduction Ratio
CIME	Confidence Interval-based Motion Estimation
CPU	Central Processing Unit
CTU	Coding Tree Unit
EPZS	Enhanced Predictive Zonal Search
ESA	Exhaustive Search Algorithm
ETRR	Encoding Time Reduction Ratio
FEN	Fast ENcoder decision
FGSE	Fine Granularity Successive Elimination
HEVC	High Efficiency Video Coding
ILMVP	Integer Level Motion Vector Predictor
IMETRR	Integer level Motion Estimation Time Reduction Ratio
LD	Low delay
MC	Motion Compensation

ME	Motion Estimation
ML-RCSEA	Multi-Level Rate-Constrained Successive Elimination Algorithm
MSEA	Multilevel Successive Elimination Algorithm
MV	Motion Vector
MVFAST	Motion Vector Field Adaptive Search Technique
MVP	Motion Vector Predictor
PMVFAST	Predictive Motion Vector Field Adaptive Search Technique
PSNR	Peak Signal to Noise Ratio
PU	Prediction Unit
QP	Quantization Parameter
RA	Random Access
RCADS	Rate-Constrained Absolute Difference of Sums
RCBMA	Rate-Constrained Block-Matching Algorithm
RCSAD	Rate-Constrained Sum of the Absolute Differences
RCSEA	Rate-Constrained Successive Elimination Algorithm
RD	Rate Distortion
SAD	Sum of the Absolute Differences
SATD	Sum of the Absolute Transformed Differences
SEA	Successive Elimination Algorithm
TSS	Three Step Search
UCBDS	Unrestricted Center-Biased Diamond Search

LISTE OF SYMBOLS AND UNITS OF MEASUREMENTS

$\mathbf{x} = (x, y)$	Pixel coordinate in the frame: x and y represent the horizontal and vertical positions, respectively.
$\psi(\mathbf{x})$	Frame pixel value at location $\mathbf{x}$.
$\psi'(\mathbf{x})$	Reconstructed frame pixel value at location $\mathbf{x}$.
$\delta(\mathbf{x})$	Residual frame value at pixel location $\mathbf{x}$.
$\mathbf{P}$	Bold uppercase letters to refer to a set of pixel coordinate pairs representing a block or a partition.
$\mathbf{S}^k$	Set of pixel coordinate pairs for a square block or partition, $\mathbf{S}^k = \left\{(x,y) \mid 0 \leqslant x,y < 2^k\right\}, \quad k \in \{3,4,5,6\}$.
$\mathbf{H}^k$	Set of pixel coordinate pairs for an horizontal rectangular block or partition, $\mathbf{H}^k = \left\{(x,y) \mid 0 \leqslant x < 2^k, 0 \leqslant y < 2^{k-1}\right\}, \quad k \in \{3,4,5,6\}$.
$\mathbf{V}^k$	Set of pixel coordinate pairs for a vertical rectangular block or partition $\mathbf{V}^k = \left\{(x,y) \mid 0 \leqslant x < 2^{k-1}, 0 \leqslant y < 2^k\right\}, \quad k \in \{3,4,5,6\}$.
$(x,y) + \mathbf{P}$	Element-wise addition of the (x,y) pair to each pixel pair in $\mathbf{P}$ $(x,y) + \mathbf{P} = \{(x+i, y+j) \mid (i,j) \in \mathbf{P}\}$.
$\mathscr{P}$	Styled capital letters refer to a set of sets of pixel coordinate pairs.
$(x,y) + \mathscr{P}$	Element-wise addition of the (x,y) pair to each element in $\mathscr{P}$ $(x,y) + \mathscr{P} = \{(x,y) + \mathbf{P} \mid \mathbf{P} \in \mathscr{P}\}$.
$\mathbf{C}^r$	Set of pixel coordinate pairs of the candidate blocks in a search area of radius r, $\mathbf{C}^r = \{(x,y) \mid -r \leqslant x,y \leqslant r\}$.
$\mathbf{F}$	The set of candidates evaluated so far in the search area, $\mathbf{F} \subseteq \mathbf{C}^r$.

$\mathscr{D}$ The set of pairs of partition shapes and motion vectors representing the metadata related to the decision made by the encoder to predict the current frame.

INTRODUCTION

Problem Statement

Video services are an essential part of consumers' lives. According to Cisco (2016), since 2012, mobile video represents more than half of global mobile data traffic. In 2015, the average monthly amount of global mobile traffic reached 3.7 Exabytes, 55 percent of which was mobile video traffic. Bandwidth demand for data and video is increasing and shows no sign of stopping. Cisco predicts that mobile video will increase eleven folds between 2015 and 2020, accounting for 75 percent of total mobile data traffic by the end of its forecast period.

Video streaming is a prime concern for network operators, as it requires much higher bit rates than other types of content. Video compression standards strive to reduce these bit rates. They have done so by considerably increasing the number of tools that encoders can make use of, resulting in what is referred to as a feature-rich video compression standard. Although feature-rich video compression standards considerably outperform their predecessors, these gains are offset by consumer demands for improved visual quality and higher resolution content.

Video encoders compliant with feature-rich video compression standards like H.264/MPEG-4 advanced video coding (AVC) (ITU-T SG16 Q.6 and ISO/IEC JTC 1/SC 29/WG11, 2003) and H.265/high efficiency video coding (HEVC) (ISO/IEC JTC 1/SC 29/WG11, 2015) heavily rely on motion estimation (ME) to achieve their high rate distortion (RD) performance. As such, ME algorithms are applied to more block sizes, to more anchor frames, and over bigger search areas than their former counterparts (Sullivan *et al.*, 2012). As shown in (Hu and Yang, 2014), the integer-level ME time increased 12-fold from the H.264 reference encoder to the HEVC reference encoder.

All these new features have made the solution space of ME so big, that evaluating it entirely is prohibitively expensive. As such, modern ME algorithms are suboptimal, in that they evaluate only a small subset of the solution space and will often fail to find the optimal solution. Suboptimal solutions not only reduce visual quality but also increase the bit-rate.

Motivations

Approaches like the successive elimination algorithm (SEA) (Li and Salari, 1995) and its derivatives, rate-constrained successive elimination algorithm (RCSEA) (Coban and Mersereau, 1998), multilevel successive elimination algorithm (MSEA) (Gao *et al.*, 2000) and fine granularity successive elimination (FGSE) (Zhu *et al.*, 2005) show great potential in reducing the solution space of ME without reducing visual quality or increasing the bit-rate. These approaches eliminate elements of the solution space that cannot be optimal. As such, they preserve the optimality of the solution.

While some of these approaches predate modern feature-rich video compression standards, they are rarely used in modern encoders. One reason for this is that the mathematical models need to be updated to support features found in modern video compression standards. Another reason is that these algorithms target the exhaustive search algorithm (ESA). Even when the ESA is combined with the SEA, the amount of computation required remains unaffordable in a commercial context.

Objectives

The main objective of this research effort is to improve the efficiency of SEAs in order to reduce their computational requirements. One way of doing this is to investigate innovative and more efficient ways for SEA to support the new features found in modern video compression standards. More efficient SEAs can benefit both optimal and suboptimal ME algorithms.

In the context of optimal ME algorithms, reducing the computational requirements will make ESA combined with SEAs more affordable. Nevertheless, it is not expected to make optimal algorithms affordable for the commercial context. Findings related to optimal ME allow greater insight into which elements of the solution space needs to be evaluated.

SEAs have never been targeted at suboptimal ME algorithms, as it is widely believed that the overhead of SEA outweighs the benefit for this class of algorithms. Our objective is to evaluate

this claim and determine if using SEAs on more modern and more complex suboptimal ME algorithms, like the TZ-Search, is justifiable. Reducing the computational requirements of these algorithms would allow them to consider bigger parts of the solution space, thus possibly finding better solutions, which in turn, might improve visual quality or reduce the bit-rate.

We also hope that by improving the mathematical models used to describe the SEA and making them compatible with modern encoding tools, more interest will develop around SEAs. Better models can also facilitate the implementation of SEAs in modern encoders.

Thesis Structure

To better understand the context related to our research effort, we give a brief overview of a modern video encoder in chapter 1. As our work mainly focuses on temporal prediction, this subject is the focus of chapter 2. This second chapter explains ME and motion compensation (MC), and also describe the temporal prediction features found in modern video compression standards.

Next, we move on to our literature review which covers SEAs in chapter 3 and ME algorithms in chapter 4. We use the plural of SEA (i.e. SEAs) when referring to the SEA and its derivatives: RCSEA, MSEA and FGSE, which are all presented in chapter 3. As the amount of research on ME algorithms is quite extensive, we focused our review on the algorithms that were implemented in the reference software encoders or algorithms that influenced them.

Chapters 5, 6 and 7 describe our contributions to SEA for optimal and suboptimal ME algorithms. We discovered very early on that the ordering by which the SEA evaluates candidates has a direct impact on its computational requirements.

In chapter 5, we introduce the fact that the search ordering can weaken transitive elimination. This leads to a new concept, that we refer to as a SEA-Optimal search ordering. We published a paper on the subject (Trudeau *et al.*, 2015a) showing that, on average, an SEA-Optimal search ordering reduces sum of the absolute differences (SAD) operations by 3.66% for the HEVC

reference software encoder. For smaller block sizes, the average rises to 8.06%. We also filed a patent application for SEA-Optimal search orderings methods in (Trudeau *et al.*, 2016c).

Chapter 5 also presents the novel concept, the *increasing rate rule*. This rule dictates that, to avoid weakening transitive elimination, the search ordering must be ordered by motion vector cost. Chronologically, we published our paper about the increasing rate rule (Trudeau *et al.*, 2014) before our work on SEA-Optimal search orderings. As such, our results are based on a previous generation video codec. These results show that, on average, for the H.264 reference software encoder, the number of SAD operations is reduced by 2.86%. For smaller block sizes, this can exceed 10%. We patented the rate-constrained search ordering method in Trudeau *et al.* (2015b).

Finally, chapter 5 also introduces a fast cost-based search ordering algorithm, which decreases the number of SAD operations by approximately 3%. It allows for a new early-termination criterion which only requires performing 36% and 46% of block-matching loop iterations for Random Access and Low Delay respectively. This new solution is more than five times faster than the HEVC HM encoder in full search mode, without impact on visual quality or rate. A journal paper has been written on this topic and has been submitted to IEEE Transactions on Circuits and Systems for Video Technology.

In chapter 6, we present a novel form of information reuse inside the encoder to reduce the ME solution space. We published a paper on the subject Trudeau *et al.* (2016b) which shows that when combined with the RCSEA in the HEVC HM encoder reference software, the number of SAD operations drops by an average of 94.9%, resulting in an average speedup of 6.13x in full search mode.

Whereas in chapter 7, we adapt the SEA to the suboptimal ME algorithm found in the HEVC reference software encoder, TZ-Search. It reduces the motion estimation time by approximately 45% contributing to an average encoding time reduction of about 7% without impact on visual quality or rate. A journal paper has been written on this topic and has been submitted to IEEE Transactions on Circuits and Systems for Video Technology.

CHAPTER 1

VIDEO ENCODER OVERVIEW

Although the compression efficiency of video encoders has greatly evolved since the 1990s, their core concepts have not. Every major video coding standard since the early 1990s is based on the same generic block-based hybrid design (Richardson, 2010, p. 68).

In this chapter, we present an overview of a generic block-based hybrid video encoder followed by a description of the quad-tree structure used to signal block sizes. Modern encoding standards rely on this recursive structure to improve compression as it allows the encoder to adjust the block sizes to the content of the video sequence.

1.1 Overview of a Block-Based Hybrid Video Encoder

Block-based video encoders separate the current frame into non-overlapping pixel regions called blocks. The term hybrid denotes the combination of predictive coding and transform coding. In fig. 1.1, we outline the general design of block-based hybrid video encoders.

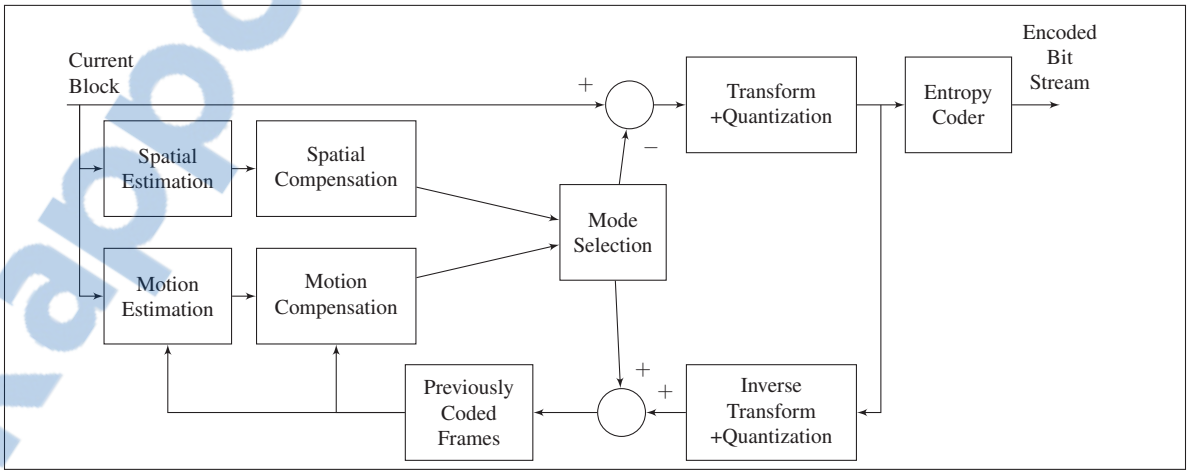


Figure 1.1 Generic outline of the design of a block-based hybrid video encoder

The design of every block-based hybrid video encoder can be separated into three distinct stages:

Predictive coding

Instead of coding the pixel values of the current block, a predictive coding scheme codes the residual (i.e., the subtraction) between the pixel values and a prediction (i.e., spatial or temporal) of the current block. A spatial prediction is based on previously encoded blocks in the current frame, whereas a temporal prediction is based on blocks in previously encoded frames. Estimation is the name given to the process of finding the prediction and compensation the name given to the process of producing the prediction. When multiple predictions are available, the mode selection module will select the one that minimizes a given criterion. Only the residual and the metadata, required to reproduce the prediction, are used in subsequent phases.

Transform and quantization coding

If information remains in the residual, it is compacted using a frequency transform. Furthermore, the transformed coefficients are mapped to a predefined subset of values, this is referred to as quantization.

Entropy coding The metadata required to reproduce the prediction and the quantized coefficients are converted to binary codewords of variable lengths. The length of each codeword is chosen in order to minimize the overall length of the bit stream.

1.2 Quad-Tree-Structured Block-Based Video Coding

A key element to the efficiency of block-based video coding is the size of the block used for prediction and transform coding. Smaller blocks allow for more precise prediction. However,

they require more signaling metadata in the bit stream. Evaluating more block sizes also increase computational complexity of the encoder.

Modern video compression standards, like AVC (Wiegand *et al.*, 2003, p. 569) and HEVC (Sullivan *et al.*, 2012, p. 1659), use a quad-tree structure to recursively decompose blocks into smaller blocks. By doing so, the encoder can adjust block sizes to optimize the tradeoff between prediction efficiency and the amount of metadata added to the bit stream.

In addition to quad-tree structure decomposition, blocks[1] can be partitioned into different shapes: a square (**S**), a horizontal rectangle (**H**) and a vertical rectangle (**V**). Figures 1.2 and 1.3 present different views of the same quad-tree structure decomposition and block partitioning example. Figure 1.2 illustrates the quad-tree structure, while fig. 1.3 shows the corresponding blocks and partitions.

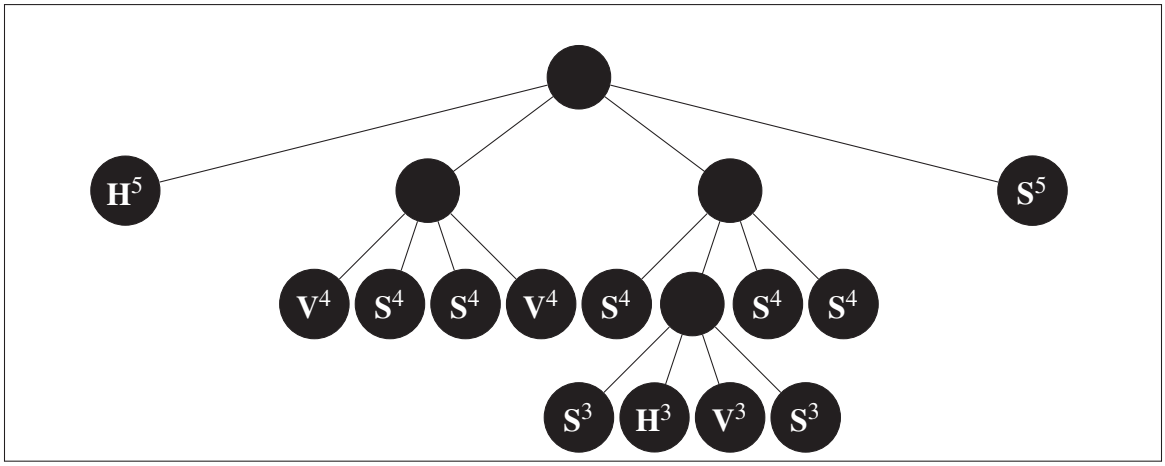


Figure 1.2 Example of a quad-tree decomposition and block partitioning. The corresponding blocks and partitions are shown in Figure 1.3

We make use of bold uppercase letters (e.g. **P**) to refer to a set of pixel coordinate pairs. Next, we denote partition primitives for a square partition ($\mathbf{S}^k$), a horizontal rectangular partition ($\mathbf{H}^k$)

[1] In modern literature, blocks are referred to by many different names: macroblocks, superblocks and partitions just to name a few. In this work, we only use the words block and partition. We distinguish a block from a partition by the fact that a block can be broken into partitions, whereas partitions may not be further partitioned.

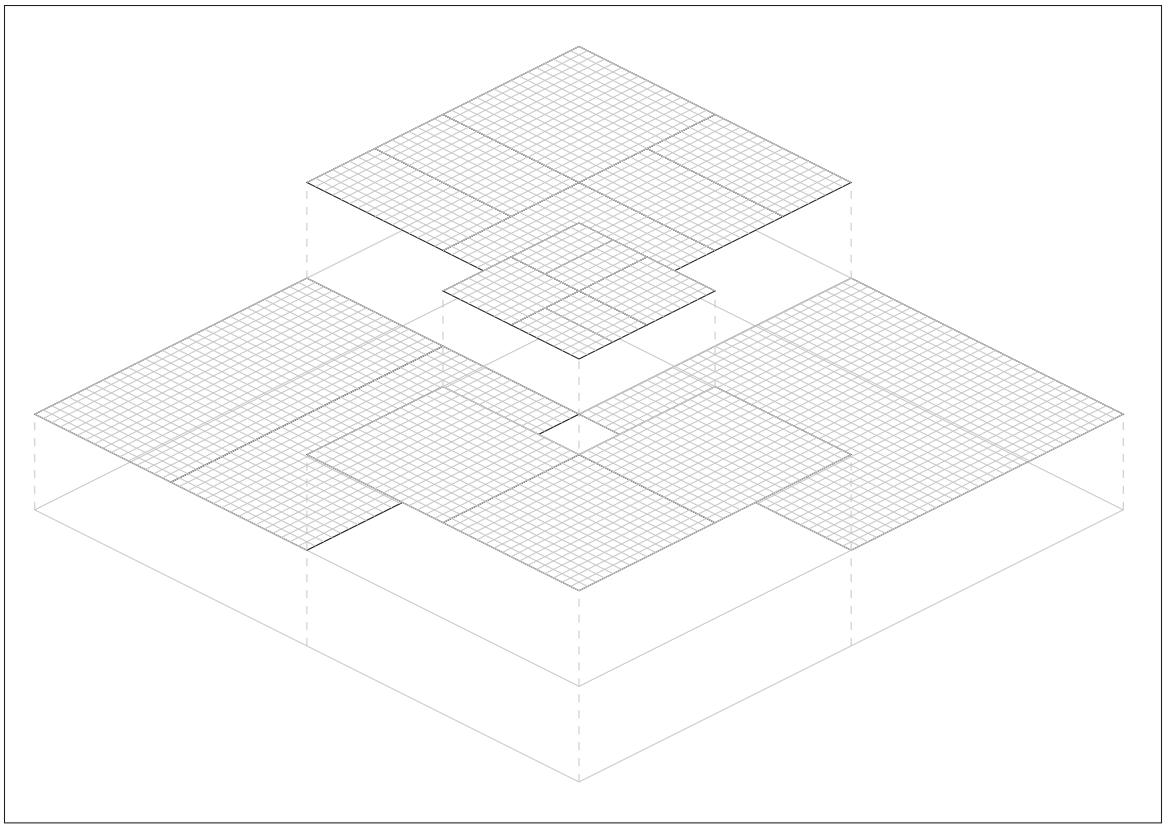


Figure 1.3 Blocks and partitions corresponding to the quad-tree decomposition and partitioning presented in Figure 1.2

and a vertical rectangular partition ($\mathbf{V}^k$):

$$\mathbf{S}^k = \left\{ (x,y) \mid 0 \leqslant x,y < 2^k \right\}, \tag{1.1}$$

$$\mathbf{H}^k = \left\{ (x,y) \mid 0 \leqslant x < 2^k, 0 \leqslant y < 2^{k-1} \right\}, \tag{1.2}$$

$$\mathbf{V}^k = \left\{ (x,y) \mid 0 \leqslant x < 2^{k-1}, 0 \leqslant y < 2^k \right\}. \tag{1.3}$$

In HEVC, block sizes range from 8×8 to 64×64. As such, we define $k \in \{3,4,5,6\}$. For example, a 8×4 block is referred to as $\mathbf{H}^3$.

In some video coding standards, like HEVC, the quad-tree structure used for predictive coding does not necessarily match the quad-tree structure used for transform coding. This enables the HEVC video coding standard to allow asymmetric partitioning for predictive coding. The

four asymmetric partitioning schemes defined by HEVC are $2N \times nU$, $2N \times nD$, $nL \times 2N$ and $nR \times 2N$ as shown in fig. 1.4. Asymmetric partitioning improves prediction efficiency when the image content inside a block follows an asymmetric pattern.

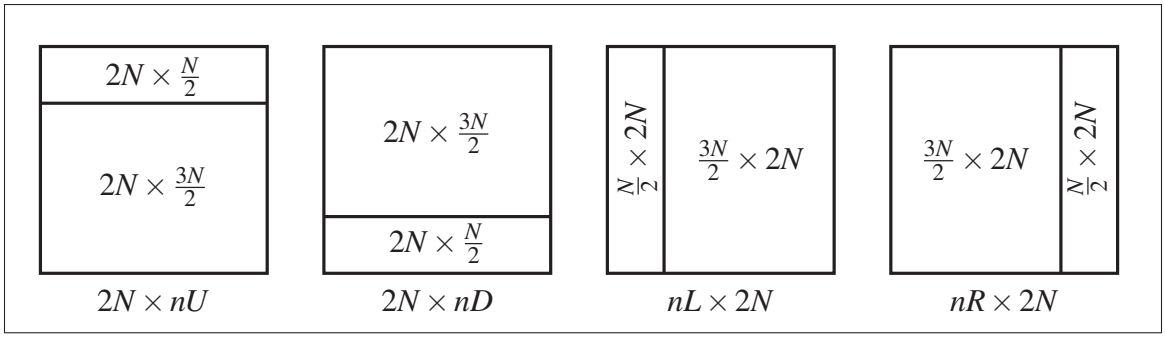


Figure 1.4 Asymmetric partitioning used in HEVC.

We refer to an arbitrary block or partition $\mathbf{P}$ at location $\mathbf{x} = (x, y)$ as $\mathbf{x} + \mathbf{P}$. For this to be valid, we define the element-wise pair addition operator[2] like so:

$$(x, y) + \mathbf{P} = \{(x + i, y + j) \mid (i, j) \in \mathbf{P}\} \ . \tag{1.4}$$

This is similar to a translation vector used for geometric transformations in the field of computer graphics (Hearn and Baker, 2004, p. 232).

We model blocks and partitions as pixel coordinate pairs rather than as pixel values directly. This is not uncommon in the field of video processing; although the proposed notation is novel, it is strongly inspired by that of Wang *et al.* (2001).

In this chapter, we presented the three distinct stages found in all block-based hybrid video encoders (i.e. predictive coding, transform coding and entropy coding). Additionally, we introduced a basic notation to represent the quad-tree-based structure and partitioning scheme found in HEVC. This notation uses pixel coordinate pairs which simplifies equations that

[2]Similar to conventional addition, element-wise pair addition is associative and commutative.

operate over blocks or partitions of different shapes and sizes as will be apparent in the next chapter.

CHAPTER 2

TEMPORAL PREDICTION

In the previous chapter, we presented the three stages of a block-based hybrid video encoder: predictive coding, transform coding and entropy coding. Our contributions only apply to the predictive coding stage. More precisely, they apply to temporal prediction. As such, this chapter starts by giving a more detailed description of ME and MC. Afterwards, we give an overview of modern techniques related to temporal prediction, such as: fractional accuracy (section 2.2), bidirectional prediction (section 2.3), motion vector prediction (section 2.4) and rate-constrained motion estimation (section 2.5).

2.1 Motion Estimation and Motion Compensation

ME and MC form the core concepts of temporal prediction. As illustrated in figures 2.1 and 2.2, ME is the process of finding the temporal prediction based on the content found in a previously encoded frame. On the other hand, MC is the process of producing the temporal prediction (i.e., the motion compensated prediction). Only the residual and the metadata, required to reproduce the prediction, are used in subsequent phases.

The pixel value at location $\mathbf{x}$ in the current frame (ψ) is accessed via $\psi(\mathbf{x})$. A previously encoded frame (ψ') may be before or after the current frame in display order. Similarly to the current frame, the value of a pixel at location $\mathbf{x}$ in a previously coded frame is accessed via $\psi'(\mathbf{x})$.

The ME algorithm searches the search area for a candidate block that minimizes a matching criterion. Let $\mathbf{C}^r$ be a set of pixel coordinate pairs of the candidate blocks in a search area or radius (r), such that:

$$\mathbf{C}^r = \{(x,y) \mid -r \leqslant x,y \leqslant r\} \ . \tag{2.1}$$

Many encoders, like the HM reference encoder implementation (McCann *et al.*, 2014), use a search area with a radius of 64 pixels. As such, in this work, we assume $r = 64$.

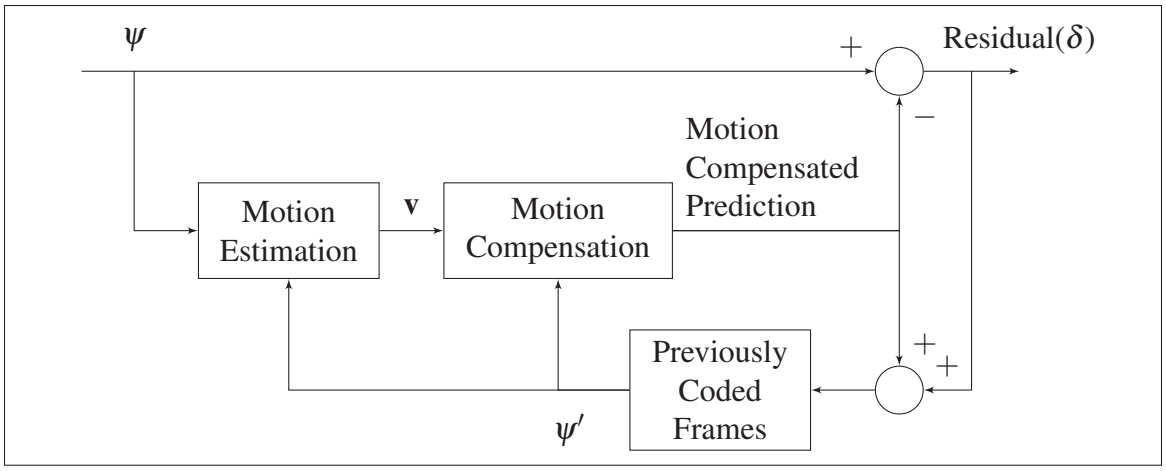


Figure 2.1 Generic outline of temporal prediction in a block-based hybrid video encoder

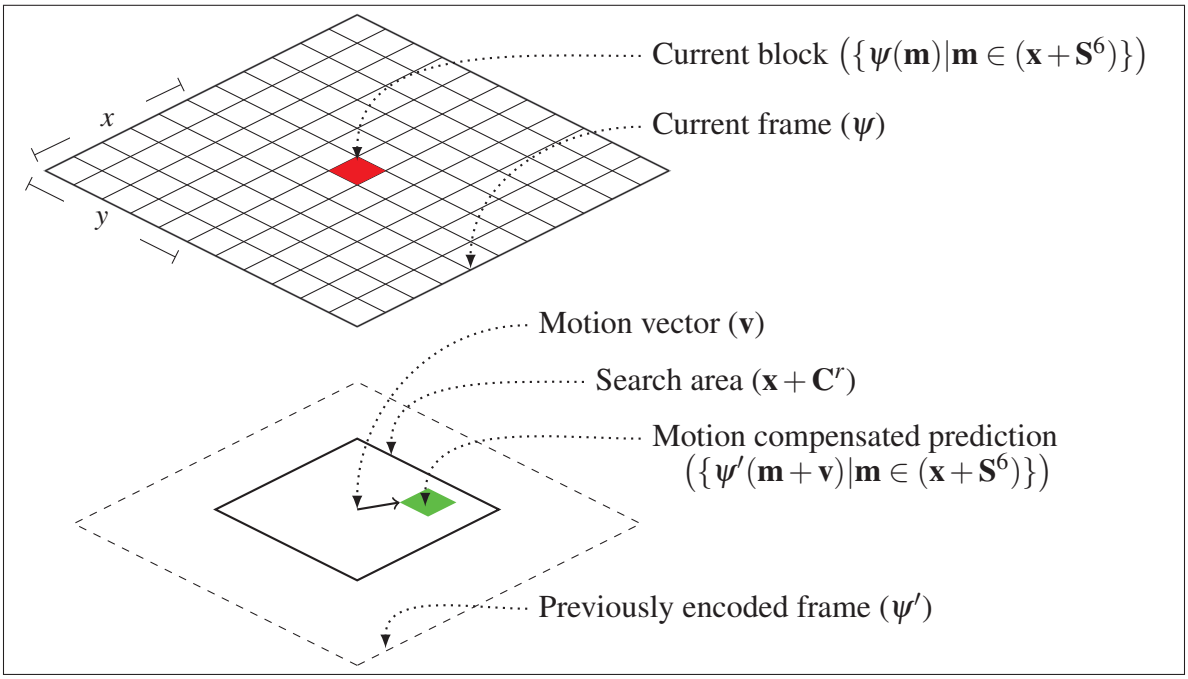


Figure 2.2 Elements and notations of motion estimation and motion compensation

Following the search, the ME algorithm returns the motion vector (MV) ($\mathbf{v}$) pointing to the chosen candidate block in the search area. The shape and size of the candidate block and the current block are identical.

The motion compensated prediction is the pixel values of the candidate block in a previously encoded frame. The pixel values of the 64×64 candidate block at position $\mathbf{v}$ in the search area are accessed as $\{\psi'(\mathbf{m}+\mathbf{v}) | \mathbf{m} \in (\mathbf{x}+\mathbf{S}^6)\}$.

The residual frame (δ) is the subtraction of the chosen predictions from the current frame. Let $\mathscr{D}$ be the set of pairs of block shapes and motion vectors representing the metadata related to the decision made by the encoder to predict the current frame. Then

$$\forall(\mathbf{P},\mathbf{v}) \in \mathscr{D}, \forall \mathbf{m} \in \mathbf{P}, \quad \delta(\mathbf{m}) \equiv \psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{v}) \,. \tag{2.2}$$

Only the δ and the block metadata are sent to subsequent phases of the encoder.

As for the matching criterion, it is not specified in the video encoding standard. However, for integer-level motion estimation, modern encoders (Lim *et al.* (2006); McCann *et al.* (2014)) widely use the SAD:

$$\mathrm{SAD}(\mathbf{P},\mathbf{v}) = \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{v})| \,. \tag{2.3}$$

2.2 Fractional Accuracy Motion Estimation and Motion Compensation

It has been widely accepted that fractional pixel accuracy can significantly improve temporal prediction (Girod, 1993, p. 139). In Figure 2.3, we illustrate the example of a half-pixel accuracy motion vector in a grid of quarter-pixel accuracy samples.

To provide fractional accuracy MVs, modern video encoding standards use fixed point number representation. We define quarter-integers $\frac{1}{4}\mathbb{Z}$ to be the group of all integers and quarter-integers $(0, \pm\frac{1}{4}, \pm\frac{1}{2}, \pm\frac{3}{4}, \pm 1, \ldots)$ (Turaev, 2010, p. 390).

Since fractional pixel information is not available to the encoder, it is approximated by interpolating from integer pixel accuracy values. As the interpolated values must match between encoder and decoder, the interpolation procedure is defined by video compression standards (Wiegand *et al.*, 2003, p. 569), (Sullivan *et al.*, 2012, p. 1659).

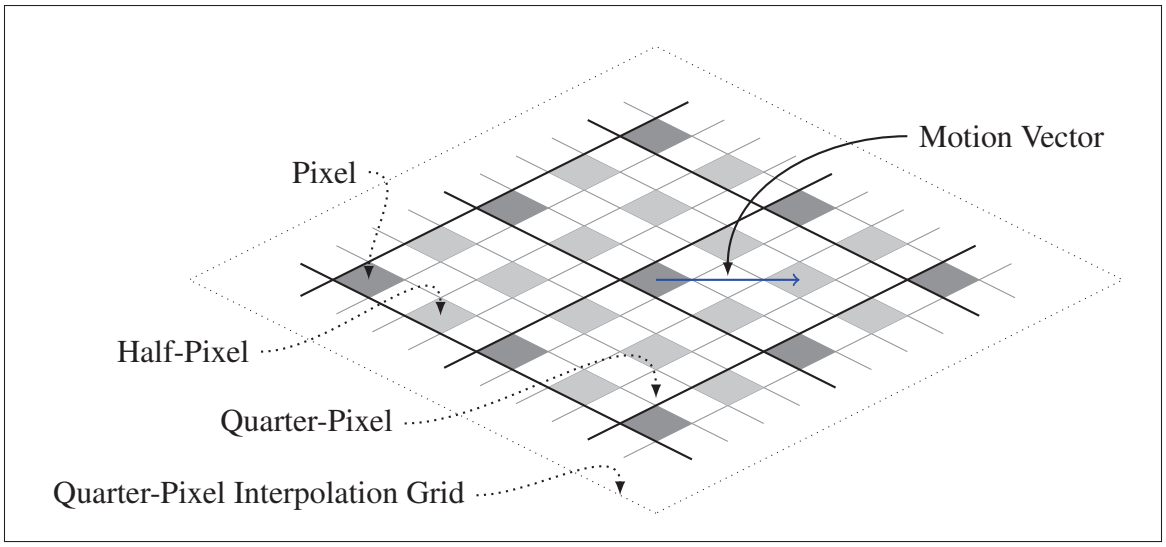


Figure 2.3 Example of a half-pixel motion vector in a quarter-pixel accuracy grid of samples. Half-pixel and quarter-pixel samples are interpolated from integer pixel values

In general, a $\frac{1}{K}$ accuracy will require a factor of K interpolation (Wang *et al.*, 2001, p. 157); this is prohibitively expensive for modern encoders. As shown in fig. 2.4, modern encoders perform ME in two steps: A first search with integer pixel accuracy, followed by a fractional pixel accuracy search over a small search area, centered at the best match found at the previous step (McCann *et al.*, 2014, p. 45).

Our work only focuses on the integer level, for two reasons. First, fractional-level block-matching operations account for only a small percentage of block-matching operations. This is mainly due to the high computational cost required to interpolate the sub-pixel values. Second, modern encoders rely on the sum of the absolute transformed differences (SATD) to evaluate candidates at the fractional level, whereas our work is based on the SAD.

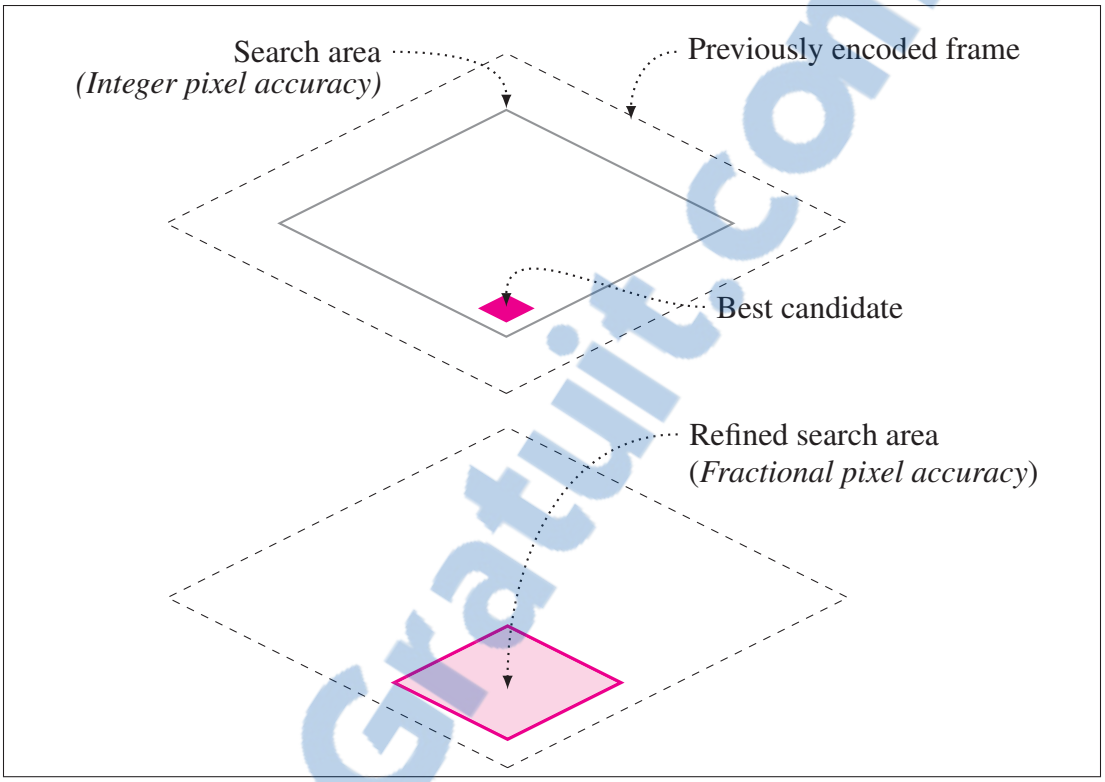


Figure 2.4 Two-step approach used in modern encoders for fractional pixel accuracy search

2.3 Bidirectional Temporal Prediction

Another significant improvement to temporal prediction is bidirectional prediction. Bidirectional prediction consists of building a prediction from two motion vectors. Each pixel in the prediction is calculated as an average, or weighted average, from the pixel values referenced by both MVs.

Bidirectional prediction requires encoding the frames in a different order from the display order. At least one frame must be coded using unidirectional prediction, the remaining frames can be coded using bidirectional prediction.

In principle, a bidirectional predictive motion search algorithm could minimize the prediction error by searching for both motion vectors simultaneously. Because of the combinatorial aspect of this problem, modern encoders like the HM resort to a greedy approach that uses an iterative

uni-predictive search for the first motion vector, followed by a refined search for the second motion vector (McCann *et al.*, 2014).

2.4 Predictive Motion Vector Coding

The signaling required for approaches like the ones presented so far (i.e. smaller block sizes, symmetric and asymmetric partitioning schemes and bidirectional prediction) yields a myriad of highly correlated temporal prediction metadata. For instance, it is a known fact that motion vectors are highly correlated both spatially and temporally. This correlation is inherent to natural images as abrupt changes, either in space or in time, are infrequent.

In order to improve compression efficiency of the temporal prediction metadata, modern video encoding standards use predictive coding on the temporal prediction metadata itself. Motion vectors are differentially coded from a motion vector prediction. A motion vector prediction is based on nearby (i.e. spatially or temporally) previously coded motion vectors.

The motion vectors used for motion vector prediction vary on the video coding standard and the availability of nearby vectors. The motion vector prediction algorithm is specified in the video coding standard, as the prediction must be identical between the encoder and the decoder.

For example, AVC uses an element-wise median of the motion vectors of three spatial neighboring blocks. As shown in fig. 2.5, these blocks are immediate neighbours, located: to the left, above and the above and to the right (Richardson, 2010, p. 159).

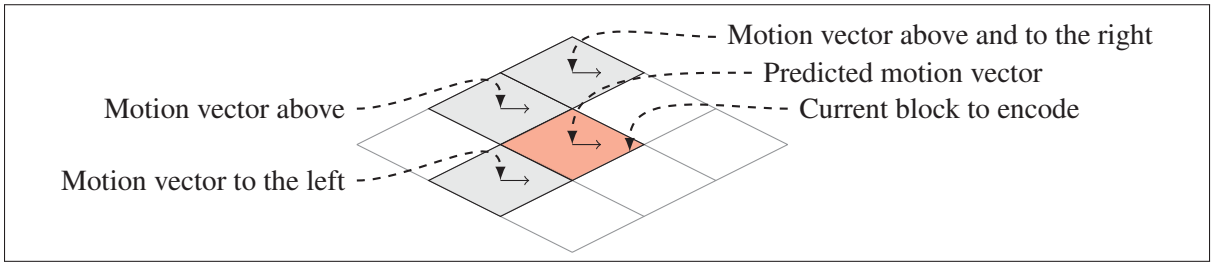


Figure 2.5 In AVC, the predicted motion vector is the median of the motion vectors: to the left, above and above and to the right of the current block

In contrast, HEVC signals the predicted motion vector from a list of candidates. This list is made up of up to two spatial candidates ($a \in \{A_0, A_1\}, b \in \{B_0, B_1, B_2\}$) and one temporal candidate ($c \in \{C_0, C_1\}$), as can be seen in fig. 2.6. The candidate C_0 represents the motion vector of co-located block in a previously encoded frame. In all cases, the first available candidate is chosen. In any case, if no candidate is available, the zero motion vector is used.

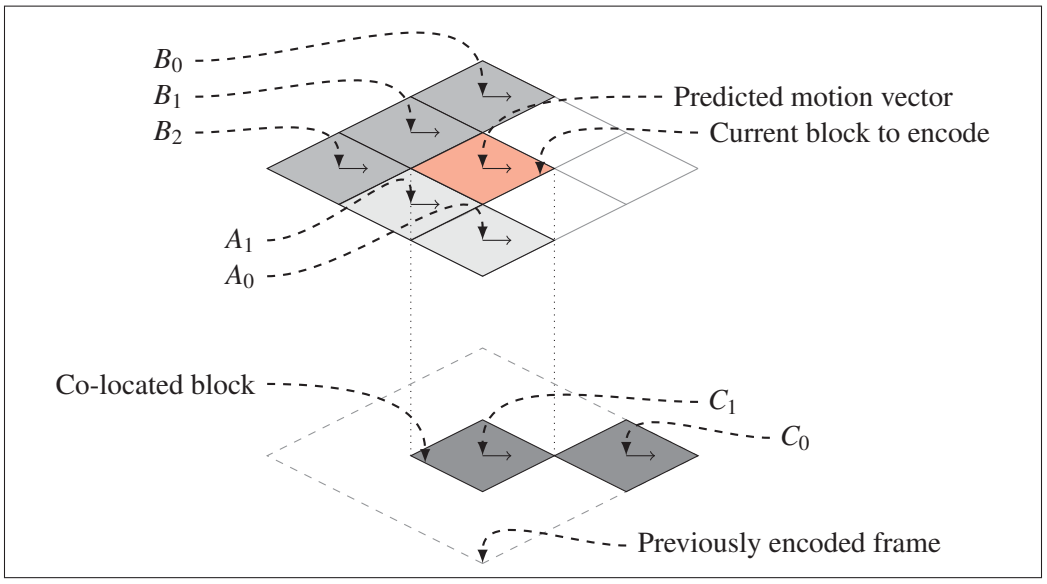


Figure 2.6 In HEVC, the predicted motion vector is signaled from a list of candidates. This list contains up to two spatial candidates ($a \in \{A_0, A_1\}, b \in \{B_0, B_1, B_2\}$) and one temporal candidate ($c \in \{C_0, C_1\}$)

2.5 Rate-Constrained Motion Estimation

As discussed earlier, the bit stream of an encoded video sequence will contain residual coefficients and metadata describing the prediction parameters. In many applications, like video-conferencing or streaming, it is desirable to limit the rate that the encoder can use to encode the video. When this happens, the residual coefficients and metadata now compete for the available bits.

When bit rates are high, this is not problematic. However, when bit rates are low, the competition between the two can become so important that it can have an impact on the visual

quality of the encoded sequence. For low bit rate scenarios, an unconstrained motion estimation algorithm will make decisions that require so much metadata that there will not be enough left for the residual. When this happens, the encoder will be forced to increase quantization to reduce the number of bits required to encode the residual.

As such, that error criterion used for motion estimation must not only take into consideration distortion, but must also take into consideration the number of bits required by the metadata. The most popular solution to this problem is to add an empirical weighing coefficient (λ) in the error criterion like so

$$\mathrm{J}(\mathbf{P},\mathbf{v}) = \mathrm{SAD}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v})\,. \tag{2.4}$$

In the previous equation, the function $\mathrm{R}(\mathbf{v})$ returns an approximation of the number of bits required to encode the motion vector related to the candidate at position $\mathbf{v}$ relative to $\mathbf{P}$.

It is important to understand that the weighing coefficient λ models the trade-off between the motion vector rate and the energy present in the SAD distortion. Minimizing this criterion in no way guarantees to minimize the rate-distortion ratio. One must not be confused by the terminology, there is an important difference between rate-distortion and rate-constrained.

Rate-distortion Authors refer to the rate-distortion ratio as the total rate of the block (metadata + residual) over the squared error after quantization.

Rate-constrained

When referring to rate-constrained motion estimation, authors imply that only the vector rate is considered. The motion vector cost is weighted and added to the SAD of the residual before quantization.

The exact definition of λ varies based on the encoder implementation. However, modern standards often recommend a value for λ. For example, the recommended value of λ for

the H.264 [1] video coding standard is

$$\lambda = 0.85 \times 2^{\frac{QP-12}{3}} . \tag{2.5}$$

The reason why the standard comity recommends a certain value is that rate and distortion are design consideration of the quantization step size (Q_{step}) (Sullivan and Wiegand, 1998). The authors further explain that Eq. (2.5) is an approximation of the assumed functional relationship between the quantization parameter (QP) and λ up to a QP of about 25. This relationship "may not be completely realistic, the derivation reveals at least the qualitative insight that it may be reasonable" (Sullivan and Wiegand, 1998).

We access the x and y elements of vectors $\mathbf{v}$ and $\mathbf{p}$ via subscripts. This allows to define the R function, used in Eq. (2.4), as follows:

$$R(\mathbf{v}) = G(4(\mathbf{v}_x - \mathbf{p}_x)) + G(4(\mathbf{v}_y - \mathbf{p}_y)) , \tag{2.6}$$

where $G(v)$ returns the length of the signed exponential Golomb code required to encode $\mathbf{v}$, and $\mathbf{p} \in \frac{1}{4}\mathbb{Z}^2$ is the motion vector predictor (MVP). The differential between the MV and the MVP is multiplied by four to obtain integer values conforming to the H.264 and the HEVC standards, which encode MVs using quarter pixel precision specified with fixed point notation.

Signed exponential Golomb codes are used in the H.264 standard to encode the differences between MVs and the MVP. Although signed exponential Golomb codes are not used by the HEVC standard to encode MV information, they are the recommended metric used to quickly measure MV costs in rate-constrained block-matching algorithm (RCBMA) for HEVC (McCann *et al.*, 2014).

[1] We acknowledge that multiple values of λ have been recommended for H.264, as explained in (Takagi *et al.*, 2003); However for this example, we use the value presented in (Richardson, 2010, p.282), which appears to be the most popular. Also note that there is an error in the equation presented in (Richardson, 2010, p.282), we present the correct version.

As described in (Richardson, 2010), exponential Golomb codes are composed of a prefix part, a marker bit and an info part. The total length of an exponential Golomb code for an integer value v is measured with the following function:

$$G(v) = 2 \times I(v) + 1\,. \tag{2.7}$$

The $I()$ function returns the length of the info part which is multiplied by 2, because the prefix is the same length as the info. The added one represents the marker bit. The $I()$ function is defined as follows:

$$I(v) = \lfloor \log_2(2|v|+1) \rfloor\,, \tag{2.8}$$

where $|v|$ is multiplied by 2, because the code words used in the info part alternate between negative and positive values. One is added to represent that the value 0 is assigned to the code word '0'. The $\lfloor\,\rfloor$ operator represents the floor function.

We started this chapter by describing ME and MC, the core concepts behind temporal predictions. We continued it by describing modern techniques related to temporal prediction. Some of these techniques are specified by the video coding standard, while others are not required but are strongly recommended and are ubiquitous in modern encoders. The presented techniques are important because they have an impact on successive elimination algorithms, which is the subject of the next chapter.

CHAPTER 3

LITERATURE REVIEW ON SUCCESSIVE ELIMINATION

As their name suggests, SEAs eliminate candidate blocks during ME. However, SEAs do not alter the outcome of the ME search algorithm, as they only eliminate candidates that cannot produce better results than the current best cost. The SEA (Li and Salari, 1995) and its derivatives: RCSEA (Coban and Mersereau, 1998), MSEA (Gao *et al.*, 2000) and FGSE (Zhu *et al.*, 2005) all share two common concepts:

Transitive elimination

A lower bound approximation of the error metric is used. When the approximated error of a candidate is higher than the current smallest error metric, the error metric does not need to be computed for that candidate, as it is greater than or equal to the approximation.

Fast summation

The lower bound approximation is precomputed using a fast algorithm. As such, after being precomputed, the lower bound approximation only requires two look up operations, thus considerably less than the error metric.

In this chapter, we start by presenting the inner workings of transitive elimination and fast summation. Afterwards, we describe MSEA and RCSEA.

3.1 Transitive Elimination

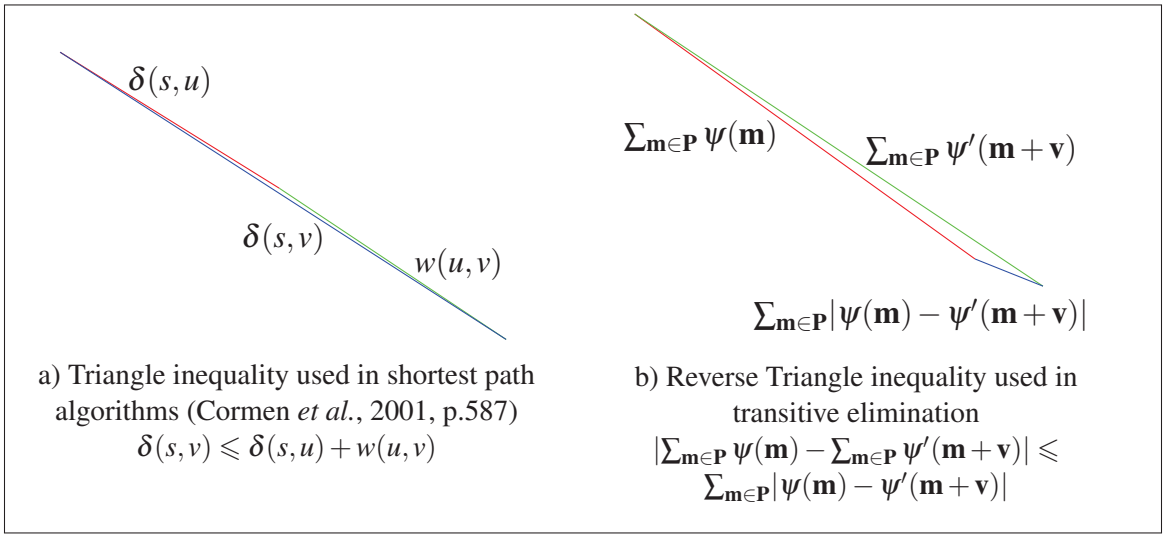
We refer to the elimination phase of SEA as transitive elimination. This descriptive name highlights that transitivity is used to perform elimination. This being said, transitive elimination is similar to the more commonly known concept of relaxation. As explained in (Cormen *et al.*, 2001, p.585): "The process of **relaxing** an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u[...]". In the previous quote,

Cormen *et al.* are describing relaxation with regards to shortest path algorithms, but the same is true of motion estimation.

We can describe transitive elimination by paraphrasing (Cormen *et al.*, 2001, p.585) in the context of ME: transitive elimination consists of testing whether computing the SAD of a candidate block can improve the current best cost found so far.

As shown in fig. 3.1, relaxation in the context of shortest path algorithms relies on an upper-bound and the triangle inequality ($\delta(s,v) \leqslant \delta(s,u) + w(u,v)$) (Cormen *et al.*, 2001, p.587); whereas transitive elimination relies on a lower-bound and the reverse triangle inequality. The proposed lower-bound is the absolute difference of sums (ADS), such that:

$$\left| \sum_{\mathbf{m}\in\mathbf{P}} \psi(\mathbf{m}) - \sum_{\mathbf{m}\in\mathbf{P}} \psi'(\mathbf{m}+\mathbf{v}) \right| \leqslant \sum_{\mathbf{m}\in\mathbf{P}} \left| \psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{v}) \right|, \quad \forall \mathbf{v} \in \mathbf{C}^r \,. \tag{3.1}$$



a) Triangle inequality used in shortest path algorithms (Cormen *et al.*, 2001, p.587)
$\delta(s,v) \leqslant \delta(s,u) + w(u,v)$

b) Reverse Triangle inequality used in transitive elimination
$|\sum_{\mathbf{m}\in\mathbf{P}} \psi(\mathbf{m}) - \sum_{\mathbf{m}\in\mathbf{P}} \psi'(\mathbf{m}+\mathbf{v})| \leqslant \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{v})|$

Figure 3.1 Geometric example of the triangle inequality and the reverse triangle inequality with regards to shortest path algorithms and transitive elimination

Let $\mathbf{F} \subseteq \mathbf{C}^r$ be the subset of candidates, evaluated so far, from a search area of size r (as defined in Eq. (2.1)) in the ME search. We define $\hat{\mathbf{v}}$ as the motion vector pointing to the current best

candidate, such that:

$$\hat{\mathbf{v}} \in \arg\min_{\mathbf{c}\in\mathbf{F}} \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{c})| \,. \tag{3.2}$$

Transitive elimination works as follows, a candidate with an ADS greater than the SAD of the current best candidate is eliminated. By transitivity, the SAD of this candidate will also be greater than or equal to the SAD of the current best found so far:

$$|\sum_{\mathbf{m}\in\mathbf{P}} \psi(\mathbf{m}) - \sum_{\mathbf{m}\in\mathbf{P}} \psi'(\mathbf{m}+\mathbf{v})| > \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\hat{\mathbf{v}})| \tag{3.3}$$

$$\implies \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{v})| > \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\hat{\mathbf{v}})| \tag{3.4}$$

$$\implies \mathbf{v} \notin \arg\min_{\mathbf{c}\in\{\mathbf{F},\mathbf{v}\}} \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{c})| \tag{3.5}$$

$$\implies \mathbf{v} \notin \arg\min_{\mathbf{c}\in\mathbf{C}} \sum_{\mathbf{m}\in\mathbf{P}} |\psi(\mathbf{m}) - \psi'(\mathbf{m}+\mathbf{c})| \tag{3.6}$$

3.2 Fast Summation

To emphasize that the sums of the ADS can be precomputed, we define the function as follows:

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) = |\mathrm{PS}(\mathbf{P}) - \mathrm{RPS}(\mathbf{P},\mathbf{v})| \,. \tag{3.7}$$

The function $\mathrm{PS}(\mathbf{P})$ return the sum of the pixels of $\mathbf{P}$ in the current frame

$$\mathrm{PS}(\mathbf{P}) = \sum_{\mathbf{m}\in\mathbf{P}} \psi(\mathbf{m}) \,, \tag{3.8}$$

whereas the function $\mathrm{RPS}(\mathbf{P},\mathbf{v})$ returns the sum of the pixels in $\mathbf{P}$, but with an offset of $\mathbf{v}$, in a previously encoded frame:

$$\mathrm{RPS}(\mathbf{P},\mathbf{v}) = \sum_{\mathbf{m}\in\mathbf{P}} \psi'(\mathbf{m}+\mathbf{v}) \,. \tag{3.9}$$

Precomputing RPS($\mathbf{P}, \mathbf{v}$) is considerably more complex than precomputing PS($\mathbf{P}$), as every overlapping block must be computed. Li and Salari (1995) propose a fast summation technique based on a two-pass sliding window approach to precompute RPS($\mathbf{P}, \mathbf{v}$). An example of the first and second passes are given in figures 3.2 and 3.3 respectively. In this example, the first two 4×4 blocks of a frame are summed.

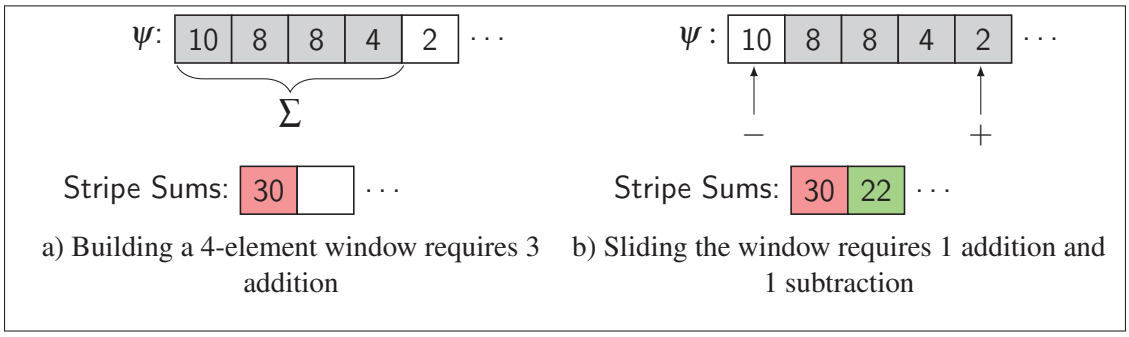


Figure 3.2 Example of the first pass of the sliding window approach proposed by Li and Salari. This data will be reused in fig. 3.3 to sum 4×4 blocks

The first pass computes stripes, which represent the sum of the rows of each block over the entire frame. At the beginning of the row, the window must be built. For an n-element window, $n-1$ additions are required, in order to sum the n first elements of the row. This sum represents the first stripe, as shown in fig. 3.2a. Afterwards, the sum of the next stripe is obtained by sliding the window forward. This procedure, demonstrated in fig. 3.2b, requires that the first element be removed from the window by subtracting it from the sum of the window and that the next element is added to the window by adding it to the sum of the window. The value of the second strip is the current sum of the window. The remaining stripes are computed by moving the window forward until the end of the row. This process is repeated for all rows of the image.

The second pass computes the block sums by adding the stripes of every column of each block over the entire frame. At the beginning of the column, the window must be built, an example is shown in fig. 3.3a. This requires adding the $n-1$ first stripes of the column which represents the sum of the first block. Subsequent block sums in this column are computed by moving

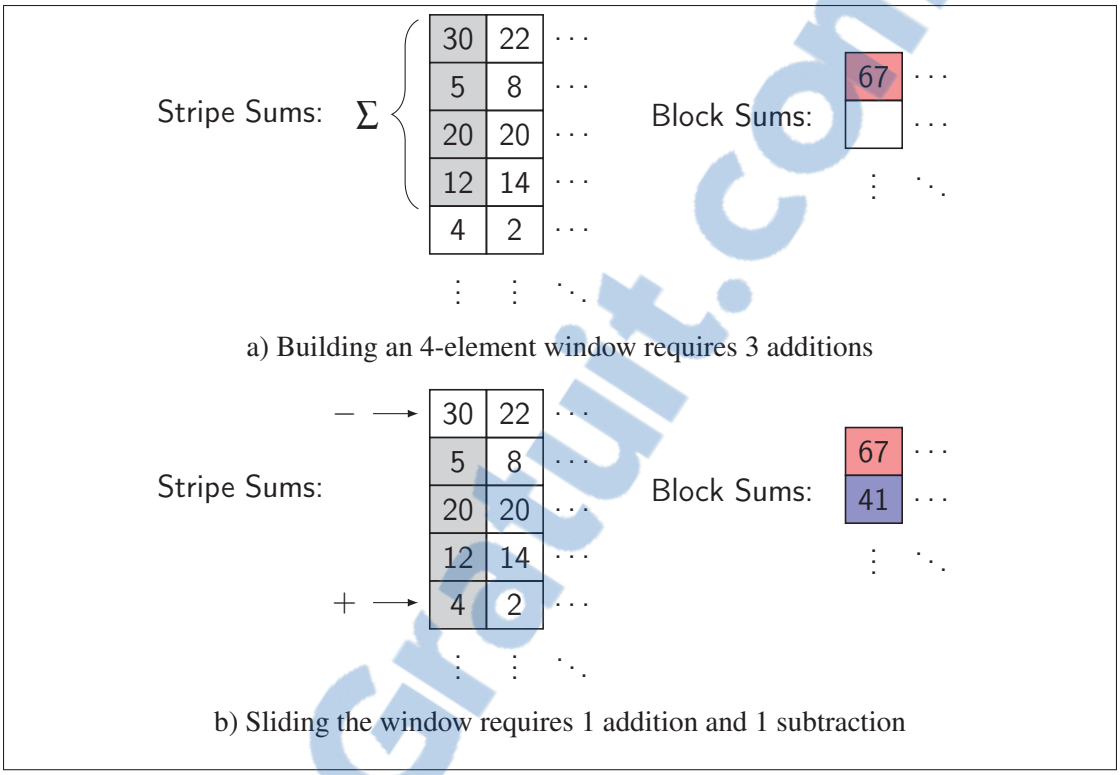


Figure 3.3 Example of the second pass of the sliding window approach proposed by Li and Salari. The stripes used where in part computed in fig. 3.2.

the window downwards, which entails subtracting the first stripe of the window and adding the stripe immediately after the window (see fig. 3.3b).

3.3 Multi-Level Successive Elimination Algorithm

As block sizes increases, so does the average distance between the SAD and the ADS of the search candidates. This considerably reduces the efficiency of transitive elimination. To address this issue, (Gao *et al.*, 2000) proposed to split a square block of size k into four square partitions of size $k-1$, like so:

$$\mathscr{S}^k = \left\{ \begin{array}{cc} (0,0)+\mathbf{S}^{k-1}, & (2^{k-1},0)+\mathbf{S}^{k-1}, \\ (0,2^{k-1})+\mathbf{S}^{k-1}, & (2^{k-1},2^{k-1})+\mathbf{S}^{k-1} \end{array} \right\}. \tag{3.10}$$

We use styled capital letters (e.g. $\mathscr{P}$) to refer to a set of sets of pixel coordinate pairs.

The multi-level ADS (MADS) is obtained by summing the ADS of each partition:

$$\mathrm{MADS}(\mathscr{P},\mathbf{v}) = \sum_{\mathbf{P}\in\mathscr{P}} \mathrm{ADS}(\mathbf{P},\mathbf{v})\,. \tag{3.11}$$

Splitting the block into partitions reduces the number of pixels per partition, thus lowering the average distance between SAD and the ADS and increasing the efficiency of transitive elimination. It follows that:

$$\mathrm{ADS}(\mathbf{S}^k,\mathbf{v}) \leqslant \mathrm{MADS}(\mathscr{S}^k,\mathbf{v}) \leqslant \mathrm{SAD}(\mathbf{S}^k,\mathbf{v})\,. \tag{3.12}$$

The FGSE (Zhu *et al.*, 2005) builds upon this and allows the set $\mathscr{S}^k$ to contain a combination of $\mathbf{S}^{k-1}$ and $\mathscr{S}^{k-1}$.

For MSEA, there is clearly a trade-off between the computational savings and the size of the partitions. Smaller partitions, on average, will improve the lower-bound and reduce SAD operations. However, blocks separated into smaller partitions require more computations to sum the ADS of each partition. Thus, the use of MSEA leads to diminishing returns in computational savings.

3.4 Rate-Constrained Successive Elimination Algorithm

As explained in section 2.5, when evaluating candidates, modern block-matching algorithms use a rate constraint, such as the cost function J defined in Lim *et al.* (2006); McCann *et al.* (2014), of which a more general form can be written as:

$$\mathrm{J}(\mathbf{P},\mathbf{v}) = \mathrm{SAD}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v})\,. \tag{3.13}$$

As described in Coban and Mersereau (1998), the rate-constraint can also be applied to the equations 3.3, 3.4, 3.5 and 3.6. Based on these equations, we can derive the same implications for the rate constrained context:

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda R(\mathbf{v}) > \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) + \lambda R(\hat{\mathbf{v}}) \tag{3.14}$$

$$\implies \mathbf{v} \notin \underset{\mathbf{c}\in\mathbf{C}}{\arg\min}\, \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda R(\mathbf{c})\,. \tag{3.15}$$

By transitivity, if the weighted ADS of a candidate is equal or greater than the current best cost, the candidate can safely be eliminated, as its cost will be greater than or equal to the current best cost.

This chapter discussed the successive elimination algorithm, its core concepts and its derivatives. Historically, SEAs have been targeted at the exhaustive search algorithm. We present the exhaustive search algorithm and other motion estimation algorithms in the next chapter.

CHAPTER 4

LITERATURE REVIEW ON MOTION ESTIMATION ALGORITHMS

In this section, we give an overview of over 30 years of research related to search algorithms used for motion estimation in the context of video coding. We start with a discussion on optimality to illustrate that finding the global minimum in a search area does not necessarily imply an exhaustive search. Next, we present 2 common assumptions made by a majority of search algorithms: monotone decreasing search area and center-biased motion vectors. Afterwards, we introduce the 2-D-logarithmic search. This algorithm is the foundation of most modern search algorithms. For the next sections, instead of presenting each search algorithm, we focus on two discriminant features of modern search algorithms: predictive search and early termination. Finally, we examine the TZ-Search algorithm, as it is the baseline for comparison for search algorithms designed for HEVC and the confidence interval-based motion estimation (CIME) as it combines the TZ-Search with a probabilistic version of the SEA.

4.1 Optimality

We classify motion estimation search algorithms as being either optimal or suboptimal. An optimal search algorithm will find the best candidate ($\mathbf{v}^*$) in the search area ($\mathbf{C}$), where

$$\mathbf{v}^* \in \underset{\mathbf{v}\in\mathbf{C}}{\arg\min} \left(\mathrm{SAD}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v})\right) . \tag{4.1}$$

It is important to note that more than one candidate can minimize the cost function. This is why "$\in$" is used over "$=$".

One way to find $\mathbf{v}^*$ is to compute the error criterion for each candidate in the search area. This is referred to as an exhaustive search or as a full search. Exhaustive search is prohibitively expensive, especially for modern encoding standards with tools like variable block sizes, symmetric and asymmetric partitioning schemes and bidirectional prediction. As

such, exhaustive search is often available as a configurable option in modern encoders but seldom used in commercial encoding contexts.

A common misconception is that an optimal search algorithm implies an exhaustive search. Optimality preserving acceleration techniques, like transitive elimination, allow for optimal but not exhaustive search algorithms. By definition, the algorithm is not exhaustive as it does not compute the error criterion for each candidate in the search area. Instead, a low complexity approximation is used in such a way that it guarantees to satisfy Eq. (4.1).

Suboptimal search algorithms, also known as fast search algorithms, will find the best candidate ($\mathbf{v}'$) in a subset of the search area ($\mathbf{C}' \subseteq \mathbf{C}$), such that:

$$\mathbf{v}' \in \arg\min_{\mathbf{v}\in\mathbf{C}'} \left(\mathrm{SAD}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v})\right) . \tag{4.2}$$

Fast algorithms distinguish themselves by the candidates they include in their search area subset. Most, if not all, fast algorithms make the assumption that the error criterion is monotone decreasing towards $\mathbf{v}^*$. This assumption does not always hold, as such

$$\mathrm{SAD}(\mathbf{P},\mathbf{v}^*) + \lambda \mathrm{R}(\mathbf{v}^*) \leqslant \mathrm{SAD}(\mathbf{P},\mathbf{v}') + \lambda \mathrm{R}(\mathbf{v}') . \tag{4.3}$$

More recent fast algorithms also take into account spatial and temporal correlation of motion vectors when building their subset.

To refer to the best motion vector independently from the type of search algorithm used, we use $\hat{\mathbf{v}}$, where $\hat{\mathbf{v}} \in \{\mathbf{v}^*, \mathbf{v}'\}$.

A surprising fact was discovered about suboptimal motion estimation algorithms. When used in conjunction with motion vector prediction, suboptimal motion estimation algorithm can outperform optimal motion estimation over the entire frame. In other words, simulations have shown that suboptimal motion estimation algorithms can produce better rate distortion ratios than optimal motion estimation, when motion vector prediction is used Tourapis *et al.* (1999).

This might seem counter-intuitive at first, but reveals a form of motion data redundancy originating from the search algorithm itself. Concretely, motion vectors of suboptimal search algorithms are more correlated than the optimal motion vectors. As such, this correlation increases the efficiency of motion vector prediction, which compensates for the difference in coding efficiency between the global minimum and the local minimum. Spatial correlation also plays a role in this, as it often entails that the difference between global minimum and the local minimum is quite small.

This phenomenon also highlights a fundamental flaw in state-of-the-art approaches. By design, current algorithms are greedy and are prone to select solutions that are optimal at the coding tree unit (CTU) level. However, the resulting combination of decisions is suboptimal at the frame level.

This accentuates the inaccuracy of the rate constraint, which is mostly due to improper use. A proper rate constraint should be applied after quantization not before it. However, performing transform coding and quantization on each candidate in the search area is exorbitantly expensive. As such, all rate-constrained search algorithms take a shortcut, they apply the rate-constraint to the SAD.

4.2 Common assumptions of search algorithms

In this section, we describe two important assumptions used by search algorithms: monotonically decreasing search areas and centered-bias motion vectors.

4.2.1 Monotonically Decreasing Search Area

The most important assumption made by fast search algorithms is that measuring the error criterion over the search area results in a surface that is monotonically decreasing towards the global minimum (Jain and Jain, 1981). An example of such a surface is given in fig. 4.1; it is apparent that it is not monotonically decreasing.

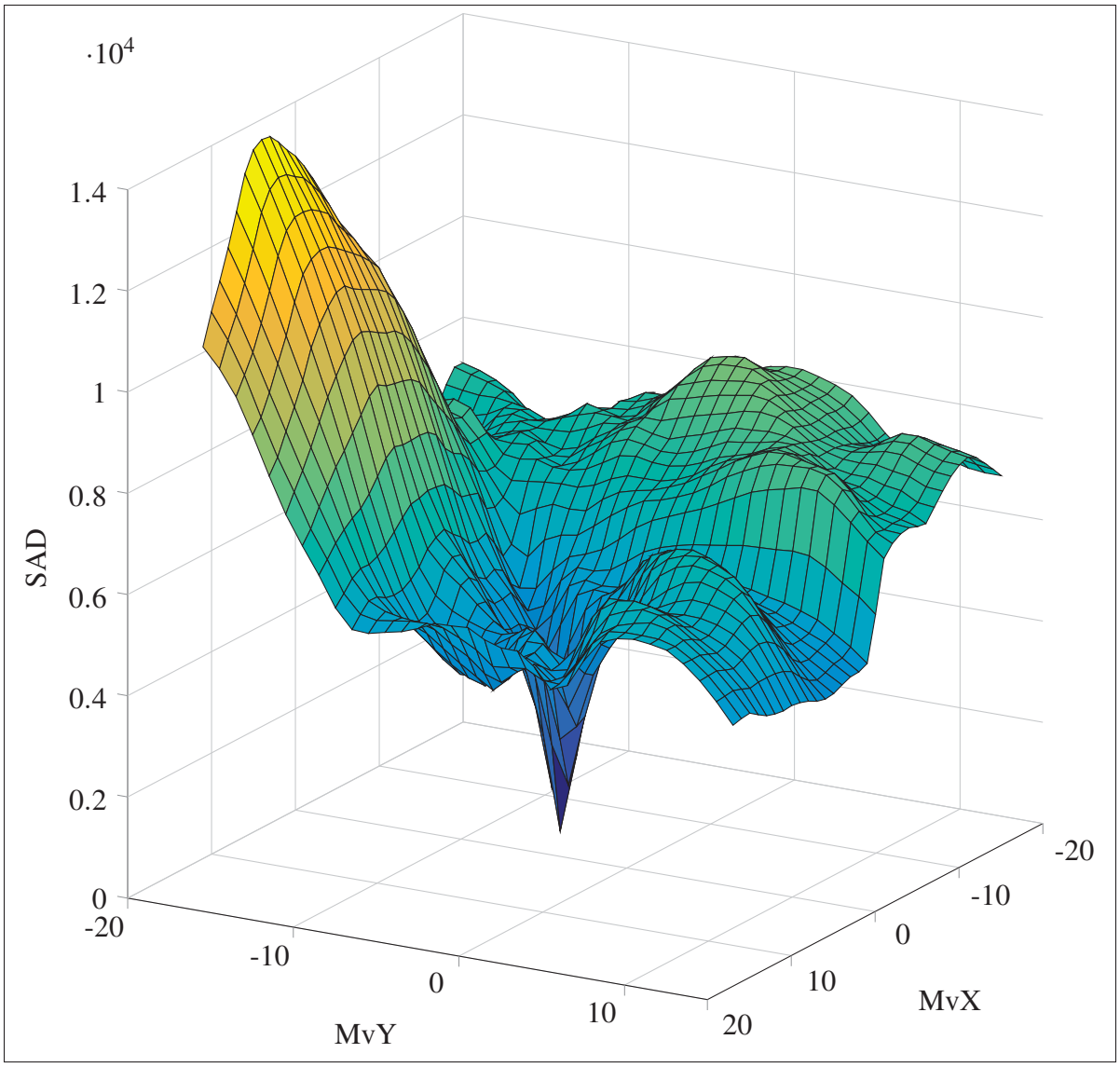


Figure 4.1 Example of the surface of the error criterion over the search area. This surface is not monotonically decreasing towards the global minimum.

However, this assumption highlights a general trend found in most search areas that the values decrease towards the global minimum. This is why so many algorithms make this assumption. The risk involved in making this assumption is getting stuck in a local minimum.

4.2.2 Center-Biased Motion Vectors

Another important assumption is that of stationary or quasi-stationary motion (Li *et al.*, 1994). This assumption is based on the temporal correlation found in natural video sequences (see (Wang *et al.*, 2001) for more information). Consequently, one of the implications of stationary or quasi-stationary motion is a strong center-bias in global minimums (He and Liou, 1997). As shown in fig. 4.2, global minimums are not uniformly distributed over the search area, they are almost all located near the center (Li *et al.*, 1994). Moreover, as the search moves away from the center of the search area, the likelihood of finding the optimal candidate decreases (Tourapis *et al.*, 1999).

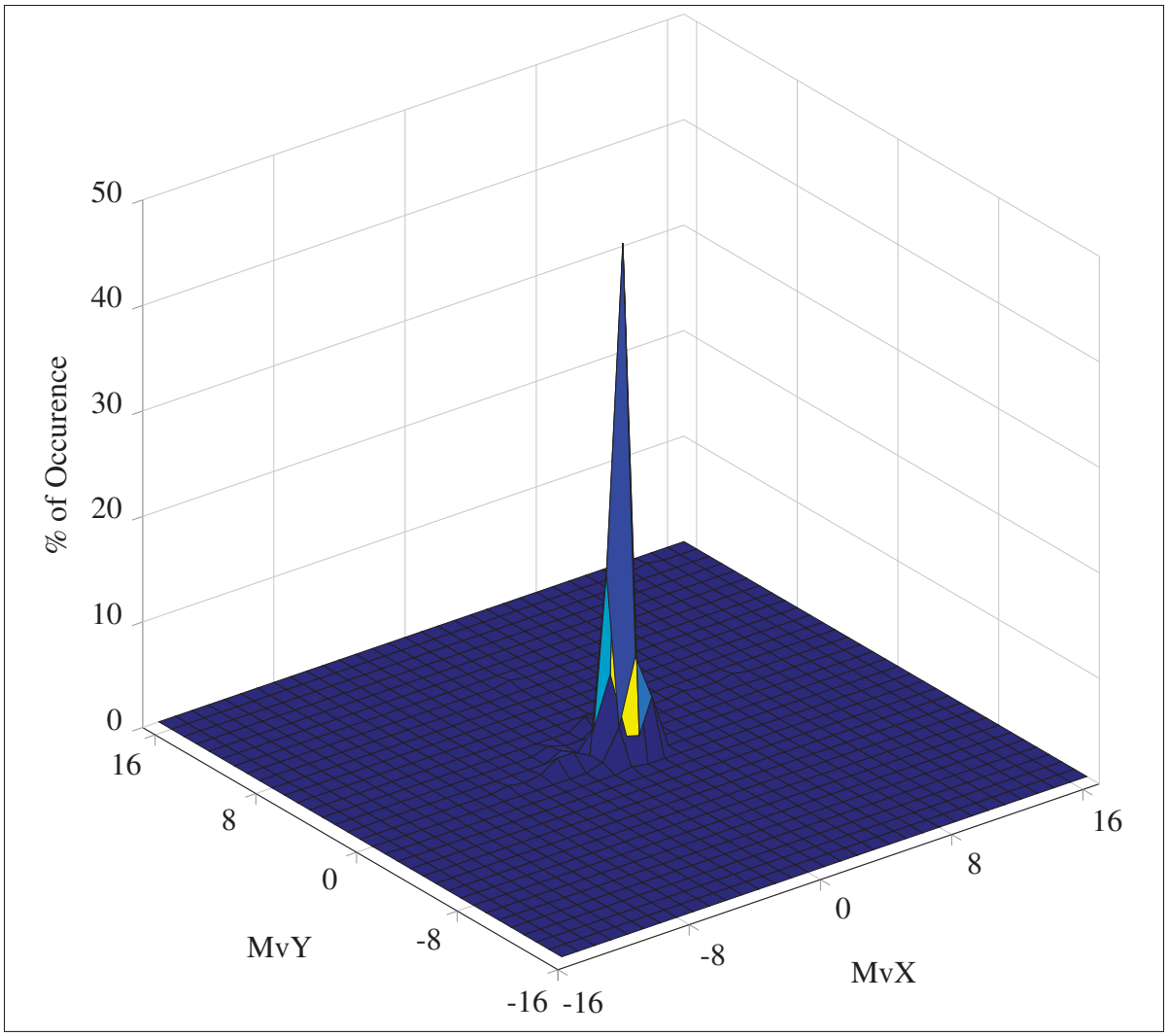


Figure 4.2 Example of the distribution of the position of the global minimum inside the search area for a quasi-stationary frame. Global minimums are distributed at the center of the search area.

4.3 2-D-Logarithmic Search

As its name suggests, the 2-D-Logarithmic search is a 2 dimensional extension of the logarithmic search (more commonly known as a binary search) (Jain and Jain, 1981). The

algorithm can use a geometric pattern like a square, a cross, a diamond[1,2] or an hexagon. In some cases, the algorithm is referred to by the name of the geometric pattern used. For example, some might refer to fig. 4.3 as an illustration of the hexagon search algorithm.

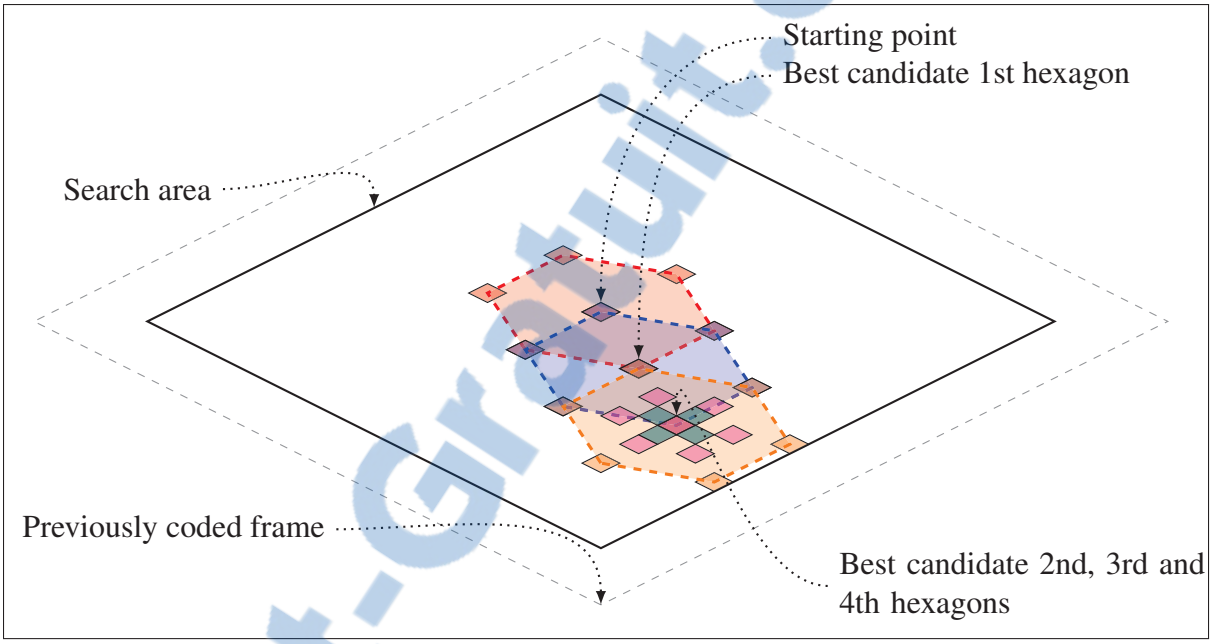


Figure 4.3 Example of a 2-D-logarithmic search algorithm using a combination of an hexagon and a cross pattern. First, the hexagon is used until it converges, then a cross is used as a refinement.

The algorithm works as follows:

- Starting from the center of the search area, the error criterion is computed for the candidates in the center and at the edges of a certain geometric shape. In fig. 4.3, the error criterion is evaluated for each red block. As can be seen, these blocks form an hexagon shaped surface.

[1]Not to be confused with unrestricted center-biased diamond search (UCBDS) (Tham *et al.*, 1998). Because of its popularity UCBDS is also often referred to as diamond search. The noticeable difference is that UCBDS uses a 3 pixel sided diamond shape. The authors argue that a compact and fixed sized diamond is center-bias.

[2]Not to also be confused with the new diamond search (Zhu and Ma, 2000). The novelty in the new diamond search arises from the use of large diamond and a complementary small diamond refinement.

- The candidate with the lowest error becomes the center of the geometric pattern. The error criterion is evaluated on candidates that have not been evaluated so far. The blue hexagon, in fig. 4.3, represents this step. Blocks that are blue and red are not evaluated twice.
- This process continues until the center of the geometric pattern remains unchanged. This is hard to illustrate in fig. 4.3, but the 2nd and 3rd best candidates are identical.
- The size of the geometric pattern is reduced and the loop resumes until the center remains unchanged. In fig. 4.3, the pink hexagon is smaller than the previous hexagon and it fails to find a better candidate.
- The pattern is reduced until it reaches its minimal size. Figure 4.3 illustrates a popular variant that uses an alternate ending, where another complementary arrangement is used as a final refinement. A cross is often used as it complements shapes like squares, diamonds and hexagons. Another implementation of this popular variant is the small diamond step in (Zhu and Ma, 2000).

We described a particular variant, but there are numerous variants as this topic has undergone research efforts for more than 30 years. Some variations use asymmetric (also referred to as unsymmetric) shapes. Usually, asymmetric shapes will be considerably wider. The reason behind this is the: “common agreed conclusion that movement in the horizontal direction is much heavier than that in the vertical direction for natural picture sequences” (Chen *et al.*, 2006).

The three step search (TSS) algorithm was published at almost the same time as the 2-D-Logarithmic search algorithm and both algorithms are closely related (Musmann *et al.*, 1985). The TSS differs in that the number of steps is fixed to three, as such the candidates are more coarsely spaced and the size of the pattern is reduced by half at each step.

Quite interestingly, in the 90s, the TSS was the most popular (Li *et al.*, 1994). This early success came from the number of steps being fixed that translated into a constant number of operations, which is a significant advantage for hardware implementations, the predominant

implementation for encoders of that era (Wang *et al.*, 2001). However, in the current era, technological advancements favor software encoders. Consequently, this fixed number of operations is not a significant advantage anymore. Moreover, the increase of video resolutions has lead to an increase in search area sizes, which is a disadvantage for the TSS.

It is important to understand that the 2-D-Logarithmic search and the TSS where not originally center-biased. Since then, many variations have been proposed to make them center-biased. Notable mentions are the new tree step search (Li *et al.*, 1994) and new diamond search (Zhu and Ma, 2000).

Furthermore, it is important to note that the binary search algorithm only works on sorted arrays. Consequently, the 2-D-Logarithmic search and the TSS both assume a monotonically decreasing search area (Musmann *et al.*, 1985). There is no guarantee of this, and when it is not the case, the algorithm converges to a local minimum. This is the problem researchers have been trying to resolve ever since.

4.4 Predictive Search

We refer to predictive search algorithms as algorithms which use predictors to determine the starting point of a 2-D-logarithmic search. First referred to as search centers by (Hosur and Ma, 1999), predictors are derived from the motion vectors of adjacent blocks either spatially or temporally.

(Tourapis, 2002) states that predictor selection appears to be the most important feature and the key to the performance of modern search algorithms. Predictive search algorithms evaluate the error criterion over a set of predictors. The candidate that minimizes this error is chosen as the starting point.

Early predictive algorithms used few candidates. For example, the motion vector field adaptive search technique (MVFAST) (Hosur and Ma, 1999) used three spatial neighboring blocks immediately located: to the left, above and the above and to the right. Because of the

correlation present between motion vectors of adjacent blocks, there is a high probability that the position of the global minimum of the current block is close to the position of the minimum found in adjacent, previously encoded blocks (Tourapis *et al.*, 1999).

It was soon discovered that accurate predictions avoided local minima and considerably reduced the number of candidates evaluated during the 2-D-logarithmic search. As such, the set of predictors used in predictive motion vector field adaptive search technique (PMV-FAST) (Tourapis *et al.*, 2002) included those of MVFAST and also two other predictors: the median predictor and the motion vector of the collocated block in the previous frame. Enhanced predictive zonal search (EPZS) (Tourapis, 2002) reused all of these predictors and also added the accelerated motion vector. This predictor is efficient for blocks not under constant velocity.

Algorithms using motion vector predictors are said to be prediction-biased. The distinction between prediction-biased and center-biased is that the prediction of the search algorithm might differ from that of the encoder and, as such, the center of the motion estimation search is not the center of the search area. Simulations show that motion vectors are more likely to be prediction-biased than center-biased (Tourapis *et al.*, 2002), even more so for high-motion sequences, where center bias is not as strong.

Another concept to emerge from predictive search is *prediction confidence* (Tourapis *et al.*, 1999). The similarity between the motion vectors of the predictors indicates a level of confidence of the predictions. A prediction originating from a set of very similar predictors is much more likely to be close to the global minimum than a prediction from a set of divergent predictors.

4.5 Early Termination

Prior to (Tourapis *et al.*, 2002), search algorithms used empirical[3] thresholds to reduce the number of operations. The problem with empirical thresholds is that setting the value too

[3]For example:"Emperically[sic], it is found that about 98% of the MBs whose SAD at $(0,0)$ is less than 512 have zero motion vector. Hence, we choose $T = 512$ in our algorithm MVFAST" (Hosur and Ma, 1999). In this quote, the acronym MBs refers to a macroblock, a 16×16 region of pixels.

low reduces speedup, whereas setting the value too high degrades quality of the temporal prediction (Tourapis *et al.*, 2002).

Tourapis *et al.* (2002) introduced adaptive thresholds, a technique of deriving the value of the threshold from adjacent blocks either spatially or temporally. This allows the threshold to adapt to the content of the video sequence.

An interesting early termination mechanism was proposed in (Tourapis *et al.*, 1999), in the context of the half stop circular zonal search. The half stop works as follows: the search terminates if the minimum found so far has not been updated after examining n candidates (Tourapis *et al.*, 1999). In light of the monotonically decreasing search area assumption, it is more likely that the minimum found so far is the global minimum.

The half stop is not used by Tourapis in his subsequent algorithms (PMVFAST or EPZS); however, it does appear in another algorithm: the TZ-Search algorithm.

4.6 TZ-Search Algorithm

The test zonal[4] (TZ)-Search algorithm is a recent search algorithm implemented in the HEVC HM reference encoder. As such, this algorithm sets the bar for search algorithms destined to HEVC. TZ-Search is used exclusively for integer-level ME and uses a repeating 3 step search strategy:

Starting point selection

The candidates evaluated to determine the starting point vary depending on the encoder configuration. By default, the median predictor and the zero motion predictor are used. In enhanced[5] mode, the search algorithm will also consider three spatial neighboring blocks immediately located: to the

[4] So far in this chapter, the author made the conscious effort to avoid the term zonal. So many algorithms claim to be zonal that it is hard to define the properties of zonal algorithms. Historically, zonal algorithms separated the search area into zones. The particularity of the candidates contained in a given zone is that the search algorithm considers them as equally likely to minimize the search area.

[5] This is also sometimes referred to as extended mode.

left, above and the above and to the right. The enhanced candidates are the same as those of MVFAST.

As explained in (Tourapis *et al.*, 2015), it is important to note that TZ-Search lacks the predictors found in PMVFAST and EPZS. This is surprising as predictor selection appears to be the most important feature and the key to the performance of modern search algorithms (Tourapis, 2002).

First search An arrangement of patterns of exponentially increasing sizes are used around the starting point. The size increases until it reaches the edge of the search area or terminates if the minimum found so far as not been updated in the last 3 sizes. An example of the first five patterns is given in fig. 4.4.

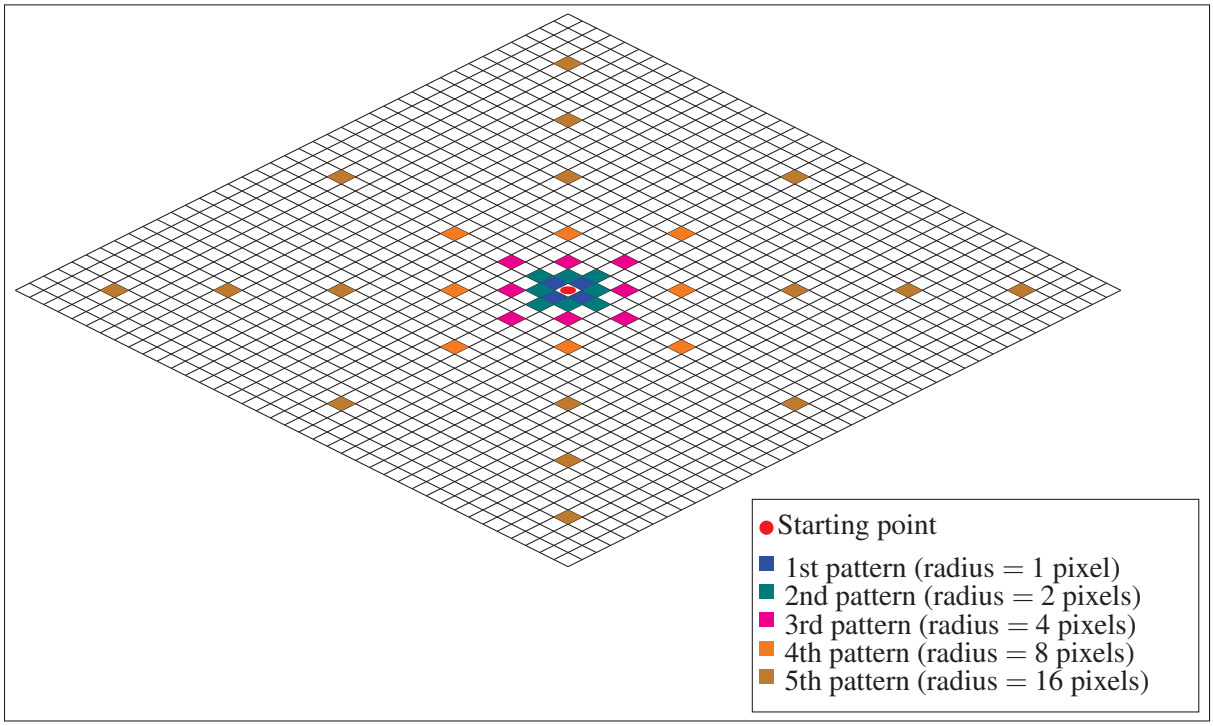


Figure 4.4 Arrangement of patterns of exponentially increasing sizes used by TZ-Search to evaluate a starting point.

The 6th and 7th patterns are the same as the 5th but are of 32 and 64 pixel radius respectively. In the case where the starting point is the best candidate, the search will terminate after the 3rd pattern.

This arrangement of patterns of exponentially increasing sizes appears to contradict the findings of (Tourapis *et al.*, 1999) that the likelihood of finding the optimal candidate decreases as the search moves away from the center of the search area. One could argue that this arrangement avoids local minima; however (Tourapis *et al.*, 2015) claim that this is not the case.

Refinement search

TZ-Search also includes the concept of *search confidence*. Similar to *prediction confidence*, search confidence is based on the radius of the last pattern used to find the candidate. A large pixel radius indicates uncertainty as the density of candidates being evaluated is smaller.

The refinement used depends on the encoder configuration and the search confidence. It can either be a diamond search, a raster search or a star search. In the case of the star search, the starting point is set to the current best point and the previous step is repeated.

4.7 Confidence Intervals For Motion Estimation

Hu and Yang (2014) proposed an approach to use the confidence interval for the value of $\mathrm{ADS}(\mathbf{P},\mathbf{v})$. This interval is based on the assumption that under the central limit theorem when $|\mathbf{P}|$ is large, the random variable

$$v = \frac{\mathrm{PS}(\mathbf{P}) - \mathrm{RPS}(\mathbf{P},\mathbf{v})}{\sqrt{|\mathbf{P}|}\sigma} \tag{4.4}$$

follows roughly a Gaussian distribution. In the previous equation, $|\mathbf{P}|$ is the cardinality of $\mathbf{P}$ and σ^2 is the variance of the residual.

Since the value of σ is unknown, it is estimated from previously encoded neighboring blocks. The blocks used for estimation are the same as the ones used for motion vector prediction in H.264 (see fig. 2.5).

Based on these findings, the probability that $\mathbf{v}$ minimizes the SAD decreases as the ADS increases

$$P\left(\mathbf{v} \in \arg\min_{\mathbf{v}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{v})\right) \propto 2Q\left(\frac{\mathrm{ADS}(\mathbf{P},\mathbf{v})}{\sqrt{|\mathbf{P}|}\sigma}\right) \tag{4.5}$$

where $P(x)$ is the probability of x and $Q(\varepsilon)$ is the Q-function, defined as

$$Q(\varepsilon) = \frac{1}{\sqrt{2\pi}} \int_{\varepsilon}^{\infty} \mathrm{e}^{-\frac{u^2}{2}} \mathrm{d}u\,. \tag{4.6}$$

Given $\varepsilon > 0$, we can use the confidence interval as a threshold for the ADS of the candidate $\mathbf{v}$ as follows

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) > \sqrt{|\mathbf{P}|}\sigma\varepsilon \tag{4.7}$$

$$\Longrightarrow P\left(\mathbf{v} \in \arg\min_{\mathbf{v}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{v})\right) \leqslant 2Q(\varepsilon)\,, \tag{4.8}$$

The value of ε is chosen according the desired compromise between speedup and prediction error.

This confidence interval can be used in conjunction with either optimal or suboptimal search algorithms. When implemented in the TZ-Search, (Hu and Yang, 2014) indicate that the CIME reduced integer-level suboptimal ME time of the TZ-Search by 70%. However, as these assumptions do not always hold, it also increases the BD-Rate by an average of 1.0%.

This chapter described the foundations of modern motion estimation algorithms. We presented the full search algorithm that we will use to evaluate our optimal algorithms. We described the TZ-Search algorithm as, according to the common test conditions, it is the algorithm to compare against in the context of HEVC. Finally, we introduced the CIME as we will compare against it when evaluating our suboptimal algorithms.

CHAPTER 5

MOTION ESTIMATION SEARCH ORDERING AND SUCCESSIVE ELIMINATION

This chapter focuses on the ordering of candidates during the motion estimation search. We start by explaining why the search ordering is so important. From this, we can derive the concepts of an SEA-optimal search ordering and the increasing rate rule. In the next subsections, we introduce our findings related to the increasing rate rule and the proposed cost-based search ordering algorithm.

5.1 Ordering and Transitive Elimination

Search ordering candidates can be separated into two categories:

Category 1 Candidates where the ADS is smaller than or equal to the minimum (optimum) SAD (as shown in fig. 5.1, where the SAD is displayed in magenta and the ADS is displayed in orange).

Category 2 All the other candidates (their SAD displayed in blue and their ADS displayed in green in fig. 5.1).

Candidates in the first category will always require SAD computations. Whereas candidates in the second category might not require any. It all depends on the search ordering used by the SEA. The problem is that block matching is performed iteratively over the candidates, and the minimum SAD is not known in advance, but rather, it is discovered through the process.

The search ordering is therefore crucial to the efficiency of the transitive elimination phase of an SEA. Considering bad candidates up front will cause a high value for the current minimum used by SEA. This high minimum will reduce the efficiency of the transitive elimination and lead to more SAD computations than if a good candidate had been considered first. The ideal situation is when the candidate with the minimum rate-constrained sum of the absolute

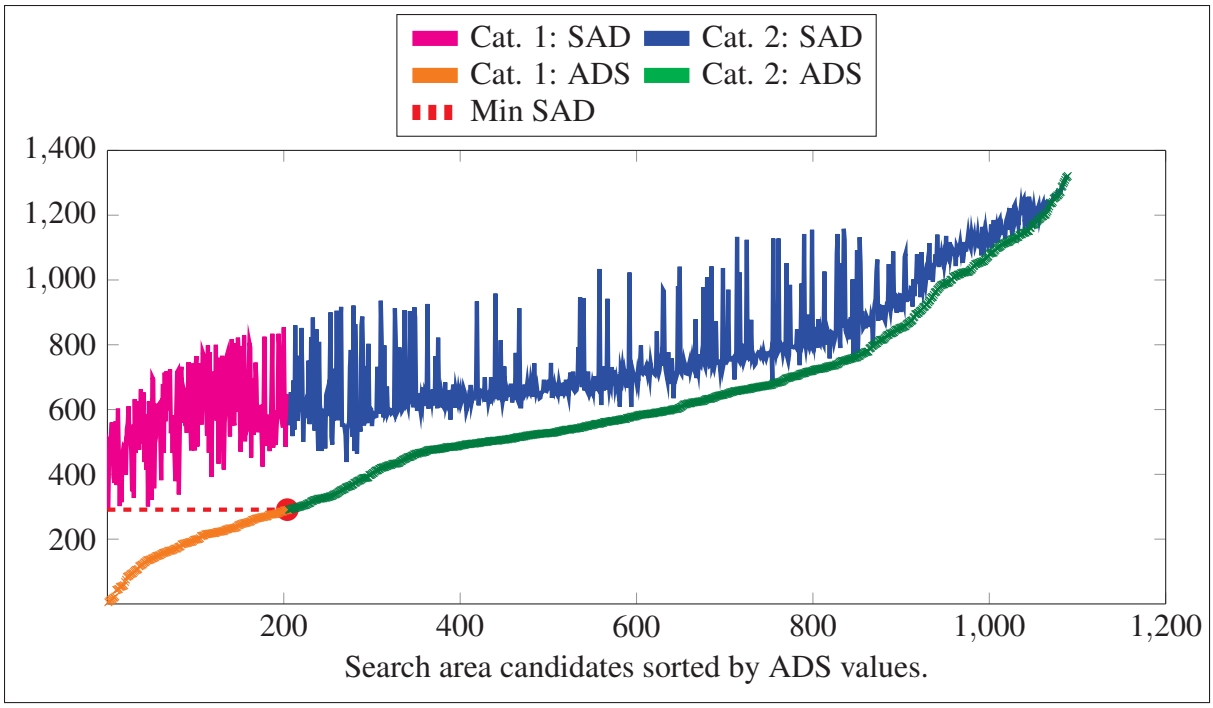


Figure 5.1 SAD and ADS values of category 1 and category 2 candidates. Adapted from Trudeau *et al.* (2015a)

differences (RCSAD) is considered first. Although it may not eliminate the need to evaluate the SAD for all the other candidates, it will lead to the transitive elimination of the highest number of candidates, and thus to the lowest amount of SAD evaluations.

Almost all modern SEA implementations use a spiral scan (fig. 5.2a), or a derivative of the spiral scan ordering, like the one found in the H.264 JM (Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2013) (fig. 5.2b).

These orderings traverse candidates starting at the MVP and move outwards. The underlying assumption is that motion vectors are center-biased. As such, the likelihood of finding the smallest candidate decreases as the magnitude of the motion vector increases.

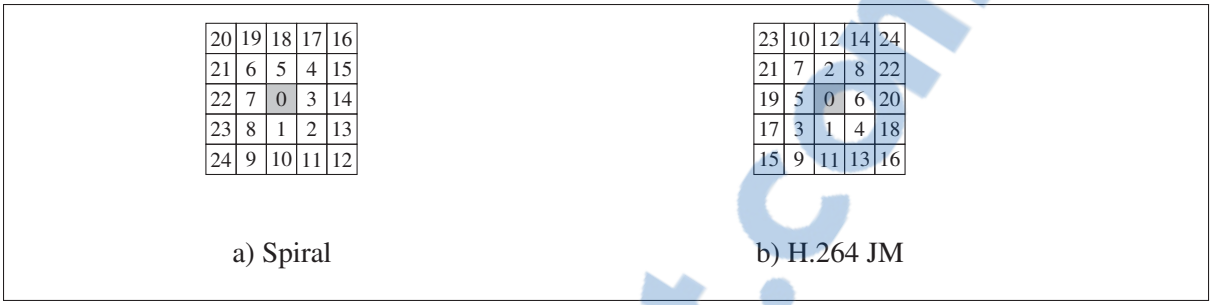


Figure 5.2 Subsets of the spiral search ordering (a) and the H.264 JM implementation of spiral search ordering (b). The gray square is the position of the MVP. Values in these grids show the evaluation order of candidate blocks (from 0 to 24).

5.2 SEA-Optimal Search Ordering

In Trudeau *et al.* (2015a), we defined the term SEA-optimal. Recall fig. 5.1, a search is SEA-optimal if only category 1 SADs are evaluated. It follows that without extra information, the smallest set of candidates that must be evaluated in order to find the global minimum are the candidates in category 1.

More precisely, let $\mathbf{O} \subseteq \mathbf{C}$ be the set of candidates, from the search area, evaluated by an SEA-optimal search algorithm, such that

$$\forall \mathbf{c} \in \mathbf{O} \quad \mathrm{ADS}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \leqslant \min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \,. \tag{5.1}$$

It can be shown that the global minimum is in $\mathbf{O}$

$$\min_{\mathbf{c}\in\mathbf{O}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) = \min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \,. \tag{5.2}$$

It easily follows that search orderings based on geometric patterns, like a spiral, cannot guarantee to be SEA-optimal. The fact of the matter is that SEA-optimal searches are content dependent, whereas static geometric patterns are content independent.

5.2.1 MVP Pruning

Any SEA-optimal search algorithm can take advantage of a powerful pruning method that reduces the set of elements that approach needs to consider. The pruning threshold is obtained by evaluating the RCSAD of a candidate upfront.

The center bias assumption, more specifically the prediction bias assumption, implies that the predicted motion vector ($\mathbf{p}$) is the most likely candidate to be the global minimum. As such, it is an excellent candidate to use for pruning. Let $\mathbf{E} \in \mathbf{C}$ be the set of pruned candidates, such that

$$\forall \mathbf{c} \in \mathbf{E} \quad \mathrm{ADS}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) > \mathrm{SAD}(\mathbf{P},\mathbf{p}) + \lambda \mathrm{R}(\mathbf{p})\,. \tag{5.3}$$

By Eq. (5.1), we know that $\mathbf{O} \subseteq \mathbf{E}^{\complement}$, where $\mathbf{E}^{\complement}$ is the complement of the set of pruned candidates (i.e., $(\mathbf{E}^{\complement})^{\complement} = \mathbf{E}$) in the set $\mathbf{C}$. In other words, MVP pruning will never prune category 1 candidates including the global minimum.

5.2.2 The Sorted Subset Approach

Sorting the candidates in $\mathbf{E}^{\complement}$ by their rate-constrained absolute difference of sums (RCADS) is a simple way to achieve an adaptive search ordering that guarantees to be SEA-optimal. This is what we proposed in Trudeau *et al.* (2015a). Starting with the lowest RCADS candidate, we successively evaluate the cost function until the latter exceeds the lowest cost found so far. At which point evaluating the cost function of the remaining block matching candidates becomes irrelevant. Concretely, we stop evaluating candidates on the first occurrence of Eq. (5.4) being true. Intuitively, one can reason that

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v}) > \min_{\mathbf{c}\in\mathbf{F}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \quad \forall \mathbf{v} \in \mathbf{F}^{\complement} \tag{5.4}$$

$$\implies \min_{\mathbf{c}\in\mathbf{F}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) < \min_{\mathbf{c}\in\mathbf{F}^{\complement}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \tag{5.5}$$

$$\implies \min_{\mathbf{c}\in\mathbf{F}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) = \min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \tag{5.6}$$

$$\implies \mathbf{F} = \mathbf{O}\,. \tag{5.7}$$

More formally, we prove that when Eq. (5.4) holds, $\mathbf{F} = \mathbf{O}$ as follows: Starting from the obvious fact that

$$\mathbf{F} \cup \mathbf{F}^{\complement} = \mathbf{O} \cup \mathbf{O}^{\complement}, \tag{5.8}$$

when Eq. (5.4) holds, Eq. (5.1) implies that

$$\mathbf{F}^{\complement} \cap \mathbf{O} = \{\}, \tag{5.9}$$

It easily follows that $\mathbf{F} = \mathbf{O}$

This approach is illustrated in fig. 5.1, where **v** in the condition described in Eq. (5.4) is the first candidate in category two. The red dot is the last point in category 1. Furthermore, **F** is the set of candidates in category 1 and $\mathbf{F}^{\complement}$ is the set of candidates in category 2. Table 5.1 recapitulates all the sets used in the previous equations.

Table 5.1 Recapitulation of the defined sets and their description

Set	Description
C	The set of candidates in the search area.
O	The set of candidates evaluated by an SEA-optimal search algorithm.
F	The set of candidates evaluated so far during the search.
E	The set of pruned candidates.

5.2.3 Experimental Results and Discussion

To measure the savings of an SEA-optimal search ordering, proposed in this section, we implemented both the proposed solution and the RCSEA with a spiral scan search ordering in the H.265/HEVC HM 13.1 reference software (McCann *et al.*, 2014). By comparing the cost function evaluation of both approaches, we could determine the percentage of unnecessary cost function evaluations performed by the RCSEA with a spiral search ordering.

Table 5.2 presents detailed results of our experiment for the first 100 frames of standard Class C (832×480) video sequences (*Basketball Drill*, *Party Scene*, *BQ Mall* and *Race Horses*). The results are presented by block sizes and by QP values. We used the main profile with the following alterations: 5 reference frames, disabled asymmetric motion partitions, integer level pixel precision motion estimation and full search motion estimation.

As stated in (Trudeau *et al.*, 2014), changing the search ordering has negligible to no impact on rate-distortion as all candidates are considered, only in a different order. However, it does have an impact on transitive elimination.

From the results in table 5.2, we can see that the proposed algorithm is more effective for smaller block sizes. This is due to the fact that smaller blocks comprise fewer pixels, which leads to more precise ADS values. These values filter out more unnecessary cost function evaluations. Since most SEA-based algorithms partition bigger blocks using multiple small partitions to improve filtering efficiency (Gao *et al.*, 2000; Zhu *et al.*, 2005), they would benefit significantly from the proposed method.

As the QP increases, the effectiveness of the proposed algorithm also increases. This is analogous to the findings of (Coban and Mersereau, 1998), and is caused by an increase in the value of the Lagrange multiplier (λ). This in turn increases the ratio between the weighted number of bits required to encode the motion vector and the prediction error. When this occurs, the rate constraint becomes more significant and allows more block-matching candidates to be filtered.

Table 5.2 shows that the proposed search ordering is, on average, more efficient with sequences that contain important and unpredictable movement (*Basketball Drill* and *Race Horses*), than with those with more predictable sequences. Unpredictable sequences lead to less precise motion vector predictions, and for them, hard-coded search orderings, such as the spiral scan, will search around a bad starting point leading to unnecessary cost function evaluations. In the same context, by sorting block matching candidates, the proposed adaptive approach exploits

Table 5.2 Unnecessary cost function evaluations on class C videos (832×480) made by an RCSEA with a spiral scan search ordering in the H.265 HM reference software compared to the proposed method

Block Size	QP	BasketballDrill	Party Scene	BQMall	RaceHorses
4×8, 8×4	22	8.29%	5.58%	3.35%	7.59%
8×8	22	4.74%	3.56%	2.33%	6.34%
8×16, 16×8	22	3.59%	2.60%	1.66%	5.97%
16×16	22	3.22%	2.15%	1.23%	5.80%
16×32, 32×16	22	2.94%	1.94%	0.96%	5.44%
32×32	22	2.61%	1.60%	0.77%	4.93%
32×64, 64×32	22	2.14%	1.18%	0.60%	3.87%
64×64	22	1.89%	0.60%	0.31%	2.89%
4×8, 8×4	27	10.99%	6.73%	3.25%	8.46%
8×8	27	5.94%	4.33%	2.14%	6.49%
8×16, 16×8	27	3.56%	3.01%	1.53%	5.79%
16×16	27	2.87%	2.21%	1.16%	5.50%
16×32, 32×16	27	2.57	1.91	0.93	5.18
32×32	27	2.23%	1.58%	0.75%	4.73%
32×64, 64×32	27	1.81%	1.16%	0.56%	3.79%
64×64	27	1.62%	0.61%	0.31%	2.88%
4×8, 8×4	32	13.12%	7.95%	3.30%	10.10%
8×8	32	7.78%	4.99%	1.97%	6.99%
8×16, 16×8	32	4.39%	3.30%	1.38%	5.86%
16×16	32	2.89%	2.29%	1.06%	5.42%
16×32, 32×16	32	2.51%	1.83%	0.88%	5.10%
32×32	32	2.14%	1.46%	0.72%	4.62%
32×64, 64×32	32	1.76%	1.07%	0.52%	3.80%
64×64	32	1.46%	0.58%	0.27%	2.79%
4×8, 8×4	37	15.35%	9.19%	3.31%	12.45%
8×8	37	9.06%	5.51%	1.84%	7.33%
8×16, 16×8	37	5.19%	3.43%	1.21%	5.50%
16×16	37	3.07%	2.18%	0.90%	4.93%
16×32, 32×16	37	2.30%	1.68%	0.78%	4.66%
32×32	37	1.93%	1.29%	0.66%	4.27%
32×64, 64×32	37	1.61%	0.98%	0.50%	3.46%
64×64	37	1.37%	0.49%	0.28%	2.72%

the relative precision of the RCADS, allowing candidates around the true motion vector to be considered earlier in the search process.

5.3 The Increasing Rate Rule

Motion vector magnitude is not equivalent to motion vector cost. As such, when transitive elimination is used in a rate-constrained context, *transitive impairment* can occur, if the search ordering is based on motion vector magnitude and not on motion vector cost (Eq. (2.4)).

As explained in section 3.1, transitive elimination of a candidate $\mathbf{v}$ occurs when

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v}) \geqslant \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) + \lambda \mathrm{R}(\hat{\mathbf{v}})\,. \tag{5.10}$$

Transitive impairment can be made more explicit by reordering the previous equation as follows:

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda\,(\mathrm{R}(\mathbf{v}) - \mathrm{R}(\hat{\mathbf{v}})) \geqslant \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}})\,. \tag{5.11}$$

Let $\Delta R = R(\mathbf{v}) - R(\hat{\mathbf{v}})$ be the rate differential. When $\Delta R < 0$, the ADS is weakened, as $\Delta R \times \lambda$ is subtracted from it. Remember that λ grows exponentially with the QP. As such, transitive impairment can have significant impact on transitive elimination in low bit rate scenarios and can cause useless SAD computations.

The only way to avoid this is to use an ordering arranged by increasing rates. That way, $\Delta R \geqslant 0$, and now, the search ordering can even improve transitive elimination. This is the reasoning behind the *increasing rate rule*, one of our first findings and a contribution proposed in Trudeau *et al.* (2014).

5.3.1 Early Termination

The increasing rate rule not only allows to improve transitive elimination, it also allows early termination, a new speedup for optimal ME algorithms. To our knowledge, no other author has ever proposed an optimality preserving early termination mechanism in the context of SEA.

We shall now describe the early termination mechanism and demonstrate that it preserves the optimal candidate. To do so, let $\mathbf{F}^{\complement}$ be the set of remaining candidates left to evaluate during ME (i.e. the complement of the set of candidates evaluated so far $\mathbf{F}$). Under the increasing rate rule, we implicitly know that

$$\mathrm{R}(\mathbf{v}) \leqslant \mathrm{R}(\mathbf{m}) \quad \forall\, \mathbf{v} \in \mathbf{F}, \forall\, \mathbf{m} \in \mathbf{F}^{\complement}. \tag{5.12}$$

When the weighted cost of the candidate motion vector is greater than or equal to the current minimum cost function, the current candidate cannot minimize the cost function. By the increasing rate rule Eq. (5.12), it follows that the current minimum is the global minimum, and the block-matching algorithm (BMA) can stop as no other candidate can produce a smaller value, as stated in

$$\lambda \mathrm{R}(\mathbf{m}) > \mathrm{SAD}(\mathbf{P}, \hat{\mathbf{v}}) + \lambda \mathrm{R}(\hat{\mathbf{v}}) \quad \forall\, \mathbf{m} \in \mathbf{F}^{\complement} \tag{5.13}$$

$$\implies \mathbf{F}^{\complement} \cap \underset{\mathbf{c} \in \mathbf{C}}{\arg\min} \left(\mathrm{SAD}(\mathbf{P}, \mathbf{c}) + \lambda \mathrm{R}(\mathbf{c})\right) = \emptyset \tag{5.14}$$

Geometrically, the early termination threshold can be visualized in a 2 dimensional cartesian grid, where the x-axis is rate and the y-axis is distortion, as the point where the line of slope λ passing by the point at coordinates ($\mathrm{R}(\hat{\mathbf{v}})$, $\mathrm{SAD}(\mathbf{P}, \hat{\mathbf{v}})$) reaches 0. As shown in fig. 5.3, candidates with higher values of rate can be ignored, as these candidates would require a negative distortion in order to be cost effective.

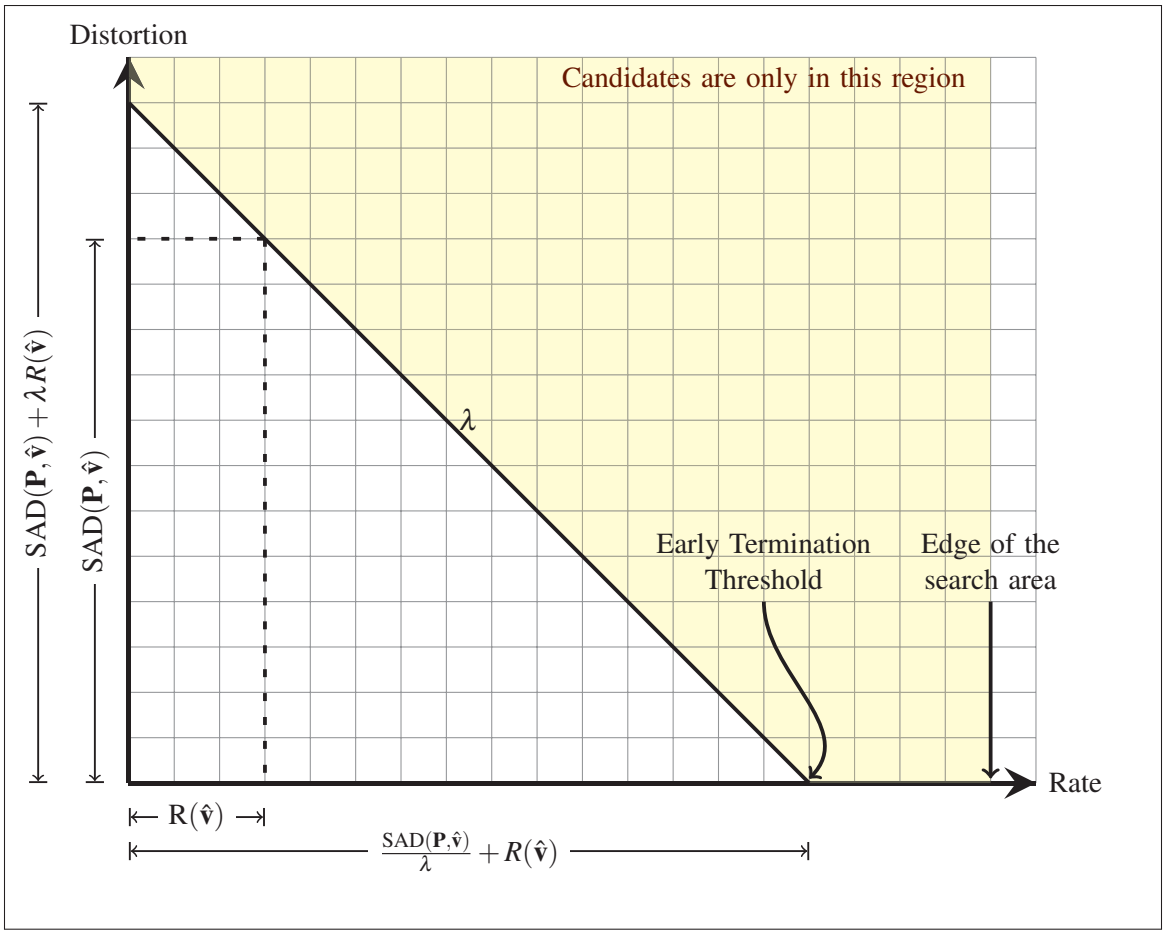


Figure 5.3 Geometric representation of the early termination threshold.

5.4 Cost-Based Search Ordering Pattern

In Trudeau *et al.* (2014), we introduced a new class of search algorithms named *cost-based search algorithms*. To be classified as such, the algorithm must evaluate its candidates by increasing rates, in order to satisfy the increasing rate rule.

We also built a geometric pattern, similar to a spiral scan, but designed to satisfy the increasing rate rule. A subset of this pattern is shown in fig. 5.4.

This is the first cost-based search algorithm. Note that in the fig. 5.4, candidates on the axes are evaluated first, since their motion vectors require fewer bits. Next, the candidates closest to the center and the axes are evaluated (similar to an asymptote shape).

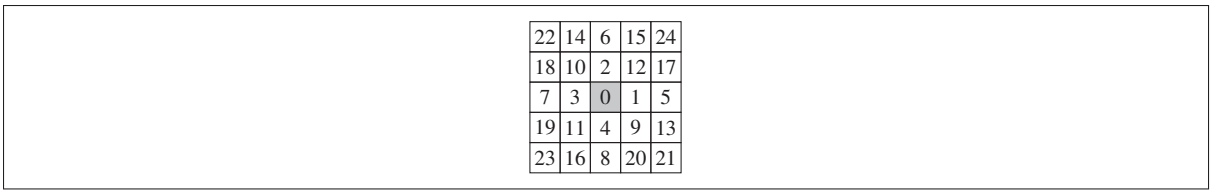


Figure 5.4 Subsets of the cost-based geometric pattern proposed in Trudeau *et al.* (2014). The gray square is the position of the MVP. The values in the grid show the evaluation order of candidate blocks (from 0 to 24).

As the likelihood of finding better candidates earlier in the search increases when the ordering alternates between quadrants, we decided that the pattern should alternate between quadrants. This is also performed in the H.264 JM implementation of the spiral search (fig. 5.2b).

5.4.1 Experimental Results and Discussion

To evaluate the cost-based search ordering pattern proposed in this chapter, we implemented it in the H.264/AVC JM 18.5 reference software Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG (2013). The reference software also contains an implementation of the spiral search ordering algorithm.

We compared the number of SAD operations required to encode CIF (352×288) sequences using the reference software's spiral search implementation against the proposed search ordering algorithm. To simplify the results, we used the baseline profile with the following alterations: 5 reference frames, integer pixel precision motion estimation and only 16×16, 8×8 and 4×4 block partitions. Similar results are expected with rectangular shaped blocks.

The number of SAD operations required for the *Foreman*, *Football* and *News* sequences are listed in detail in tables 5.3, 5.4 and 5.5 respectively.

Table 5.6 lists the average reduction percentage of SAD operations for the *Foreman*, *Flower*, *Football*, *Mobile*, *News* and *Tempete* sequences. In this table, the column Δ Bits (kb/s) is the average bit rate difference, measured in kilobits per second, between the spiral search ordering encoding and the proposed search ordering encoding. The difference is very small, and is

Table 5.3 SAD reduction using the proposed cost-based search ordering pattern compared to H.264 JM reference software's implementation of the spiral search, as a function of the block size and the QP for the Foreman video sequence.

		Foreman # of SAD operations for 300 frames		
QP	Size	Spiral	Proposed	Red. %
28	4	416 262 070	388 993 410	6.55%
28	8	785 227 992	765 544 865	2.51%
28	16	409 325 310	401 608 855	1.89%
32	4	225 442 778	204 861 411	9.13%
32	8	648 570 481	627 984 818	3.17%
32	16	422 019 606	414 160 704	1.86%
36	4	107 804 660	95 285 467	11.61%
36	8	529 033 021	507 752 081	4.02%
36	16	426 912 515	419 050 593	1.84%
40	4	47 435 348	41 836 990	11.80%
40	8	405 457 244	383 738 455	5.36%
40	16	421 173 116	413 071 553	1.92%
		Average SAD reduction		5.14%

attributable to the search ordering algorithms finding different best candidates, but with the same cost values. This phenomenon has a low probability, but considering the number of candidate blocks evaluated, it does occur. This leads to an even smaller average difference in peak signal to noise ratio (PSNR) for the luma plane (Y), listed in the Δ PSNR-Y column. For the Δ columns, a negative value indicates that the value, resulting from the encoding of the proposed search ordering, is smaller than that obtained by the spiral search encoding.

From the results in tables: 5.3, 5.4 and 5.5, we can see that the proposed algorithm is more effective for smaller partition sizes. This is due to the higher ratio of bits required for the motion vector of the candidate block versus its SAD value. When this ratio increases, the weakening effect on the rate-constraint of the filtering criterion caused by the spiral search is more significant. A similar situation arises when the QP increases, which leads to an increase in the value of λ, which is multiplied by $\mathrm{R}(\mathbf{v})$.

Table 5.4 SAD reduction using the proposed cost-based search ordering pattern compared to H.264 JM reference software's implementation of the spiral search, as a function of the block size and the QP for the Football video sequence.

		Football # of SAD operations for 300 frames		
QP	Size	Spiral	Proposed	Red. %
28	4	1 115 661 675	1 035 142 134	7.22%
28	8	1 955 919 279	1 882 526 019	3.75%
28	16	903 904 793	879 156 973	2.74%
32	4	698 105 494	638 767 242	8.50%
32	8	1 659 309 376	1 594 498 115	3.91%
32	16	922 208 528	898 298 829	2.59%
36	4	393 409 060	353 080 194	10.25%
36	8	1 185 610 980	1 133 690 522	4.38%
36	16	923 795 311	900 217 448	2.55%
40	4	183 815 418	161 532 698	12.12%
40	8	760 172 034	712 290 223	6.30%
40	16	876 436 298	856 138 643	2.32%
		Average SAD reduction		5.55%

Since most recent SEA algorithms use partitions to improve filtering efficiency, for example Gao *et al.* (2000); Zhu *et al.* (2005), many 16×16 and 8×8 blocks will be evaluated using smaller partitions. When combined with the proposed method, these algorithms will lead to an overall increase in the reduction of SAD operations.

It can be shown that the filtering criterion cannot be weakened when there is no motion or when the motion is well predicted by MVP. This can be observed in tables 5.3, 5.4, 5.5 and 5.6, where the proposed search ordering is, on average, more efficient with sequences that contain important and unpredictable movement (*Foreman*, *Football*), as compared to those with more predictable movement.

Table 5.5 SAD reduction using the proposed cost-based search ordering pattern compared to H.264 JM reference software's implementation of the spiral search, as a function of the block size and the QP for the News video sequence.

		News # of SAD operations for 300 frames		
QP	Size	Spiral	Proposed	Red. %
28	4	134 537 882	128 099 468	4.79%
28	8	290 136 328	286 173 266	1.37%
28	16	309 741 039	308 103 709	0.53%
32	4	81 710 975	76 734 942	6.09%
32	8	255 001 256	249 897 228	2.00%
32	16	291 083 798	289 592 570	0.51%
36	4	44 836 321	41 544 099	7.34%
36	8	215 288 507	212 294 102	1.39%
36	16	270 475 527	269 005 875	0.54%
40	4	24 308 026	22 453 474	7.63%
40	8	166 808 837	163 627 932	1.91%
40	16	264 566 993	263 016 343	0.59%
		Average SAD reduction		2.89%

Table 5.6 Average SAD reduction for the spiral search ordering versus the proposed search ordering.

Sequence	# Fr.	SAD Red.	Δ Bits (kb/s)	Δ PSNR-Y
Foreman	300	5.14%	-0.18	0.0000
Flower	250	1.61%	-0.21	-0.0017
Football	260	5.55%	0.09	-0.0025
Mobile	300	0.80%	-0.18	0.0008
News	300	2.89%	-0.04	0.0017
Tempete	260	1.14%	-0.11	0.0008
Average		2.86%	-0.10	-0.0001

5.5 Implementation Considerations of the Increasing Rate Rule

Further work has shown that implementation considerations found in HEVC and H.264 introduce certain conditions where a static geometric pattern will not always fully satisfy the increasing rate rule. More specifically, we identified two problematic issues: off-centered

search areas and asymmetric motion vector costs. In order to satisfy the increasing rate rule in all conditions, an impractical amount of geometric patterns is required.

5.5.1 Asymmetric Distribution of Motion Vector Costs

As described in section 2.2, modern encoders perform ME in two steps. The BMA is first performed at the integer pixel level, followed by a fractional pixel level refinement. Our work is focused on the integer level, for two reasons. First, fractional level block-matching operations account for only a small percentage of block-matching operations. This is mainly due to the high computational cost required to interpolate the sub pixel values. Second, modern encoders rely on the SATD to evaluate candidates at the fractional level, whereas our work is based on the SAD.

However, even at the integer level ME, the methods used to compute the MVP, previously described in section 2.4, commonly yield fractional MVPs. Fractional MVPs are problematic because they point to positions that do not exist at the integer level.

We define the integer level motion vector predictor (ILMVP), $\widetilde{\mathbf{p}} \in \mathbb{Z}^2$, as the closest integer level position from the fractional level MVP $\mathbf{p} \in \frac{1}{4}\mathbb{Z}^2$.

This position is obtained by a round half up operation (round( )) on the fractional values of the MVP:

$$\widetilde{\mathbf{p}}_x = \text{round}\left(\mathbf{p}_x\right) \text{ , } \widetilde{\mathbf{p}}_y = \text{round}\left(\mathbf{p}_y\right) \text{ .} \tag{5.15}$$

Similarly, we denote the integer level version of $\mathbf{v}$ as $\widetilde{\mathbf{v}}$.

Figure 5.5 shows an example of asymmetric distribution of MV costs, where horizontal integer level candidates to the left of the ILMVP are less expensive than candidates to the right.

This phenomenon is caused by the fact that the search ordering is based on an integer level MVs while quarter pixel level MV differentials are encoded. To cope with this situation, we must track the offset $(x_{\text{off}}, y_{\text{off}}) \in \{(x,y)\,|\,x,y \in \{0, \pm\frac{1}{4}, \pm\frac{1}{2}\}\}$, defined as the difference between

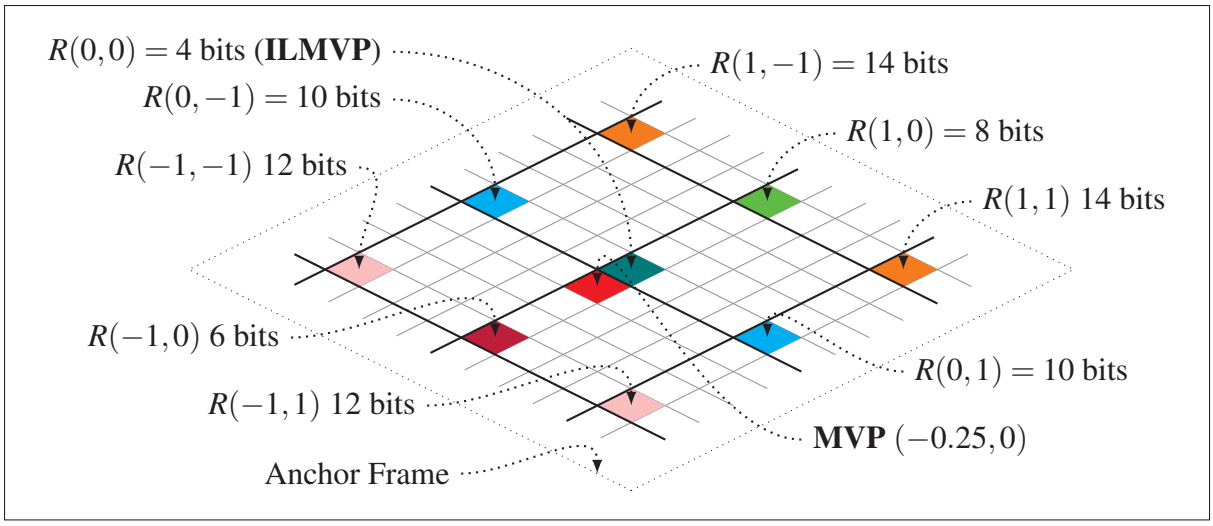


Figure 5.5 The MVP is shown with a red block at $(-0.25,0)$. The candidates are symmetrically located around the ILMVP (teal block); however, horizontal integer level candidates on the left require fewer bits than the candidates on the right.

the ILMVP $\widetilde{\mathbf{p}}$ and the MVP $\mathbf{p}$. The offset is computed as follows:

$$x_{\text{off}} = \mathbf{p}_x - \widetilde{\mathbf{p}}_x \,,\; y_{\text{off}} = \mathbf{p}_y - \widetilde{\mathbf{p}}_y \,. \tag{5.16}$$

The reason why asymmetric distributions of MV costs are problematic is that the offset between the ILMVP and the MVP must be known in order to satisfy the increasing rate rule. Considerable complexity must be added to an approach that relies on a search ordering based on a geometric pattern in order to handle all possible asymmetric configurations of MV costs.

5.5.2 Off-Centered Search Areas

We define an off-centered search area as a search area where the center does not correspond to the ILMVP. Off-centered search areas can be caused by clipping or by the use of refinement zones inside the search area.

Clipping occurs when the current block is near the edge of the frame and the search area is limited by the size of the computer memory allocated to store the image. Correspondingly,

refinement zones are used in order to save computations during bidirectional search, as explained in section 2.3.

A refinement zone is a subset of the search area centered on the position of a best candidate found in another reference frame. Some of these refinement zones do not even include the MVP, as illustrated in fig. 5.6.

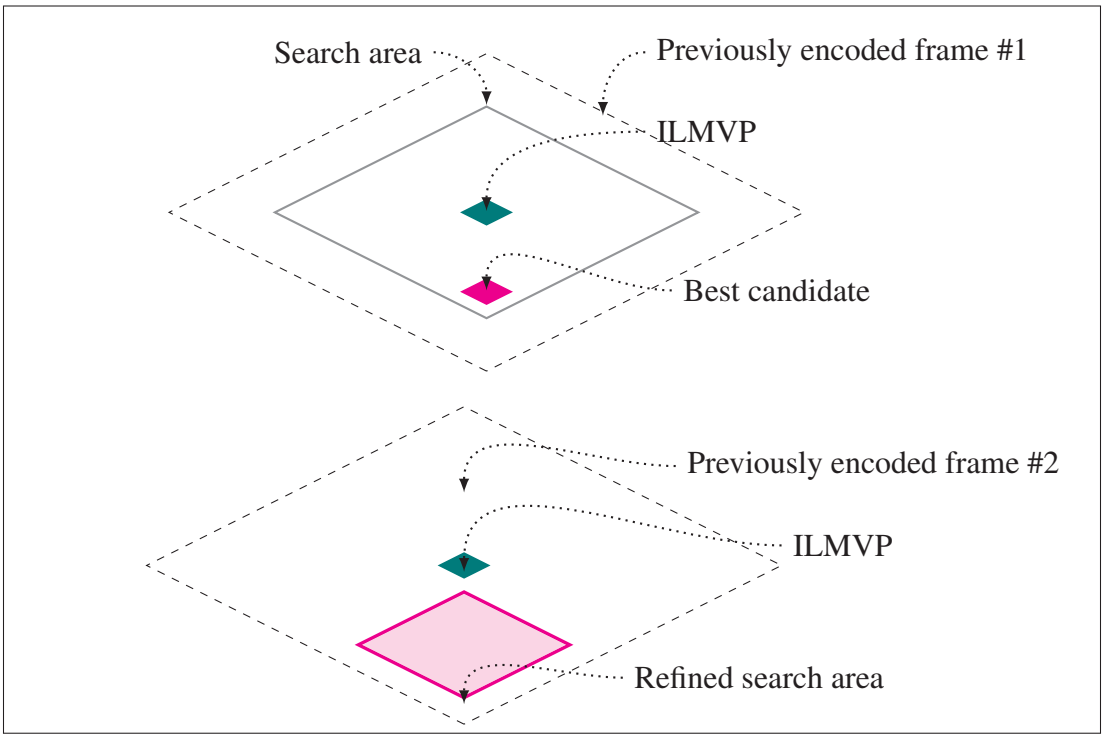


Figure 5.6 Two-step approach used in modern encoders for fractional pixel accuracy search, notice that the ILMVP (teal block) is not in the refined search area (magenta surface).

In this situation, additional computations are required by the BMA to adapt the geometric pattern to the context. It follows that, for off-centered search areas, satisfying the increasing rate rule with a geometric pattern requires supplemental computations. As such, off-centered search areas impose context-specific requirements on the search ordering.

As explained, because of the asymmetric distribution of MV costs and off-centered motion estimation, the increasing rate rule cannot be satisfied by a static geometric pattern like a spiral.

The combinatorial requirements also prohibits hard coding multiple static search orderings into the encoder. A dynamic solution is required as a specific ordering must be built for every context. Concretely, geometric patterns are obsolete; the next candidate must be determined by its properties. For example, the next candidate can be determined by its cost (Trudeau *et al.*, 2016a), or by its ADS (Trudeau *et al.*, 2015a).

5.6 Cost-Based Search Ordering

To resolve the previously identified search ordering considerations, we proposed a new cost-based search ordering method in Trudeau *et al.* (2016a). This method, based on a cost-based search ordering model, dynamically produces a search ordering that follows the increasing rate rule. This new ordering not only improves the effectiveness of the SEA, but, more importantly, it does not require the computation of the motion vector cost (Eq. (2.6)), it reduces the overhead related to off-centered search areas and it allows for a new optimization: an early termination criterion for the BMA.

In the conventional approach, the MVs are ordered according to a geometric pattern which requires computing the rate (Eq. (2.6)) for every MV. A cost-based model is fundamentally different: the encoding cost c of interest is known, but the associated MVs are unknown. Given an unlimited search area, each quadrant will contain as many surfaces of cost c as there are combinations of bit lengths that sum to c. This relates to the complementary relationship between x and y, which we derive from Eq. (2.8), Eq. (2.7) and Eq. (2.6):

$$c = 2 \times I(4(\mathbf{v}_y - \mathbf{p}_y)) + 2 \times I(4(\mathbf{v}_x - \mathbf{p}_x)) + 2 \tag{5.17}$$

$$\implies I(4(\mathbf{v}_y - \mathbf{p}_y)) = \frac{c}{2} - I(4(\mathbf{v}_x - \mathbf{p}_x)) - 1\,, \forall \mathbf{v} \text{ such that } R(\mathbf{v}) = c\,. \tag{5.18}$$

The encoding cost of an MV differential is broken down into different combinations of bit lengths for the x and y differentials. Let $\Delta_x = \mathbf{v}_x - \mathbf{p}_x$ be the x differential and $\Delta_y = \mathbf{v}_y - \mathbf{p}_y$ be the y differential, where $\Delta \in \frac{1}{4}\mathbb{Z}^2$. Given a certain bit length, we extract Δ_x from Eq. (2.8), in

order to find the corresponding range of values of Δ_x, as in:

$$I(4\Delta_x) = \lfloor \log_2(2|4\Delta_x| + 1) \rfloor \tag{5.19}$$

$$\implies \frac{2^{I(4\Delta_x)-1} - \frac{1}{2}}{4} \leqslant |\Delta_x| < \frac{2^{I(4\Delta_x)} - \frac{1}{2}}{4} \tag{5.20}$$

To obtain Eq. (5.20), we first reformulate Eq. (2.8) without the binary logarithm. This is done by combining the binary logarithm definition $i = \log_2(k) \iff 2^i = k$ and the floor function equivalence $\lfloor i \rfloor = k \iff k \leqslant i < k+1$ as follows:

$$I(v) = \lfloor \log_2 2|v| + 1 \rfloor \iff 2^{I(v)} \leqslant 2|v| + 1 < 2^{I(v)+1}. \tag{5.21}$$

Then, isolating v on the right side of Eq. (5.21) and replacing v by $4\Delta x$ leads to Eq. (5.20).

As explained in section 5.5.1, this model only applies to integer level ME, and so $\mathbf{v} = \widetilde{\mathbf{v}} \in \mathbb{Z}^2$. This implies, using Eq. (5.16), that

$$\Delta_x = \widetilde{\mathbf{v}}_x - \mathbf{p}_x = \underbrace{\widetilde{\mathbf{v}}_x - \widetilde{\mathbf{p}}_x}_{\delta_x \in \mathbb{Z}} - x_{\text{off}}\,. \tag{5.22}$$

Since the approach is quadrant-based (more on this in the next section), we only consider $\Delta_x \geqslant 0$. Therefore, for that quadrant, we have:

$$\frac{2^{I(4\Delta_x)-1} - \frac{1}{2}}{4} \leqslant \Delta_x < \frac{2^{I(4\Delta_x)} - \frac{1}{2}}{4} \tag{5.23}$$

$$\implies \frac{2^{I(4\Delta_x)-1} - \frac{1}{2}}{4} \leqslant \delta_x - x_{\text{off}} < \frac{2^{I(4\Delta_x)} - \frac{1}{2}}{4} \tag{5.24}$$

$$\implies \frac{2^{I(4\Delta_x)-1} - \frac{1}{2}}{4} + x_{\text{off}} \leqslant \delta_x < \frac{2^{I(4\Delta_x)} - \frac{1}{2}}{4} + x_{\text{off}} \tag{5.25}$$

Next, we multiply the numerator and the denominator by two to remove the $-\frac{1}{2}$ for both fractions

$$\frac{2^{I(4\Delta x)} - 1}{8} + x_{\text{off}} \leqslant \delta_x < \frac{2^{I(4\Delta x)+1} - 1}{8} + x_{\text{off}}\,. \tag{5.26}$$

Note that the denominator is the multiplication of four, for quarter pixel values, and two, because of adjacent negative and positive codes in the exponential Golomb coding.

Now, since δ_x is an integer, we need integer values for range boundaries. Therefore, to satisfy Eq. (5.25)

$$\left\lceil \frac{2^{I(4\Delta_x)}-1}{8} + x_{\text{off}} \right\rceil \leqslant \delta_x < \left\lceil \frac{2^{I(4\Delta_x)+1}-1}{8} + x_{\text{off}} \right\rceil . \tag{5.27}$$

Note that the 8 at the denominator is the multiplication of 4, for quarter pixel values, and 2, because of adjacent negative and positive codes in the exponential Golomb coding (see section 2.5).

To accelerate the implementation, integer divisions are used. This is not problematic, since $\mathbf{v} = \widetilde{\mathbf{v}}$. However, integer divisions are equivalent to floored divisions. To convert from ceiling to floor, 7 is added to the numerator of the integer divisions. It can be shown that we obtain the equivalent equation:

$$\left\lfloor \frac{2^{I(4\Delta_x)}+6}{8} + x_{\text{off}} \right\rfloor \leqslant \delta_x < \left\lfloor \frac{2^{I(4\Delta_x)+1}+6}{8} + x_{\text{off}} \right\rfloor \tag{5.28}$$

$$\Longrightarrow \frac{2^{I(4\Delta_x)}+6+8x_{\text{off}}}{8} \leqslant \delta_x < \frac{2^{I(4\Delta_x)+1}+6+8x_{\text{off}}}{8} . \tag{5.29}$$

Note that in Eq. (5.27), the x_{off} was inside the ceiling function. In Eq. (5.29), it must be moved into the integer division, yielding $8x_{\text{off}}$. Although the same notation is used, the divisions in the second line of Eq. (5.29) are integer divisions. As described in the next section, the integer divisions are implemented in the ClipRange function using a right bit shift operator $>>$.

Eq. (5.29) also applies to the y differential. By combining the Δ_x range with the complementary Δ_y range, we obtain a surface, where all candidates have the same cost (see fig. 5.8).

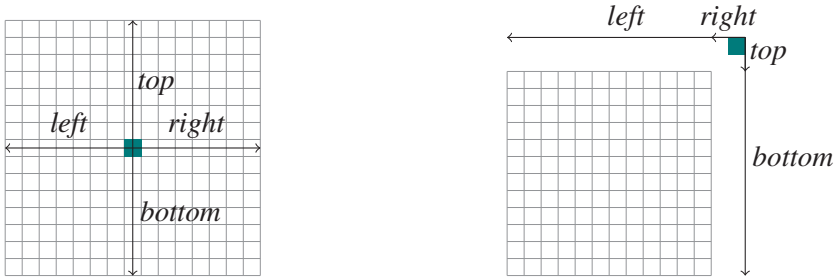
5.6.1 Fast Cost-Based Search Ordering Implementation

A cost-based approach offers the novel possibility of working with same-cost surfaces. Same-cost surfaces are formed inside the search area, when multiple (x, y) pairs yield the same value

for $\mathrm{R}(\mathbf{v})$. Because of the log operator in Eq. (2.8), these same cost surfaces grow exponentially as the value $\mathrm{R}(\mathbf{v})$ increases. These surfaces have interesting properties: the rate-constraint is constant, they are rectangular, disjoint, symmetric in other quadrants, and they appear only once per row inside each quadrant.

The SEARCHORDERING function, presented in Algorithm 5.1, uses these properties to "jump" into every same-cost surfaces of cost c inside the search area. Moreover, because of the underlying binary structure of the exponential Golomb codes and Eq. (5.29), the CLIPRANGE function, shown in Algorithm 5.2, can quickly identify same-cost surfaces boundaries only using simple operations like sums and shifts.

To deal with asymmetric distributions of MV costs and off-centered search areas, the search area is divided into quadrants, where the origin is the ILMVP, as shown in fig. 5.7a. The offset between the MVP and the ILMVP (x_{off}, y_{off}) from Eq. (5.16) is passed as a parameter to the SEARCHORDERING function. The search area is represented using four variables: *top*, *bottom*, *left*, *right*. These variables specify the integer level limits of each quadrant relative to the ILMVP; this can be seen in fig. 5.7.



a) ILMVP centered inside the search area. b) ILMVP outside the search area.

Figure 5.7 Search area defined by: *top*, *bottom*, *left*, *right*. The colored candidate is the ILMVP. In fig. 5.7b, the ILMVP is outside the search area and the values of *top* and *right* are negative.

The SEARCHORDERING function, which will be explained in detail below, is summarized as follows:

Algorithm 5.1: The SEARCHORDERING function creates a cost-based search ordering. For each cost, this function combines the bit length of the x and y components to determine the surface of all candidates of the same cost in every quadrant.

```
SEARCHORDERING(top, bottom, left, right, x_off, y_off)
 1  O = {}
 2  qTop = 4 × (top + y_off)
 3  qBottom = 4 × (bottom − y_off)
 4  height = I(MAX(qTop, qBottom))
 5  yStart = I(MIN(0, qTop, qBottom))
 6  c = 2 × (yStart + 1)
 7  oT = MAX(−bottom, y_off == 0)
 8  oL = MAX(−right, x_off == 0)
 9  while yStart ⩽ height
10      yLen = yStart
11      xLen = c/2 − yLen − 1
12      while xLen >= 0
13          T = CLIPRANGE(2^yLen, −y_off, oT, top)
14          B = CLIPRANGE(2^yLen, y_off, −top, bottom)
15          L = CLIPRANGE(2^xLen, −x_off, oL, left)
16          R = CLIPRANGE(2^xLen, x_off, −left, right)
17          ADDCANDSFROMSURFACE(O, T, B, L, R, c)
18          yLen = yLen + 1
19          xLen = xLen − 1
20          if R.up == right and L.up == left
21              yStart = yLen
22      c = c + 2
23  return O
```

- Lines 2 to 8 initialize key variables of the algorithm.
- Line 9 iterates over all valid costs inside the search area.
- Line 10 assigns the smallest valid bit length to the Δ_y component (*yLen*).
- Line 11 implements Eq. (5.18) to establish the complementary bit length for the Δ_x component (*xLen*).
- Line 12 iterates over all combinations of *xLen* and *yLen* that lead to the same cost, c.

- Lines 13 to 16 invoke the CLIPRANGE function to identify valid ranges for the Δx and Δ_y components (implementing Eq. (5.29)).
- Line 17 invokes the ADDCANDSFROMSURFACE function to add all valid candidates in the same-cost surfaces to the search ordering.

The *qTop* and *qBottom* variables store the *top* and *bottom* quarter pixel level values, adjusted with the offset. With these variables and Eq. (2.8), we can compute their corresponding bit lengths. The biggest bit length is stored in the *height* variable. Since the algorithm starts at the ILMVP and iterates outwards in a vertical fashion, the *height* variable represents the vertical limit from the ILMVP.

The starting bit length of the Δ_y component *yStart* is greater than 0, when either the *top* variable or *bottom* variable is negative. As shown in fig. 5.7b, in such a case, the ILMVP is not in the search area. Since the codewords used alternate values between negative and positive values, negative and positive values have the same bit length (see Eq. (2.8)). Hence, the bit length of a negative value will dictate the starting bit length of the Δ_y component. On line 6, the cost is measured as $2 \times (yStart + 1)$, the '1' counts for the marker bit of the Δ_y component and the smallest code for the Δ_x component: no prefix, no info, just the marker bit.

The c variable stores the current cost, the sum of the exponential Golomb code lengths (Eq. (2.6)). Cost increments are of two, because the next exponential Golomb code requires one bit for the prefix and one bit for the info.

An ambiguous issue with a quadrant-based approach is assigning a quadrant to the column zero and the row zero. This is handled by oL and oT. These variables will skip over column zero or row zero when the $y_{\text{off}} == 0$ or $x_{\text{off}} == 0$ condition is met, respectively ($(y_{\text{off}} == 0)$ returns 1 when the condition is met and returns 0 otherwise; similarly for $(x_{\text{off}} == 0)$). When the offset is non-zero, there is no ambiguity since only one quadrant will have a column zero or a row zero. Another case that is not ambiguous is when the variable of the opposite direction is negative.

The outer loop of the algorithm (line 9) starts at the ILMVP and iterates upwards and downwards (see fig. 5.8). Contrary to conventional approaches that work with row and column indexes, the SEARCHORDERING function works directly with row and column bit lengths. This allows the SEARCHORDERING function to work on an exponential increment of rows and columns in each loop iteration, thus saving considerable looping operations.

An important consideration for this fast implementation is that the $I(\,)$ function is not used to find the bit lengths of the Δ_x and Δ_y values of candidates inside the search ordering. Line 11 shows that the bit length of Δ_x (*xLen*) is computed via the complementary relationship between $I(\Delta_x)$ and $I(\Delta_y)$, Eq. (5.18). Subsequent combinations of Δ_x and Δ_y can be found by decrementing *xLen* and incrementing *yLen*.

An inner loop iterates over all bit length combinations of the Δ_x and Δ_y elements. For each combination, the surface parameters are computed using the CLIPRANGE function and stored in T,B,L,R. These variables represent the range of the cost surface heading towards the top, bottom, left and right respectively. The sign of the offset is adjusted for each invocation of the CLIPRANGE function. Lower and upper boundaries are also specified based on the quadrant of the range and whether or not the zero column or row is included.

Algorithm 5.2: The CLIPRANGE function returns the range of the same-cost surface, between *min* and *max*, using Eq. (5.29), where *v* is 2 to the power of the component bit length, and *off* the offset.

```
CLIPRANGE(v, off, min, max)
1  num = v + 8off + 6
2  R.low = CLIP(num >> 3, min, max)
3  R.up = CLIP((num + v) >> 3, min, max)
4  return R
```

The numerator, *num*, of the lower bound of the range is computed based on Eq. (5.29). It is also reused to compute the upper bound of the range. The function CLIP(v,min,max) will clip the value *v* to allow it to remain between *min* and *max* (the boundaries of the search area).

The ADDCANDSFROMSURFACE function assembles the cost-based search ordering O, by adding all the candidates found inside the previously computed ranges to it. The search ordering, which starts out empty at the beginning of the search, will contain all the candidates, ordered by cost, at the end of the search. The search ordering is passed by reference to the ADDCANDSFROMSURFACE function.

The VEC function is used to create a new vector structure. This structure contains the cost so the RCSEA does not need to compute the cost of the candidates.

Algorithm 5.3: The ADDCANDSFROMSURFACE function appends the MV and cost of the candidates inside the same-cost surface for each quadrant: top (T), bottom (B), left (L), and right (R) to the search ordering (O).

```
ADDCANDSFROMSURFACE(O,T,B,L,R,c)
 1  for y = B.low to B.up − 1
 2      for x = R.low to R.up − 1
 3          O[O.i] = VEC(x,y,c)
 4          O.i = O.i + 1
 5      for x = L.low to L.up − 1
 6          O[O.i] = VEC(−x,y,c)
 7          O.i = O.i + 1
 8  for y = T.low to T.up − 1
 9      for x = R.low to R.up − 1
10          O[O.i] = VEC(x,−y,c)
11          O.i = O.i + 1
12      for x = L.low to L.up − 1
13          O[O.i] = VEC(−x,−y,c)
14          O.i = O.i + 1
```

An important optimization accelerates the inner loop of the SEARCHORDERING function (lines 20 and 21). For a given cost, if the horizontal boundaries of the same-cost surfaces touch the outer horizontal limits of the search area, the starting bit length *yStart* is now *yLen*. This indicates that no subsequent areas are possible for all rows of the current bit length, as depicted in the example of fig. 5.8.

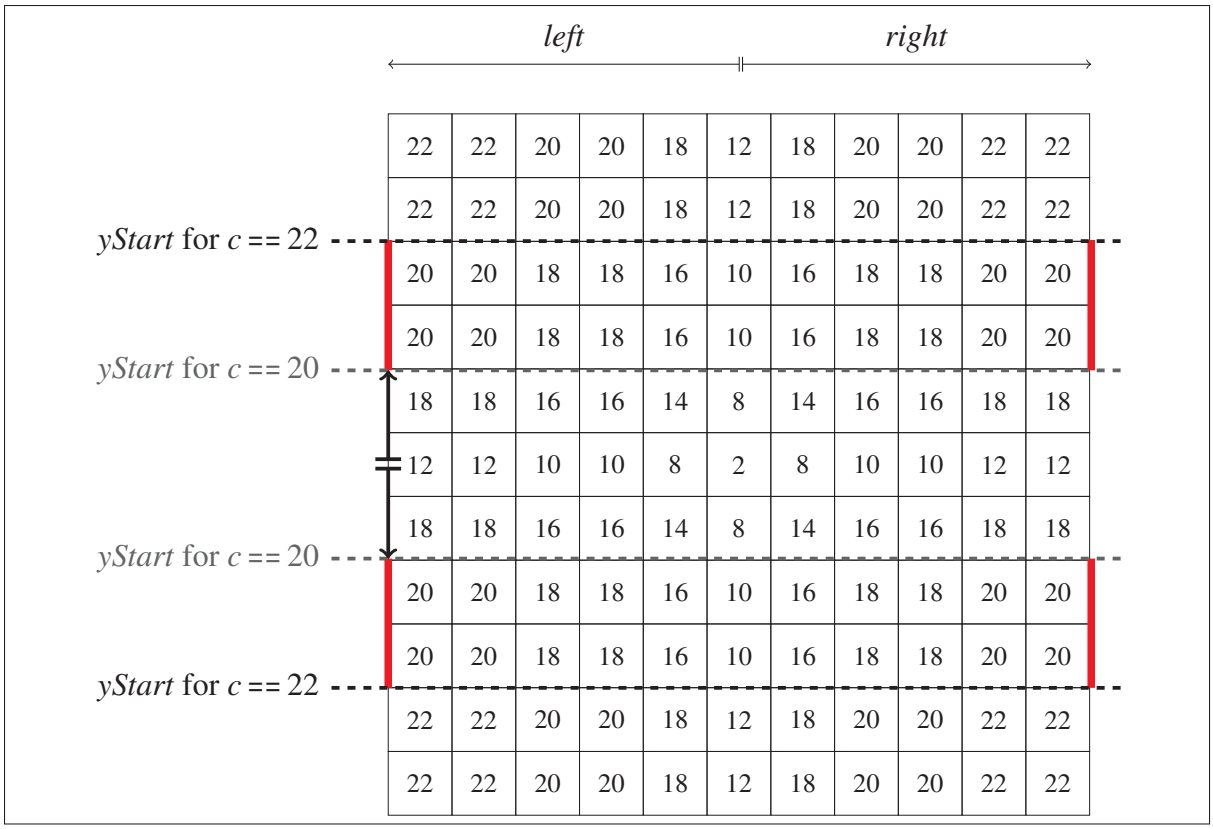


Figure 5.8 Illustrated example of the *yStart* optimization. At the end of the loop iteration where $c == 20$, the boundaries of the same-cost surfaces touch the limits of the search area (highlighted by the red lines). All subsequent loop iterations, will contain candidates with a Δy component $\geqslant$ *yLen*; as such, *yStart* == *yLen*.

Figure 5.8 illustrates that for a cost of 20, the conditions of line 20 of the SEARCHORDERING function are true; thus, the starting bit length *yStart* is now *yLen*. This entails that, as of the next increment of c ($c == 22$), all candidates have a Δy component greater than or equal to *yLen*.

5.6.2 Early Termination Criterion

The increasing rate rule guarantees that the encoding cost of the MV of subsequent candidates in the search ordering will be greater than or equal to the MV encoding cost of the current candidate. This entails that if the weighted MV encoding cost of the current candidate is greater than the smallest rate-constrained candidate cost, then no other candidate will be able to produce a lower rate-constrained cost, and thus the search function should end.

It is important to note that this early termination criterion differs from the one proposed in Trudeau *et al.* (2015a). The latter takes into consideration the ADS which is not possible in this work, because the candidates are sorted by cost, not by ADS.

Algorithm 5.4 shows the MOTIONESTIMATION function, which terminates its BMA loop when the weighted-rate exceeds the current best cost (line 5).

Algorithm 5.4: Implementation of the ME algorithm combined with the RCSEA and the early termination criterion.

```
MOTIONESTIMATION(O, i)
 1  bestCost = ∞
 2  bestVector = (0,0)
 3  for v = 1 to O.length
 4       (x, y, cost) = O[v]
 5       if bestCost ⩽ λ × cost
 6            break
 7       if ADS(i, x, y) + λ × cost ⩽ bestCost
 8            cost = SAD(i, x, y) + λ × cost
 9            if cost ⩽ bestCost
10                 bestCost = cost
11                 bestVector = (x, y)
12  return bestVector
```

5.6.3 Experimental Results and Discussion

To validate the cost-based search ordering described in this section, we used 3 versions of the HEVC HM 16.2 encoder software (McCann *et al.*, 2014). The first version, which we will refer to as HM-FS, is the unmodified encoder software. It uses the built-in raster search ordering for block candidates during ME.

The second version, which we will refer to as HM-RCSEA, implements the RCSEA and represents a state-of-the art implementation of SEA in HEVC. As previously stated, raster search ordering is not commonly used in conjunction with RCSEA. As such, this version

uses the spiral ordering found in the H.264 JM (Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2013). An example of this ordering is shown in fig. 5.2b.

The third version, which we will refer to as HM-CBSEA, uses the same RCSEA implementation as version 2, but here, the search ordering is dynamically built using the SEARCHORDERING function of section 5.6.1. In addition, the BMA was modified to include the early termination criterion (MOTIONESTIMATION).

The test conditions and software configurations used in our experiments conform to the common test conditions and software reference configurations (Bossen, 2013). All versions of the encoder software run the main profile with 8-bit coding for both random access (RA) and low delay (LD) settings. The only changes to the standard configuration files are: enable full search, disable the fast encoder decision (FEN) and disable asymmetric motion partition (AMP). We disabled the latter only to simplify implementation considerations, but AMP and SEA are compatible. All tests were performed on the first 100 frames of the sequences specified by (Bossen, 2013) for classes B, C, and D. Furthermore, they were all performed on a Dual Intel Sandy Bridge EP E5-2670 microprocessor.

5.6.3.1 Comparison with HEVC HM Full Search

In tables 5.7 and 5.8, we compare the encoding time speed up with early termination (ET), the Bjøntegaard Delta PSNR (BD-PSNR) and the Bjøntegaard Delta Rate (BD-Rate) of HM-FS against the proposed solution (HM-CBSEA) for RA and LD settings, respectively. The speed up is measured as the encoding time ratio between HM-FS and HM-CBSEA.

The proposed solution is approximately 5 times faster than the HM encoder. The data shows that the speed up is unaffected by resolution. Moreover, the gains vary between sequences.

We observed that the bit streams produced by HM-FS and HM-CBSEA are not identical. This is caused by ordering differences between same-cost candidates. In other words, when multiple

Table 5.7 Encoding time speed up (with early termination), BD-PSNR and BD-Rate between HM-FS and HM-CBSEA, for the main profile and RA settings.

	RA Main			
Class	Sequence name	Speed up$_{(ET)}$	BD-PSNR	BD-Rate
B (1920×1080)	Kimono	6.3	0.00	-0.04%
	ParkScene	5.5	0.00	-0.05%
	Cactus	6.3	0.00	-0.04%
	BQTerrace	4.9	0.00	0.10%
	BasketballDrive	5.5	0.00	0.18%
C (832×480)	RaceHorses	4.6	0.00	0.12%
	BQMall	6.7	0.00	-0.02%
	PartyScene	4.1	0.00	0.03%
	BasketballDrill	5.2	0.00	-0.01%
D (416×240)	RaceHorses	4.7	0.00	0.08%
	BQSquare	7.3	0.01	-0.12%
	BlowingBubbles	6.4	0.00	0.02%
	BasketballPass	6.0	0.00	0.00%
	Overall	5.8	0.00	0.02%

Table 5.8 Encoding time speed up (with early termination), BD-PSNR and BD-Rate between HM-FS and HM-CBSEA, for the main profile and LD (with B frames) settings.

	LD B Main			
Class	Sequence name	Speed up$_{(ET)}$	BD-PSNR	BD-Rate
B (1920×1080)	Kimono	5.4	0.00	0.01%
	ParkScene	4.8	0.00	-0.01%
	Cactus	5.6	0.00	0.02%
	BQTerrace	4.5	0.00	-0.01%
	BasketballDrive	5.0	0.00	0.00%
C (832×480)	RaceHorses	4.2	0.00	-0.03%
	BQMall	5.7	0.00	-0.01%
	PartyScene	3.7	0.00	0.09%
	BasketballDrill	4.6	0.00	-0.05%
D (416×240)	RaceHorses	4.1	0.01	-0.13%
	BQSquare	6.5	0.00	-0.10%
	BlowingBubbles	5.5	-0.01	-0.01%
	BasketballPass	5.2	0.00	0.00%
	Overall	5.0	0.00	0.01%

global minimums exist, neither encoder might pick the same one. This being said, as shown in tables 5.7 and 5.8, this difference is negligible.

5.6.3.2 Comparison with RCSEA

Part of the speed up of HM-CBSEA over HM-FS is due to RCSEA. Tables 5.9 and 5.10 compare HM-CBSEA with HM-RCSEA.

Table 5.9 The percentage of SAD computation savings, the encoding time speed up (without early termination), the percentage of iterations performed by the block-matching loop and the encoding time speed up (with early termination) between HM-RCSEA and HM-CBSEA, for the main profile and RA settings.

	Proposed Solution vs RCSEA (RA Main)				
Class	Sequence name	SAD Savings	Speed up	Itr. Performed	Speed $Up_{(ET)}$
B	Kimono	2.49%	1.5	28.48%	1.9
	ParkScene	1.87%	1.3	30.87%	1.8
	Cactus	2.23%	1.4	30.77%	1.9
	BQTerrace	1.63%	1.2	34.73%	1.7
	BasketballDrive	3.03%	1.4	33.37%	1.8
C	RaceHorses	3.64%	1.4	43.51%	1.6
	BQMall	4.28%	1.4	31.95%	1.9
	PartyScene	3.25%	1.3	48.76%	1.6
	BasketballDrill	3.43%	1.3	31.00%	1.7
D	RaceHorses	4.85%	1.3	45.23%	1.6
	BQSquare	9.60%	1.4	41.14%	2.0
	BlowingBubbles	7.36%	1.5	42.02%	2.0
	BasketballPass	7.20%	1.4	27.90%	1.9
	Overall	3.64%	1.4	36.33%	1.8

For Tables 5.9 and 5.10, the speed up is measured as the encoding time ratio between HM-RCSEA and HM-CBSEA. The RCSEA only accounts for half of the speed up offered by HM-CBSEA over HM-FS. In other words, our solution almost doubles the speed up that can be achieved with an implementation of the RCSEA. This speed up is achieved by the combination of two of our contributions, the cost-based search ordering algorithm and the early termination optimization.

The percentage of SAD operations saved is much lower than the percentage of iterations saved. However, performing a SAD requires $3 \times M \times N - 1$ operations, whereas a saved iteration

Table 5.10 The percentage of SAD computation savings, the encoding time speed up (without early termination), the percentage of iterations performed by the block-matching loop and the encoding time speed up (with early termination) between HM-RCSEA and HM-CBSEA, for the main profile and LD (with B frames) settings.

	Proposed Solution vs RCSEA (LD B Main)				
Class	Sequence name	SAD savings	Speed up	Itr. Performed	Speed up$_{(ET)}$
B	Kimono	2.99%	1.5	38.90%	1.8
	ParkScene	1.31%	1.3	40.96%	1.7
	Cactus	2.24%	1.4	39.29%	1.8
	BQTerrace	1.44%	1.3	42.35%	1.7
	BasketballDrive	3.06%	1.4	43.40%	1.7
C	RaceHorses	3.51%	1.4	54.33%	1.6
	BQMall	3.78%	1.5	42.80%	1.8
	PartyScene	2.64%	1.4	60.20%	1.5
	BasketballDrill	3.04%	1.4	40.53%	1.7
D	RaceHorses	4.10%	1.5	57.39%	1.6
	BQSquare	7.63%	1.5	49.64%	2.0
	BlowingBubbles	5.05%	1.6	55.00%	1.8
	BasketballPass	6.18%	1.4	37.24%	1.8
	Overall	3.61%	1.4	46.31%	1.7

only accounts for a small number of operations. The computational savings related to a saved iteration are rather limited: 2 operations for the ADS, the early termination check and the looping mechanism. The set of candidates saved by early termination is a subset of the set of candidates eliminated by the RCSEA. In other words, all candidates saved by early termination would not have required a SAD computation. This being said, the high percentage of saved iterations does have an impact on the encoding time.

The first part of the speed up is a direct result of the cost-based search ordering and the various considerations detailed in section 5.6.1. By respecting the increasing rate rule in all encoding conditions, the number of SAD operations decreases by 3.64% on average for RA and 3.61% on average for LD, when compared to the H.264 JM search ordering implemented in the HM. Moreover, as explained in section 7.4, since the search ordering is cost-based, the motion vector cost (Eq. (2.6)) is not computed during the BMA. Furthermore, contrary to HM-RCSEA, the

proposed solution is not burdened with the supplementary computations related to off-centered search areas, as described in section 5.5.2.

Both fig. 5.9 and fig. 5.10 show that SAD savings increase when the QP increases. This is in line with the findings of Coban and Mersereau (Coban and Mersereau, 1998) to the effect that an increase in the Lagrange multiplier has a direct impact on transitive elimination in a rate-constrained context.

The SAD savings shown in fig. 5.9 and fig. 5.10 represent SAD operations that were performed as a result of the impairment of the transitive elimination. As such, we see significant savings in situations where transitive elimination is most effective.

Figure 5.9 also indicates that savings decrease when the resolution increases. This suggests that the increasing rate rule is less problematic for high-resolution video sequences. When the increasing rate rule is less problematic, issues derived from it, such as asymmetric distributions of MV costs and the off-centered search area, are also less problematic. Conversely, for smaller video resolutions, the increasing rate rule, asymmetric distribution of MV costs and off-centered search areas are more problematic.

In fig. 5.10, a significant increase in SAD savings is observed when smaller block sizes are used by the BMA. The obvious reason for this is that the size of the block directly affects the SAD values. This changes the ratio between the SAD and the weighted rate. The rate constraint becomes a more significant part of the transitive elimination. Another reason for this is that smaller size blocks the ADS tends to be a more precise lower bound, again making transitive elimination more efficient.

5.6.3.3 Influence of Early Termination

The second part of the speedup indicates that not all operations of the BMA need to be performed. Tables 5.9 and 5.10 indicate that, on average, the proposed solution performs less than half of the BMA operations. The early termination criterion relies on two important

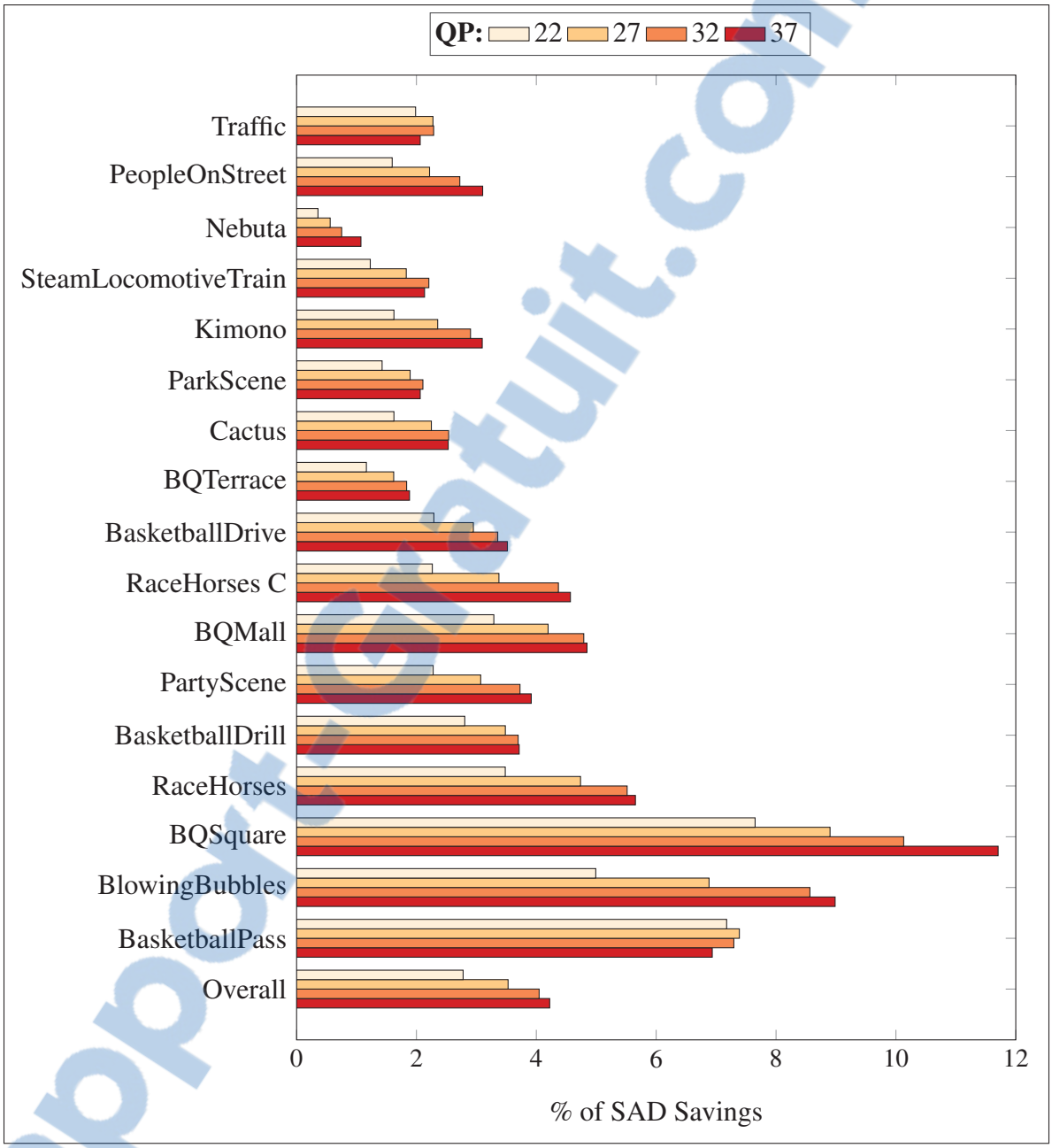


Figure 5.9 Percentage of SAD operation savings, per sequence, for HM-CBSEA, when compared to HM-RCSEA, for the main profile and RA settings.

factors: the precision of the ADS as an estimate for the SAD and the ratio between the rate constraint and the SAD.

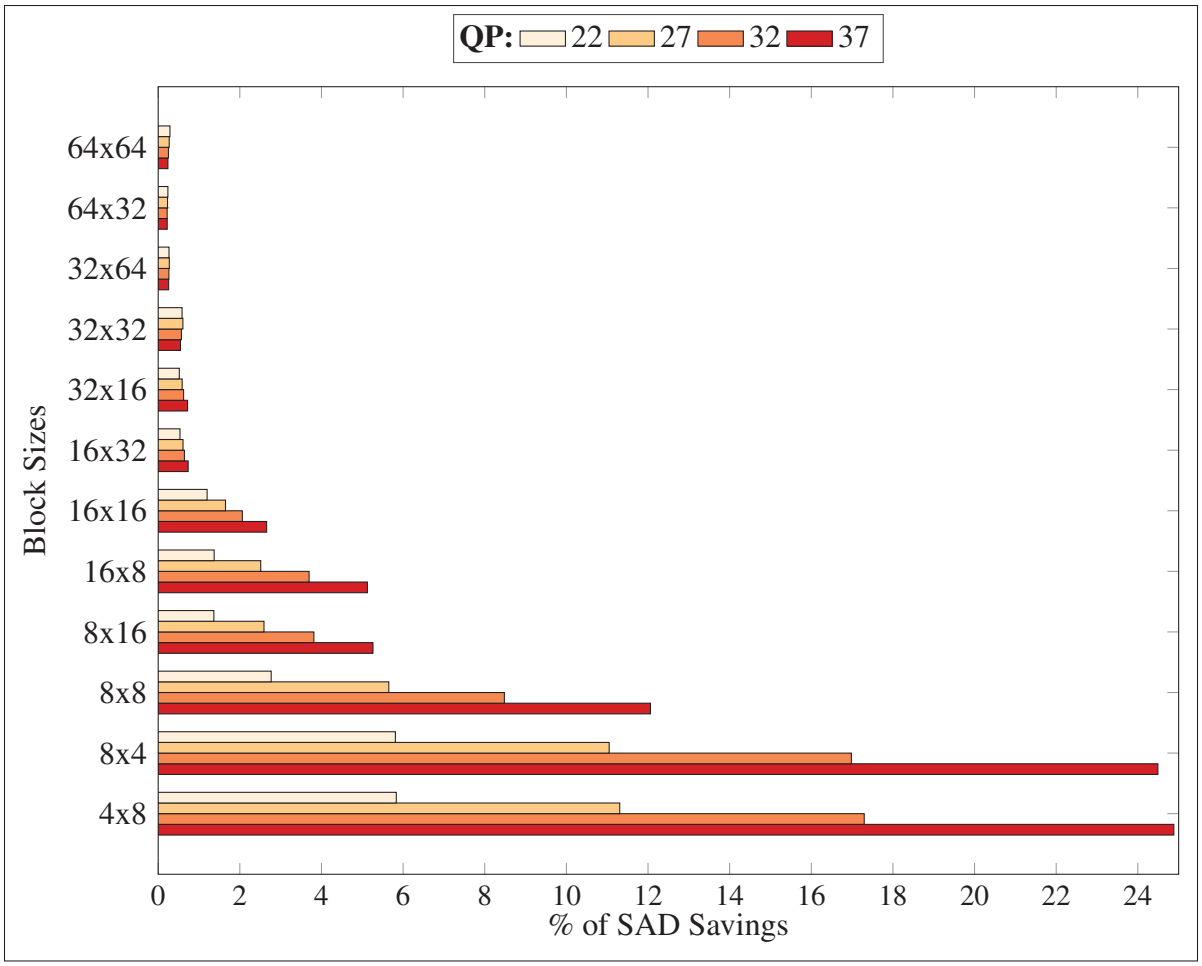


Figure 5.10 Percentage of SAD operation savings, per block size, for HM-CBSEA, when compared to HM-RCSEA, for the main profile and RA settings.

In fig. 5.11, the percentage of iterations performed by the BMA loop is shown by sequence. This data does not suggest a relationship between the number of iterations performed and the resolution of the video sequence. For example, there is less than a 2% difference in the average number of iterations performed between the class D (416×240) and the class C (832×480) versions of the *RaceHorses* sequence.

A relationship clearly exists between the percentage of iterations performed and the QP. Higher QP values require fewer iterations of the BMA, and here again, the Lagrange multiplier is the cause. This is not surprising since increasing the weighted rate causes the early termination to occur earlier in the BMA loop.

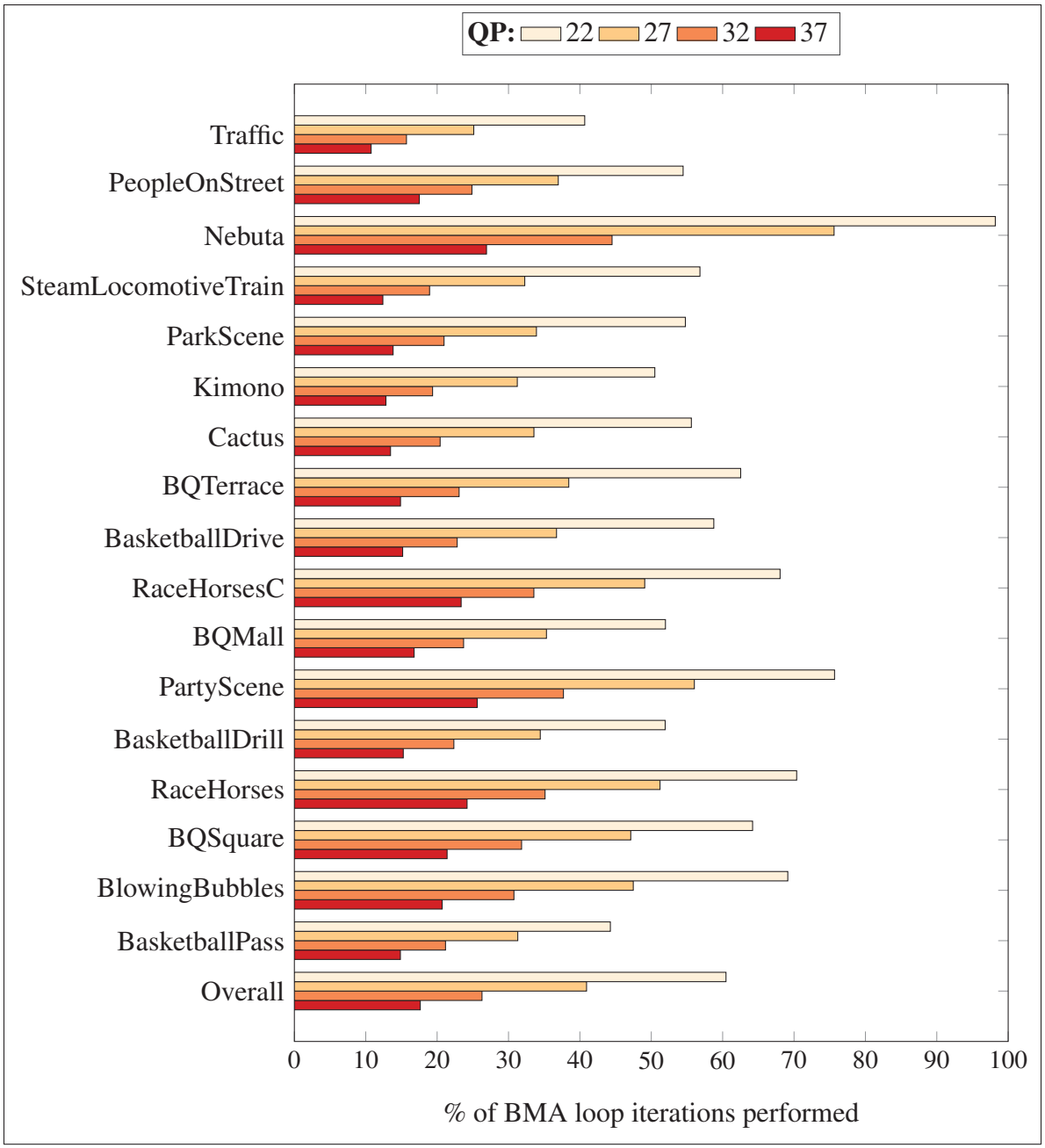


Figure 5.11 Sequence-wise results for the percentage of iterations performed by the block-matching loop in the proposed solution. A value of 100% indicates that all the iterations of the loop are performed.

As shown in fig. 5.12, early termination is particularly effective with small size blocks, where two factors favor the early termination criterion. First, small blocks contain fewer values, and the spatial correlation of natural images generally leads to a small distance between the ADS

and the SAD. Second, fewer values also implies smaller SAD, increasing the ratio between the rate constraint and the SAD.

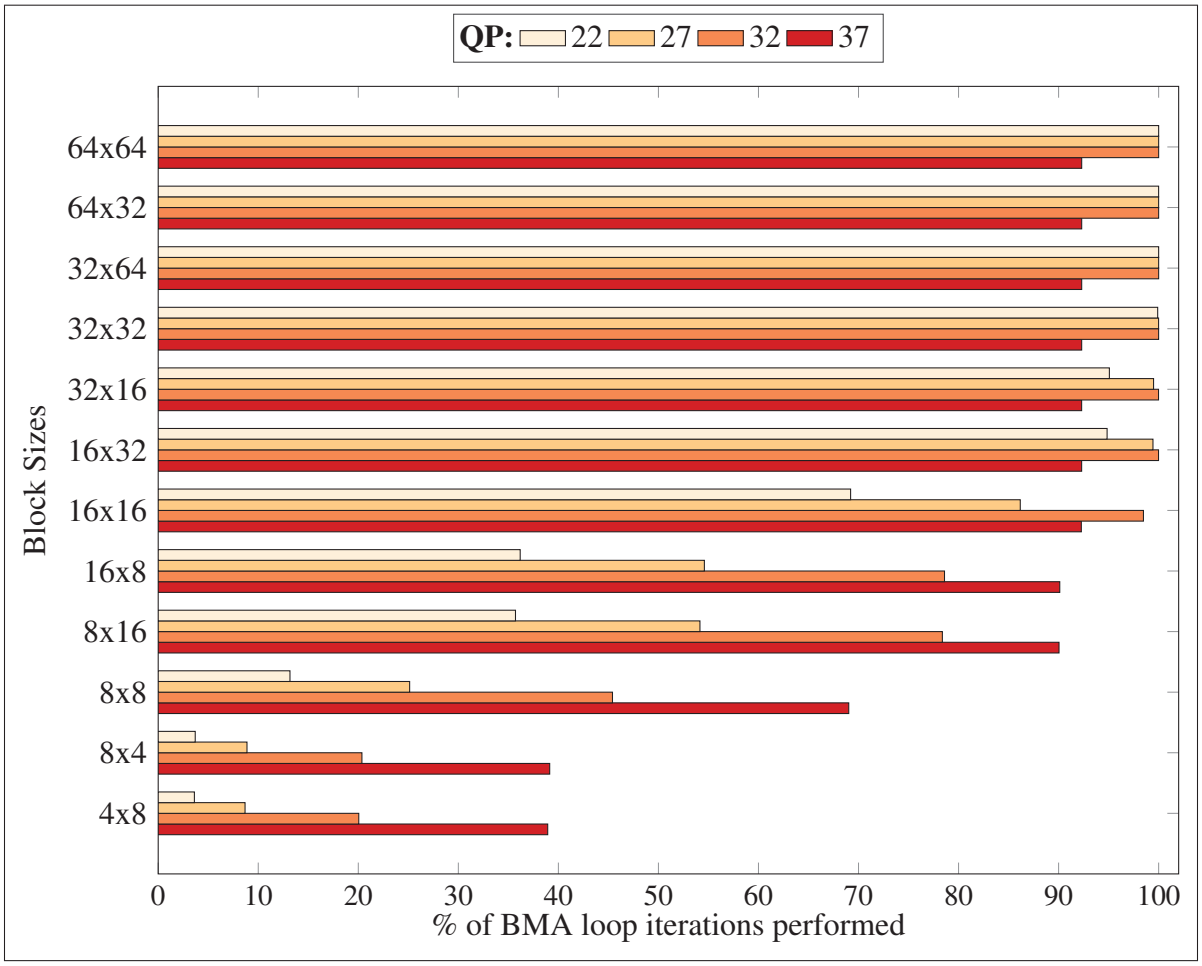


Figure 5.12 Block-wise results for the percentage of iterations performed by the block-matching loop in the proposed solution. A value of 100% indicates that all the iterations of the loop are performed.

5.6.3.4 Comparison with Suboptimal Algorithms

When compared to the suboptimal TZ-Search algorithm found in the HEVC HM 16.2 encoder software (McCann *et al.*, 2014), HM-CBSEA is about 5× slower, as shown in fig. 5.13. HM-CBSEA is itself 5× faster than HM-FS, as presented in Tables 5.7 and 5.8. In a situation where

the increase in bit rate resulting from a suboptimal algorithm is unacceptable, the proposed algorithm offers the same results as an exhaustive search.

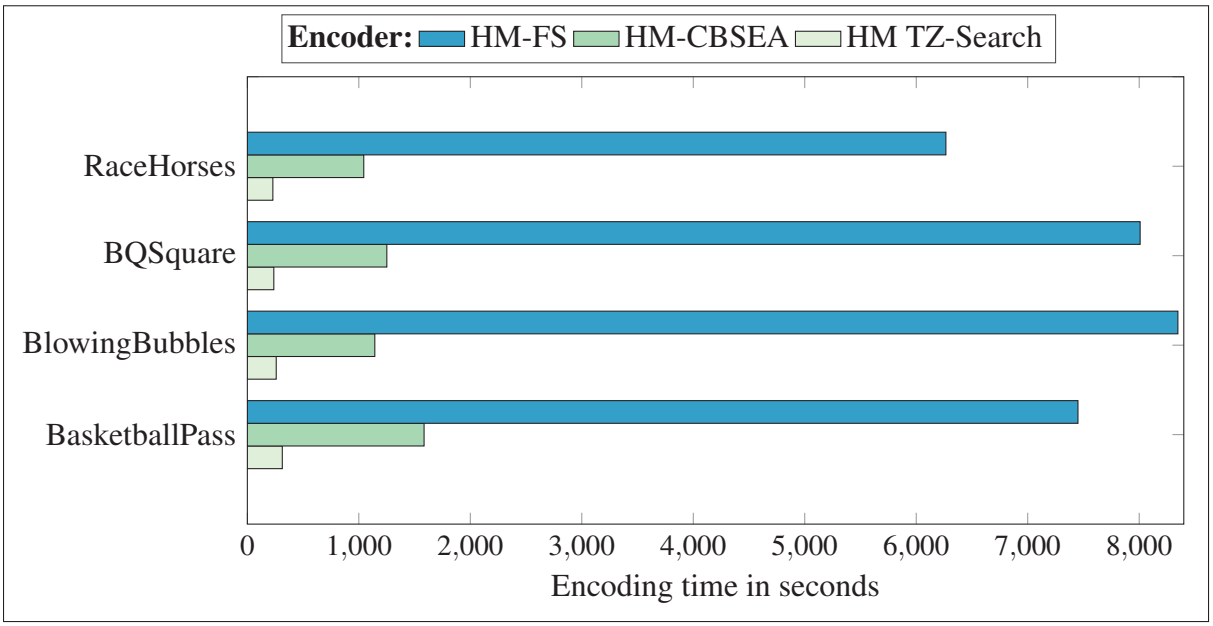


Figure 5.13 Encoding time in seconds, for version 3, when compared to HM TZ-Search, for class D sequences using the main profile and RA settings.

In this chapter, we presented different approaches that reorder candidates during motion estimation and that by doing so reduce the number of SAD operations performed. In the next chapter, we also propose to reduce the number of search operations performed during motion estimation, but this time by enhancing the rate-constraint.

CHAPTER 6

ENHANCED RATE CONSTRAINT

Inspired by the SEA, we developed our own lower bound on which to perform elimination. The novelty of this lower bound is that it reuses information obtained from the partitioning. Not only does it allow to eliminate candidates, but if it used with a cost-based search ordering, it can serve as a criterion to terminate the search. In this chapter, we describe how information is reused when partitioning blocks and how this information can enhance early termination.

6.1 Information Reuse when Partitioning Blocks

Conventional approaches evaluate a block before evaluating its partitions. For example, in the HEVC reference implementation software encoder (McCann *et al.*, 2014):

$$\mathbf{S}^k \rightarrow \mathbf{V}^k \rightarrow \mathbf{H}^k \,.$$

This ordering is well-suited for suboptimal algorithms, since block level heuristics determine if the partitioning can be skipped. However, in the context of an optimal encoding, all partitions are evaluated. As such, this ordering offers no advantages. More favorable orderings are

$$\mathbf{V}^k \rightarrow \mathbf{H}^k \rightarrow \mathbf{S}^k \text{ and } \mathbf{H}^k \rightarrow \mathbf{V}^k \rightarrow \mathbf{S}^k$$

as they allow the reuse of search information from partitions when performing the search at the block level.

The most obvious form of information reuse from the partitions to the block can be performed as follows:

$$\mathrm{SAD}(\mathbf{S}^k, \mathbf{v}) = \mathrm{SAD}(\mathbf{V}^k, \mathbf{v}) + \mathrm{SAD}((2^{k-1}, 0) + \mathbf{V}^k, \mathbf{v}) \,, \tag{6.1}$$

or

$$\mathrm{SAD}(\mathbf{S}^k, \mathbf{v}) = \mathrm{SAD}(\mathbf{H}^k, \mathbf{v}) + \mathrm{SAD}((0, 2^{k-1}) + \mathbf{H}^k, \mathbf{v}) \,. \tag{6.2}$$

The previously computed SADs of both rectangular partitions at a given position can be summed up to obtain the SAD of the square block at that position. This is valid if we assume that the search areas remain constant between the block and its partitions. However, this assumption does not always hold, as the motion vector prediction can change between partitions.

This being said, the main limiting factor of this approach is the SEA. As previously explained, the SEA filters out many SAD operations, which are therefore not available for reuse. Although it is possible to manage missing SADs, the high percentage of SAD operations filtered out by the SEA, combined with the management overhead, makes such an approach impractical.

One useful piece of information that is unaffected by the SEA is the minimum SAD, defined as:

$$\mathrm{MinSAD}(\mathbf{P}) \triangleq \min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c})\,. \tag{6.3}$$

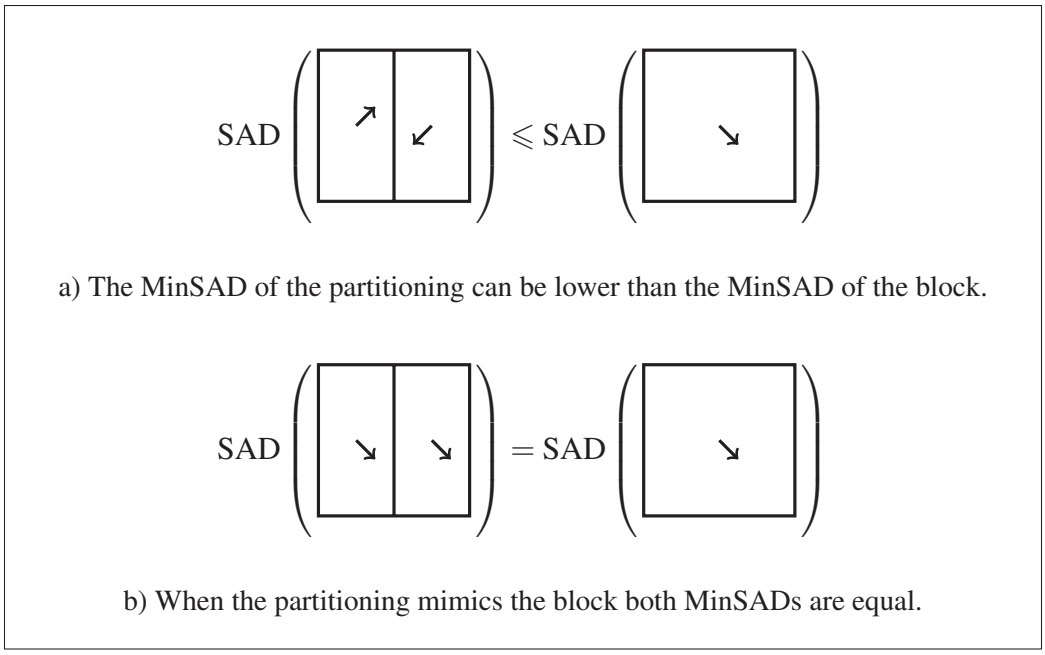
This information is useful because, summing up the MinSADs of the partitions of either the vertical rectangular partitioning or the horizontal rectangular partitioning will yield a lower bound for the MinSAD of the block:

$$\mathrm{MinSAD}(\mathbf{S}^k) \geqslant \mathrm{MinSAD}(\mathbf{V}^k) + \mathrm{MinSAD}((2^{k-1},0)+\mathbf{V}^k)\,, \tag{6.4}$$

$$\mathrm{MinSAD}(\mathbf{S}^k) \geqslant \mathrm{MinSAD}(\mathbf{H}^k) + \mathrm{MinSAD}((0,2^{k-1})+\mathbf{H}^k)\,. \tag{6.5}$$

As shown in fig. 6.1, the minSAD of a partitioning is less than or equal to the minSAD of the block. When the partitioning mimics the block both MinSADs are equal.

These inequalities are valid if we assume that the candidates that minimize the partition are also in the search area of the block itself. This is more reasonable than assuming the search areas are constant.



a) The MinSAD of the partitioning can be lower than the MinSAD of the block.

b) When the partitioning mimics the block both MinSADs are equal.

Figure 6.1 Visualization of LowSAD, a lower bound for blocks that is based on the reuse of MinSADs from their partitioning.

Let LowSAD be the lower bound of the MinSAD($\mathbf{S}^k$), such that:

$$\begin{aligned}\text{LowSAD}(\mathbf{S}^k) = \max(&\text{MinSAD}(\mathbf{V}^k) + \text{MinSAD}((2^{k-1},0) + \mathbf{V}^k),\\ &\text{MinSAD}(\mathbf{H}^k) + \text{MinSAD}((0,2^{k-1}) + \mathbf{H}^k)).\end{aligned} \tag{6.6}$$

By using the highest lower bound, we have a tighter lower bound to the MinSAD($\mathbf{S}^k$).

Per the definition of LowSAD and using Eq. (6.4) and Eq. (6.5), it follows that

$$\text{LowSAD}(\mathbf{S}^k) \leqslant \text{MinSAD}(\mathbf{S}^k) \leqslant \text{SAD}(\mathbf{S}^k, \mathbf{c}),\ \forall\, \mathbf{c} \in \mathbf{C}\,. \tag{6.7}$$

As such, no SAD in the search area will be less than LowSAD($\mathbf{S}^k$). This fact can be used to enhance early termination.

6.2 Improved Early Termination

As shown in section 5.3.1, ordering the candidates by rate transforms the rate constraint into an early termination mechanism. As illustrated in fig. 6.3, the rate constraint not only weighs the SAD and the rate function ($\mathrm{R}(\mathbf{v})$), but it also determines the size of the search area.

This can be made more explicit by isolating the rate from Eq. (5.13) as follows

$$\mathrm{R}(\mathbf{v}) \geqslant \frac{\mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}})}{\lambda} + \mathrm{R}(\hat{\mathbf{v}})\,. \tag{6.8}$$

When the rate is larger than the current best cost, the search can terminate without evaluating the remainder of the search area, as the rate of all subsequent candidates will be equal or greater.

From this equation we can conclude that, for a cost-based search ordering, the size of the search area is inversely proportional to λ. Recall that λ grows exponentially with the QP, as such the relationship between the QP and the rate threshold is shown in fig. 6.2.

It can also be seen in fig. 6.2 that the SAD of the candidate with the best cost will also influence the size of the search area. In fig. 6.2, all the curves converge towards $\mathrm{R}(\hat{\mathbf{v}})$. This follows naturally from the fact that the search area must include $\hat{\mathbf{v}}$.

Each curve in fig. 6.2 represents a different value of $\mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}})$. This value is an important factor in the size of the search area. Elements that can influence the SAD of the best candidate is the size of the block and how good of a match can be found in the search area.

Notice that an important element is missing in Eq. (6.8): $\mathrm{SAD}(\mathbf{P},\mathbf{v})$. The $\mathrm{SAD}(\mathbf{P},\mathbf{v})$ is absent because no assumptions can be made about the SAD of subsequent candidates beyond the fact that they are greater than or equal to zero. As such, we can view the equation as

$$\mathrm{R}(\mathbf{v}) \geqslant \frac{\mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) - 0}{\lambda} + \mathrm{R}(\hat{\mathbf{v}})\,. \tag{6.9}$$

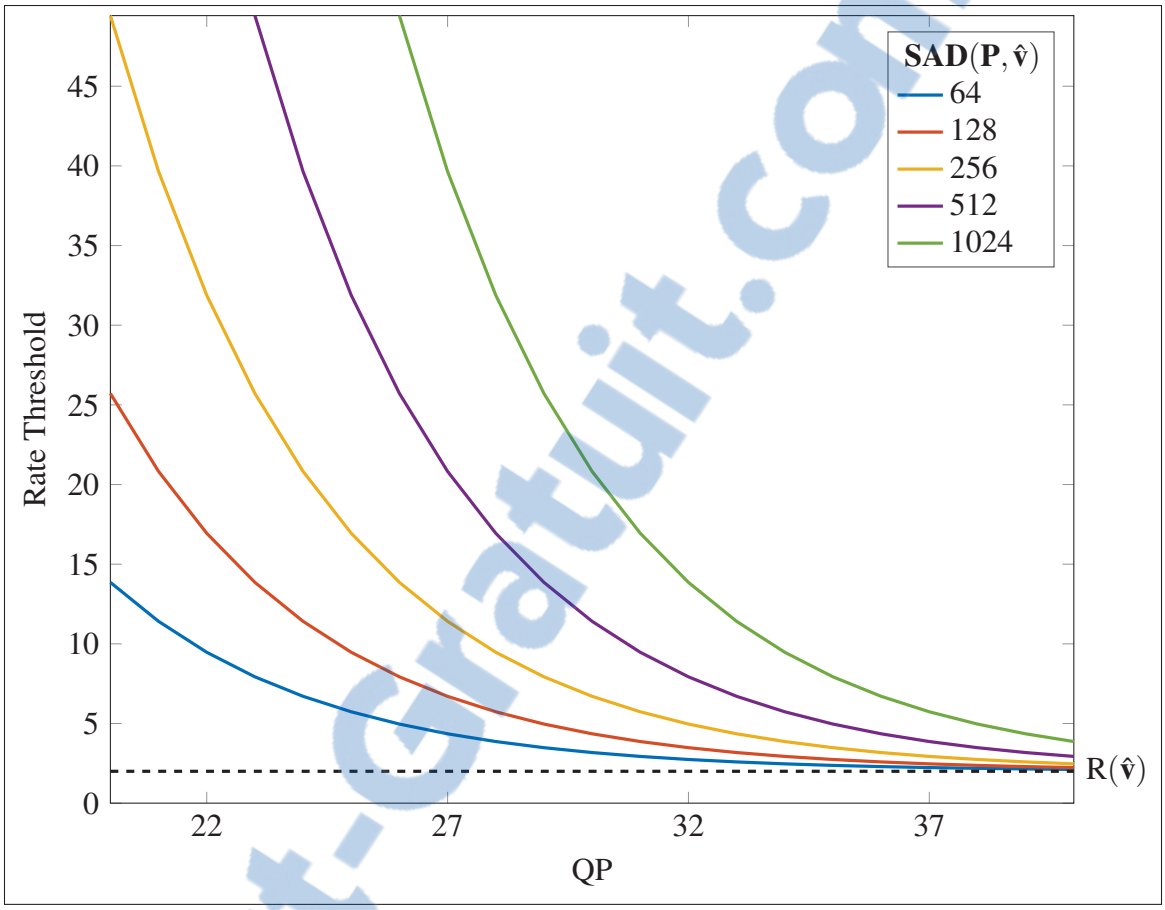


Figure 6.2 Relationship between the rate threshold and SAD($\mathbf{P}, \hat{\mathbf{v}}$) over the recommended range of QPs for HEVC.

In other words, when the rate-constraint is used as an early termination criteria, it assumes that a SAD value of zero is possible. This delays the termination of the search algorithm. Although zero might be possible in theory, for natural video sequences using lossy compression it is extremely unlikely.

A lower bound of the SAD greater than zero would permit early termination. This is indeed what we propose for $\mathbf{S}^k$. In Eq. (6.7), we demonstrated that $\text{LowSAD}(\mathbf{S}^k) \leqslant \text{SAD}(\mathbf{S}^k, \mathbf{c})$, $\forall\, \mathbf{c} \in \mathbf{C}$. Therefore, for $\mathbf{S}^k$, termination occurs when

$$\text{R}(\mathbf{v}) \geqslant \frac{\text{SAD}(\mathbf{S}^k, \hat{\mathbf{v}}) - \text{LowSAD}(\mathbf{S}^k)}{\lambda} + R(\hat{\mathbf{v}})\,. \tag{6.10}$$

Per Eq. (6.7), when Eq. (6.10) holds, the candidate cannot be an optimal solution, as its weighted rate ($\lambda \mathrm{R}(\mathbf{v})$) added to the lower bound of the $\mathrm{SAD}(\mathbf{S}^k, \mathbf{v})$ exceeds the current minimum rate-constrained SAD. In other words

$$\mathrm{LowSAD}(\mathbf{S}^k) + \lambda R(\mathbf{v}) \geqslant \mathrm{SAD}(\mathbf{S}^k, \hat{\mathbf{v}}) + \lambda \mathrm{R}(\hat{\mathbf{v}}) \tag{6.11}$$

$$\implies \mathrm{SAD}(\mathbf{S}^k, \mathbf{v}) + \lambda R(\mathbf{v}) \geqslant \mathrm{SAD}(\mathbf{S}^k, \hat{\mathbf{v}}) + \lambda \mathrm{R}(\hat{\mathbf{v}}) \tag{6.12}$$

According to the increasing rate rule, the algorithm can terminate; all subsequent candidates cannot be optimal solutions, as they cannot have a SAD lower than $\mathrm{LowSAD}(\mathbf{S}^k)$.

$$\mathbf{F}^{\complement} \cap \underset{\mathbf{c} \in \mathbf{C}}{\arg\min}\ \mathrm{SAD}(\mathbf{P}, \mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) = \emptyset \tag{6.13}$$

$$\implies \underset{\mathbf{c} \in \mathbf{F}}{\arg\min}\ \mathrm{SAD}(\mathbf{P}, \mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) = \underset{\mathbf{c} \in \mathbf{C}}{\arg\min}\ \mathrm{SAD}(\mathbf{P}, \mathbf{c}) + \lambda \mathrm{R}(\mathbf{c}) \tag{6.14}$$

Figure 6.3 clearly demonstrates that the LowSAD is a powerful concept that considerably enhances early termination.

The size of the search area is not determined by SAD of the best candidate, but by the difference between the SAD of the best candidate during partitioning and the current best SAD. On average, these values should be rather close, thus the difference will be rather small. Recall fig. 6.2, it clearly demonstrates that the closer the value of $\mathrm{LowSAD}(\mathbf{S})$ is to $\mathrm{MinSAD}(\mathbf{S})$, the sooner the termination in Eq. (6.10) occurs, compared to Eq. (6.8).

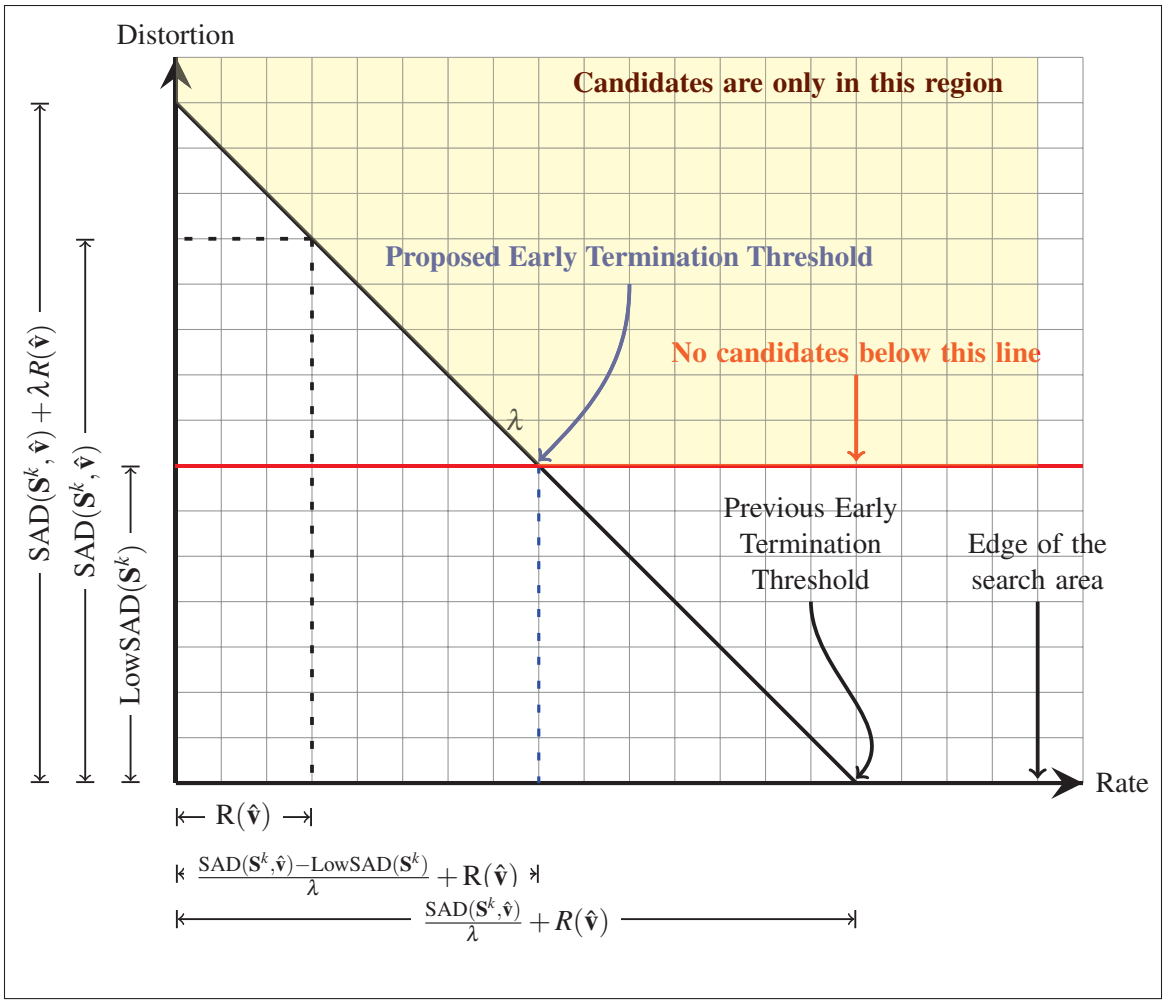


Figure 6.3 Geometric representation of the early termination thresholds.

6.3 Experimental Results and Discussion

To evaluate the partition reuse approach presented in chapter, we used the common test conditions and software reference configurations of the JCT-VC (Bossen, 2013). The encoder software runs the main profile with 8-bit coding and Low Delay P settings. The only changes to the standard configuration files are to enable full search, and to disable FEN and AMP. We disable the latter only to simplify the implementation of the proposed methods, but AMP and RCSEA are compatible. All tests were performed on the first 100 frames of the sequences specified by Bossen (Bossen, 2013), for classes: B (1920×1080), C (832×480), and D (416×240).

6.3.1 Comparison with HEVC HM Full Search

In table 6.1, we compare the SAD savings, the encoding time speedup and the BD-PSNR (Bjøntegaard, 2001) of the unmodified HM reference encoder, version 16.6, against the proposed solution also implemented in version 16.6 of the HM reference encoder. The values shown are averaged from the results for QPs: 22, 27, 32 and 37. The speedup is measured as the ratio between the encoding time of the unmodified HM reference software (T_{HM}) and the encoding time of the HM reference software with the proposed solution (T_{Proposed}):

$$\text{Speedup} = \frac{T_{\text{HM}}}{T_{\text{Proposed}}}\,. \tag{6.15}$$

The SAD savings are measured as the relative difference between the number of SAD operations of the HM reference encoder (#SAD$_{\text{HM}}$) and the number of SAD operations of the proposed solution (#SAD$_{\text{Proposed}}$):

$$\text{SAD savings} = \frac{\#\text{SAD}_{\text{HM}} - \#\text{SAD}_{\text{Proposed}}}{\#\text{SAD}_{\text{HM}}}\,. \tag{6.16}$$

The proposed solution, implemented with the RCSEA in the HM reference encoder, eliminates 94% of SAD operations on average, and is approximately 6 times faster than the original HM encoder. We observed that the bit streams produced by the proposed solution and the unmodified HM software encoder are not identical. This is due to two factors: the ordering differences between same-cost candidates during ME and the ordering differences of same-cost partitioning shapes during the coding of the prediction unit (PU). In other words, when multiple global minimums exist, either when choosing an MV or when choosing a partitioning shape, neither encoders might pick the same one. That being said, as shown in table 6.1, the difference in BD-PSNR is negligible (less than 0.004 dB).

		Proposed vs HM		
Class	Sequence name	Speedup	SAD savings	BD-PSNR
B (1920×1080)	Kimono	6.30	96.7%	0.0006
	ParkScene	6.42	95.8%	0.0014
	Cactus	7.07	96.3%	0.0018
	BQTerrace	5.92	94.6%	-0.0020
	BasketballDrive	6.05	95.4%	0.0016
C (832×480)	RaceHorses C	4.73	92.7%	0.0011
	BQMall	6.70	95.5%	-0.0008
	PartyScene	4.68	91.6%	-0.0003
	BasketballDrill	5.59	95.4%	-0.0026
D (416×240)	RaceHorses	4.56	93.0%	-0.0030
	BQSquare	8.75	96.1%	0.0032
	BlowingBubbles	6.78	95.2%	-0.0020
	BasketballPass	6.18	95.4%	-0.0011
	Overall	6.13	94.9%	0.0002

Table 6.1 Comparison of the proposed solution with the HEVC HM reference encoder software (Prop. vs. HM).

6.3.2 Comparison with HM-CBSEA

Table 6.2 shows the encoding time speedup, the total SAD operation savings, and the SAD operation savings for square (**S**) PUs for the partition reuse approach. Compared to the proposed solution to a HM-CBSEA, and implemented in version 16.6 of the HM reference encoder. We did not compare it to (Trudeau *et al.*, 2015a), as speedups would be biased because of the time required to sort candidates by ascending ADS.

Compared to HM-CBSEA, the proposed solution eliminates, on average, 19.8% more SAD operations, which is directly attributable to the 63.7% savings of square SAD operations, resulting in a speedup of approximately 1.23. Square blocks are twice the size of their rectangular counterparts. As a result, square SADs are, more or less, twice as big as rectangular SADs. From Eq. (6.8), it therefore follows that early termination requires twice the rate. Because of exponential Golomb codes, doubling the rate exponentially increases the size of the search area. However, based on an assumption of spatial-temporal correlation, we can assume that doubling the rate also exponentially increases the efficiency of rate-constrained

		Proposed vs HM-CBSEA		
Class	Sequence name	Speedup	SAD savings	**S** SAD savings
B (1920×1080)	Kimono	1.15	14.9%	45.6%
	ParkScene	1.35	25.7%	79.4%
	Cactus	1.27	21.8%	67.9%
	BQTerrace	1.36	26.3%	81.9%
	BasketballDrive	1.23	20.2%	64.0%
C (832×480)	RaceHorses C	1.13	14.8%	50.2%
	BQMall	1.18	16.0%	53.1%
	PartyScene	1.27	19.9%	66.2%
	BasketballDrill	1.24	19.3%	61.0%
D (416×240)	RaceHorses	1.15	12.9%	43.1%
	BQSquare	1.34	27.6%	90.4%
	BlowingBubbles	1.22	20.7%	68.1%
	BasketballPass	1.20	17.8%	56.9%
	Overall	1.23	19.8%	63.7%

Table 6.2 Comparison of the proposed solution with the HM-CBSEA (Prop. vs. HM-CBSEA).

transitive elimination. From this, we can expect the number of SAD operations to double. Thus, the weighted ratio of square SAD operations is about one third of rectangular SADs (a good approximation of the savings observed in Table 6.2).

Figure 6.4 shows that the square SAD operation savings increase when the QP increases. This is in line with the findings of Coban and Mersereau (Coban and Mersereau, 1998) to the effect that an increase in the Lagrange multiplier has a direct impact on transitive elimination in a rate-constrained context. As demonstrated, this property still holds for the proposed solution.

In this chapter, we presented a novel lower bound to the SAD based information reuse between partitions. Like in the previous chapter, this lower bound can reduce the number of SAD operations performed during motion estimation search. We also introduced an early termination criterion that be used when the search ordering follows the increasing rate rule.

From our literature review on motion estimation, skipping candidates and early termination is nothing new for suboptimal ME algorithms. However, these orderings are not compatible

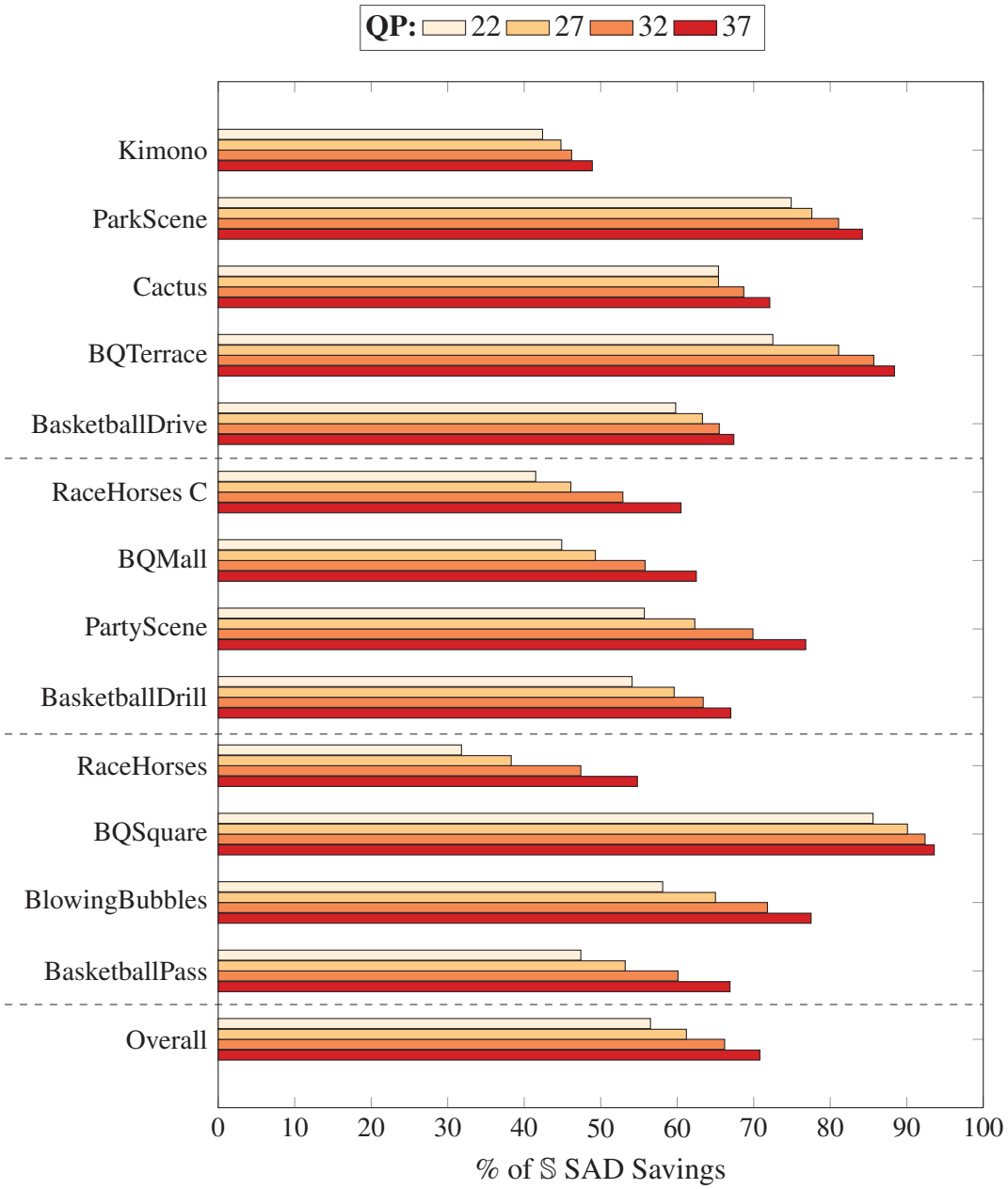


Figure 6.4 Percentage of square SAD operation savings, per sequence, for the proposed solution, when compared to HM-CBSEA.

with optimality preserving approaches. An RCSEA using a dynamic search ordering from a suboptimal algorithm would have to either return a suboptimal value or avoid early termination.

Dynamic search ordering and early termination for optimality preserving algorithms is new and noteworthy. By dynamically satisfying the increasing rate rule, the proposed algorithm not only allows for early termination, but it also guarantees that the optimal candidate is returned.

In the next chapter, we combine our findings with the TZ-Search algorithm. This allows to speed up the TZ-Search without introducing more prediction error as the candidates that are eliminated cannot minimize the current best minimum.

CHAPTER 7

MULTI-LEVEL RATE-CONSTRAINED SUCCESSIVE ELIMINATION ALGORITHM IN SUB-OPTIMAL SEARCH ALGORITHMS

Historically, SEAs have been targeted at exhaustive search algorithms. The contributions described in the two previous chapters improved the state of the art for optimal but not exhaustive search orderings. Our findings allow exhaustive search algorithms to consider fewer candidates and terminate earlier. In this chapter, we use SEA algorithms in the context of suboptimal search algorithms, more precisely the current state of the art, the TZ-Search algorithm. We first start by justifying the use of SEA in the TZ-Search and in the following section, we describe the proposed multi-level rate-constrained successive elimination algorithm (ML-RCSEA).

7.1 Justification for SEA in TZ-Search

SEA-based algorithms are targeted at ESA because of the a priori computation cost of sum buffers. Although the TZ-Search algorithm will not require the same amount of SAD operations for each block, on average, the number of evaluated candidates is high enough to justify the use of the proposed ML-RCSEA approach.

To demonstrate this, we will establish a minimum SAD operation saving threshold. Let $\mathscr{O}^{\mathrm{SAD}}$ be the number of operations required to compute the SAD, such that:

$$\mathscr{O}^{\mathrm{SAD}}(\mathbf{P}) = 3 \times |\mathbf{P}| - 1\,, \tag{7.1}$$

where $|\mathbf{P}|$ is the cardinality of the partition $\mathbf{P}$ (its number of pixels). Similarly to Eq. 11 in Coban and Mersereau (1998), let $\mathscr{O}^{\mathrm{RPS}}$ be the number of operations required to compute the

sum of all overlapping candidate blocks, for a reference frame of size $W \times H$, such that:

$$\mathscr{O}^{\mathrm{RPS}}(\mathbf{P}) = \underbrace{(2W - \mathrm{Cols}(\mathbf{P}) - 1)}_{\substack{\text{Sliding window} \\ \text{operations for} \\ \text{one row}}} H + \underbrace{(2H - \mathrm{Rows}(\mathbf{P}) - 1)}_{\substack{\text{Sliding window} \\ \text{operations for} \\ \text{one column}}} W\,, \tag{7.2}$$

where $\mathrm{Cols}(\mathbf{P})$ and $\mathrm{Rows}(\mathbf{P})$ respectively return the number of columns and the number of rows in $\mathbf{P}$.

For example, computing the sum of all overlapping 8×8 blocks in a 416×240 frame requires the same number of operations as 2060 8×8 SAD operations. This number might seem prohibitive, but given that a 416×240 frame contains 1560 8×8 blocks, it corresponds to an average of 1.32 SAD operation savings per block.

At sufficiently high resolutions, the ratio between the number of operations required to compute the reconstructed pixel sums and the number of operations required to compute the SAD can be approximated as

$$\frac{\mathscr{O}^{\mathrm{RPS}}(\mathbf{P})}{\mathscr{O}^{\mathrm{SAD}}(\mathbf{P})} \approx \frac{4WH}{3|\mathbf{P}|}\,. \tag{7.3}$$

This equation represents the number of SAD operations to compensate for the entire frame. At the block level, this can be approximated by a constant

$$\frac{\frac{4WH}{3|\mathbf{P}|}}{\frac{WH}{|\mathbf{P}|}} = \frac{4}{3}\,. \tag{7.4}$$

These approximations reveal qualitative insight that it is reasonable for the minimal SAD savings threshold to be $\frac{4}{3}$. Figure 7.1 shows that as the resolution increases, the SAD savings threshold tends towards $\frac{4}{3}$ for all blocks and partition sizes.

To compute the minimal SAD savings threshold for the whole frame, we sum the number of operations for all the HEVC block sizes except for $64 \times 64, 64 \times 32, 32 \times 64$, because for theses blocks, the ADS is summed from smaller blocks. The same applies for all the partition sizes

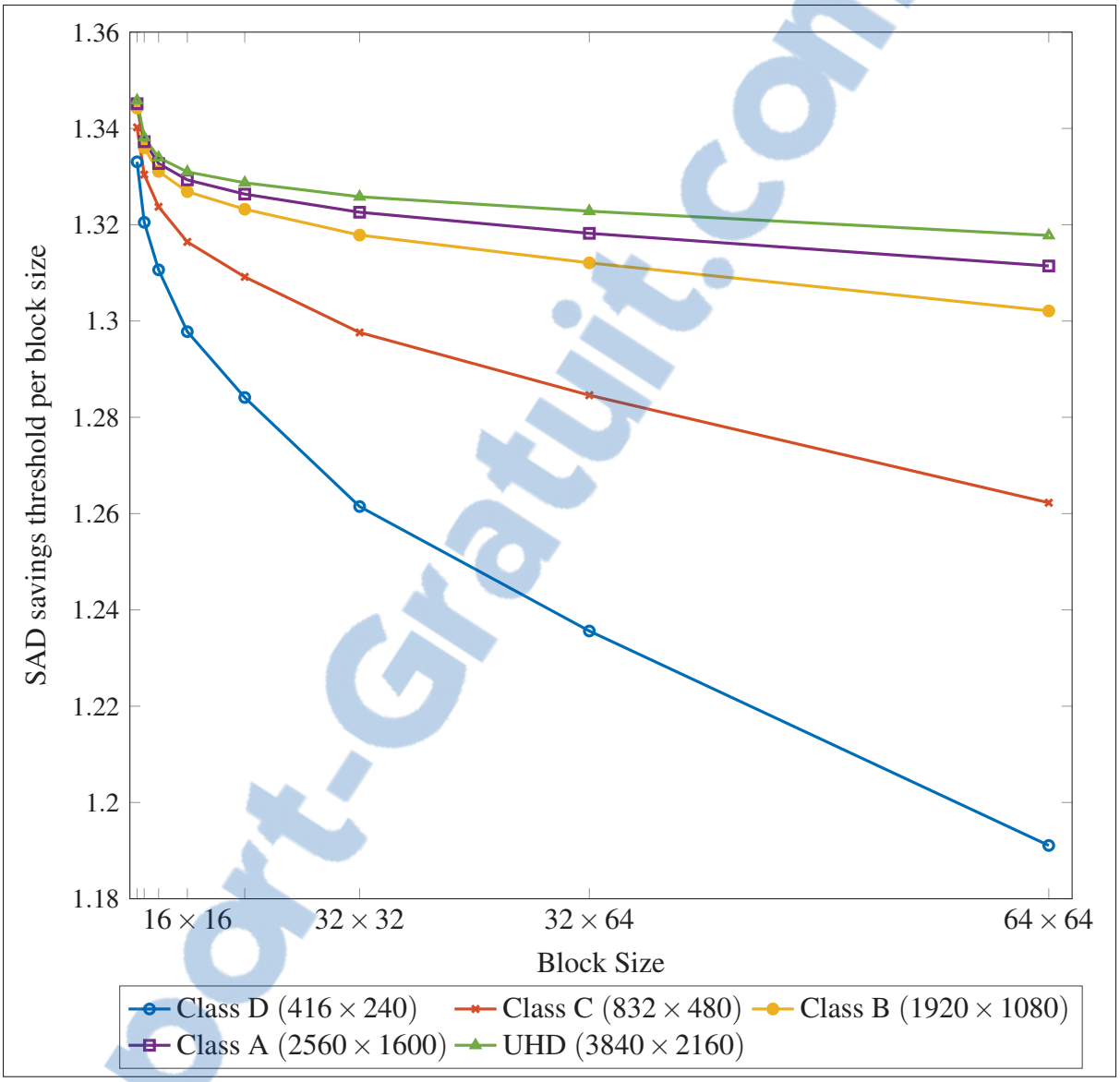


Figure 7.1 SAD savings threshold per block for HEVC block/partition sizes and for common frame resolutions.

related to AMP. Also note that, because of the hierarchical B-frame structure, for RA, the sum buffers are computed only for even numbered frames, as odd numbered frames cannot serve as reference frames (McCann *et al.*, 2014).

For a 832×480 frame, this threshold is about 54590 SAD operations. This might seem excessive, but as we describe in section 7.5.4, table 7.4 clearly demonstrates that the SAD

operation savings per frame of our proposed method are well above the minimum SAD operation saving threshold. Also note that the minimum SAD threshold is easily surpassed only by the savings on the TZ-Search, as such, the SAD savings for bi-predictive search require no additional pre-computation.

7.2 Multi-Level Composition Patterns

In this section, we present the multi-level composition patterns for rectangular and asymmetric partitioning. We propose to use multi-level composition to compute the ADS of an asymmetric partitioning from sub-partitions also present in symmetric partitioning. The advantage of using this pattern is that it does not require computing additional sum buffers or having to manage the complexities of offset and overlap.

7.2.1 Rectangular Partitions

In order to improve transitive elimination on larger rectangular partitions, we propose to split rectangular partitions into smaller rectangular sub-partitions. We do not split the smallest partitions, since the number of pixels is small and transitive elimination is already very efficient. As such, this approach is more efficient than the RCSEA and reduces the number of sum buffers required as the biggest rectangular sum buffers are not required.

As shown in fig. 7.2, we propose to split horizontal rectangular partitions $\mathscr{H}^k$ into four vertical rectangular sub-partitions $\mathbf{V}^{k-1}$ and vertical rectangular partitions $\mathscr{V}^k$ into four horizontal rectangular sub-partitions $\mathbf{H}^{k-1}$:

$$\mathscr{H}^k = \left\{ \begin{array}{ll} (0,0)+\mathbf{V}^{k-1}, & (2^{k-1},0)+\mathbf{V}^{k-1}, \\ (2^k,0)+\mathbf{V}^{k-1}, & (3\times 2^{k-1},0)+\mathbf{V}^{k-1} \end{array} \right\}, \tag{7.5}$$

$$\mathscr{V}^k = \left\{ \begin{array}{ll} (0,0)+\mathbf{H}^{k-1}, & (0,2^{k-1})+\mathbf{H}^{k-1}, \\ (0,2^k)+\mathbf{H}^{k-1}, & (0,3\times 2^{k-1})+\mathbf{H}^{k-1} \end{array} \right\}. \tag{7.6}$$

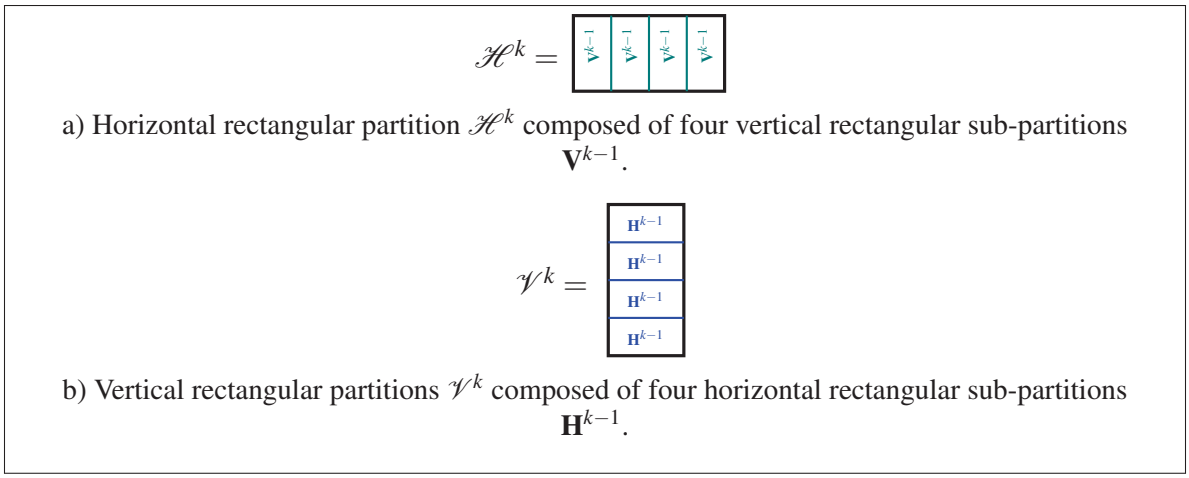


Figure 7.2 Proposed multi-level composition pattern for rectangular partitions.

7.2.2 Asymmetric Partitioning

We propose a multi-level composition pattern to perform ML-RCSEA on an asymmetric partitioning using pre-computed symmetric sub-partitions. This proposition resolves many of the issues encountered when combining asymmetric motion partition (AMP) with ML-RCSEA (i.e. offset and overlap). As can be seen in fig. 7.3, an offset occurs when partitions are positioned at coordinates that are not a multiple of their own size; whereas an overlap occurs when partitions of the same size overlap each other.

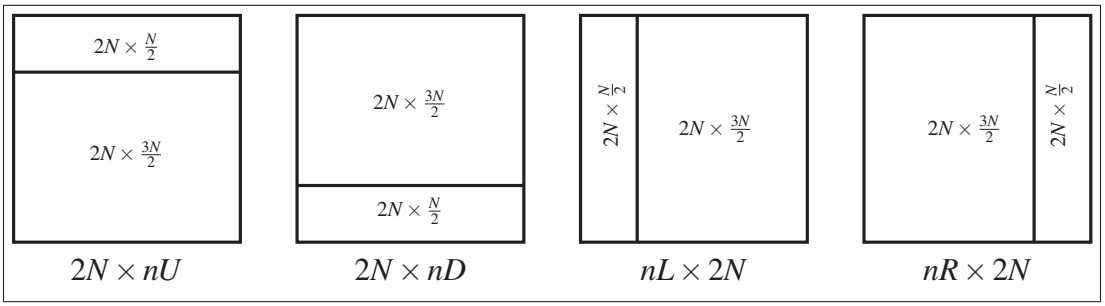


Figure 7.3 AMP used in HEVC. Multiple new partition sizes must be considered. Some partitions are offset and others overlap each other. These are all issues for ML-RCSEA.

This is not to say AMPs and ML-RCSEA are incompatible, it only highlights issues that make combining these two approaches non-trivial. Accessing sum buffers for asymmetric partitioning requires extra computational overhead to manage these issues. Also computing each of these sum buffers requires more pre-processing.

By applying the proposed multi-level compositional pattern, illustrated in fig. 7.4, an encoder using ML-RCSEA (i.e. for symmetric partitioning) can perform AMPs without computing additional sum buffers or having to manage the complexities of offset and overlap. The key concept behind our approach is to split asymmetric partitioning into partitions also used by symmetric partitioning.

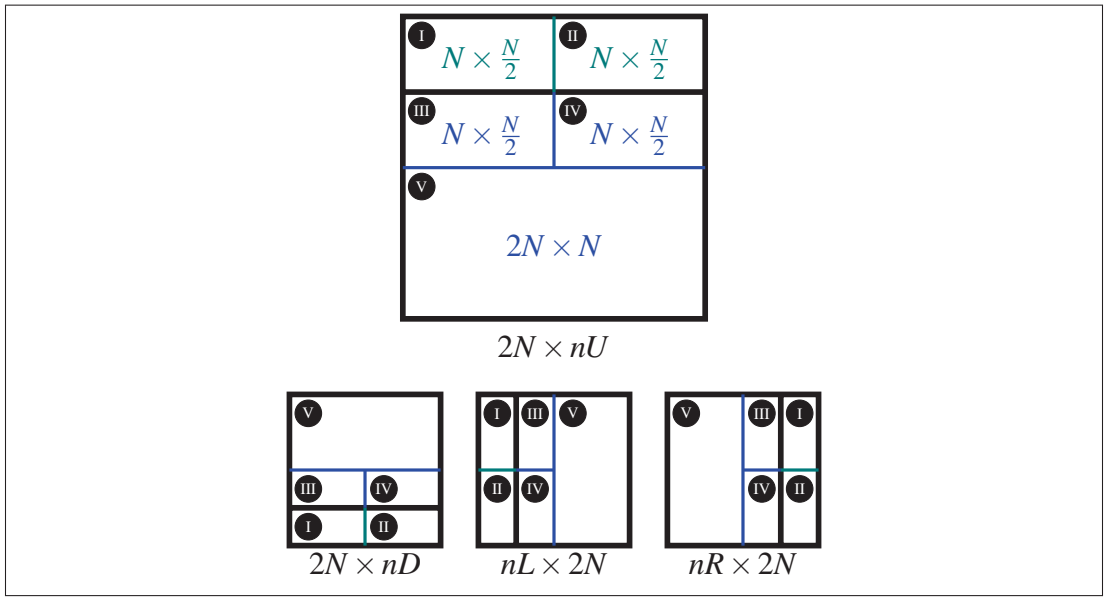


Figure 7.4 Proposed multi-level composition pattern to perform ML-RCSEA on an asymmetric partitioning using pre-computed symmetric sub-partitions.

To describe the proposed multi-level composition pattern, we use the $2N \times nU$ asymmetric partitioning scheme as an example (as such, its illustration is bigger in fig. 7.4). Recall fig. 7.3, the $2N \times nU$ partitioning scheme splits the PU into two partitions: the first of size $2N \times \frac{N}{2}$ and the second of size $2N \times \frac{3N}{2}$. As with symmetric partitioning, these partitions are evaluated separately.

We refer to the first partition as **U** and its compositional pattern as $\mathscr{U}$. We split this partition into two $N \times \frac{N}{2}$ partitions, identified with the labels ❶ and ❷ in fig. 7.4. An $N \times \frac{N}{2}$ partition of size k is equivalent to a $2N \times N$ partition of size $k-1$. A $2N \times N$ is a partition also used by symmetrical partitioning, $\mathbf{H}^{k-1}$, for which the sum buffer is already available. The compositional pattern of the $2N \times \frac{N}{2}$ partition is the combined sum of the two $2N \times N$ partitions of size $k-1$:

$$\mathscr{U}^k = \{\underbrace{(0,0)+\mathbf{H}^{k-1}}_{❶}, \underbrace{(2^{k-1},0)+\mathbf{H}^{k-1}}_{❷}\} \tag{7.7}$$

We refer to the second partition as **D** and its compositional pattern as $\mathscr{D}$. Just like the first partition, we split the top part of the $2N \times \frac{3N}{2}$ partition into two $N \times \frac{N}{2}$ partitions, identified with the labels ❸ and ❹ in fig. 7.4. This split must be performed on the part of the partition that is next to the $2N \times \frac{N}{2}$ partition, as the remainder will be a symmetrical $2N \times N$, identified with the label ❺. The compositional pattern of the $2N \times \frac{3N}{2}$ partition is the combined sum of the two $2N \times N$ partitions of size $k-1$ and the $2N \times N$ partition of size k.

$$\mathscr{D}^k = \{\underbrace{\mathscr{U}^k}_{❸,❹} \cup \underbrace{(0,2^{k-1})+\mathbf{H}^k}_{❺}\} \tag{7.8}$$

The labels show the corresponding splits for the other asymmetric partitioning schemes.

7.3 Double-check Mechanism for RCSEA in TZ-Search

Recall the RCSEA, proposed in Coban and Mersereau (1998), which applies the SEA to the cost function

$$\mathrm{J}(\mathbf{P},\mathbf{v}) = \mathrm{SAD}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{v})\,. \tag{7.9}$$

As we have already shown, in a rate-constrained context, transitive elimination works as follows

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda R(\mathbf{v}) > \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) + \lambda R(\hat{\mathbf{v}}) \tag{7.10}$$

$$\implies \mathbf{v} \notin \arg\min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda R(\mathbf{c})\,. \tag{7.11}$$

Although transitive elimination reduces the number of SAD being computed, the $\mathrm{R}(\mathbf{v})$ function must be computed for every candidate. In many cases, the value of $\mathrm{R}(\mathbf{v})$ does not alter the outcome of the transitive elimination performed by RCSEA (Eq. (7.11)). Depending on how $\mathrm{R}(\mathbf{v})$ is defined, it can be a costly operation.

We propose a double-check mechanism to avoid useless $\mathrm{R}(\mathbf{v})$ computations (i.e. when $\mathrm{R}(\mathbf{v})$ does not alter the outcome of Eq. (7.11)). It is important to note that one cannot just ignore the rate term when performing the RCSEA, as it will lead to false positive eliminations. We propose to extend Eq. (7.11) as follows:

$$\mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda \mathrm{R}(\mathbf{p}) > \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) + \lambda R(\hat{\mathbf{v}}) \tag{7.12}$$

$$\implies \mathrm{ADS}(\mathbf{P},\mathbf{v}) + \lambda R(\mathbf{v}) > \mathrm{SAD}(\mathbf{P},\hat{\mathbf{v}}) + \lambda R(\hat{\mathbf{v}}) \tag{7.13}$$

$$\implies \mathbf{v} \notin \arg\min_{\mathbf{c}\in\mathbf{C}} \mathrm{SAD}(\mathbf{P},\mathbf{c}) + \lambda R(\mathbf{c})\,. \tag{7.14}$$

where $\mathbf{p}$ is the position of the predicted motion vector. Note that $\mathbf{p}$ is the smallest motion vector cost in the search area

$$\mathbf{p} \in \arg\min_{\mathbf{v}\in\mathbf{C}^r} \mathrm{R}(\mathbf{v})\,. \tag{7.15}$$

When the inequality of Eq. (7.12) holds then, by transitivity, the candidate can safely be eliminated without computing $\mathrm{R}(\mathbf{v})$.

Concretely, $\mathrm{R}(\mathbf{p})$ is computed for the MVP and then reused as a first step for every other candidate in the search area. In the context where the motion vector costs are approximated (i.e. HEVC), saving a considerable amount of motion vector cost computation allows for the

use of a more precise, yet more complex approximation of motion vector costs contributing to increasing the rate-distortion performance.

7.4 Cost-Based Search Ordering for Bi-Predictive Search

Bi-prediction consists of building a prediction from two motion vectors. In this section, we propose to adapt our prior work on cost-based search ordering to the bi-predictive search.

In principle, a bi-predictive motion search algorithm could minimize the prediction error by searching for both motion vectors simultaneously. Because of the combinatorial aspect of this problem, modern encoders like the HM resort to a greedy approach that uses an iterative uni-predictive search for the first motion vector, followed by a refined search for the second motion vector (McCann *et al.*, 2014).

This refinement is performed using a raster search algorithm over a small part of the search area. A raster search is used because the refinement surface is rather small (9×9). However, as shown in fig. 7.5, our proposed enhancements to the TZ-Search algorithm have reduced the number of SAD operations it produces to such a point that the bi-predictive raster scan is now the dominant source of SAD operations performed by the HM.

A raster search ordering results in a rather limited number of candidates being eliminated by transitivity. As explained in Cai and Pan (2010); Trudeau *et al.* (2014, 2015a), the search ordering plays a crucial role on the efficiency of transitive elimination. Finding good candidates early on lowers the transitive threshold used for elimination. This leads to a considerably more efficient transitive elimination than when good candidates are farther positioned in the search ordering.

SEA-based algorithms will often use a search ordering based on a spiral shaped geometric pattern. This follows the assumption that the likelihood of finding the best-matched candidate would decrease with the increase in cost of the MV (Cai and Pan, 2010). However, as explained in Trudeau *et al.* (2016a), these small refinement search areas are prone to be off-centered.

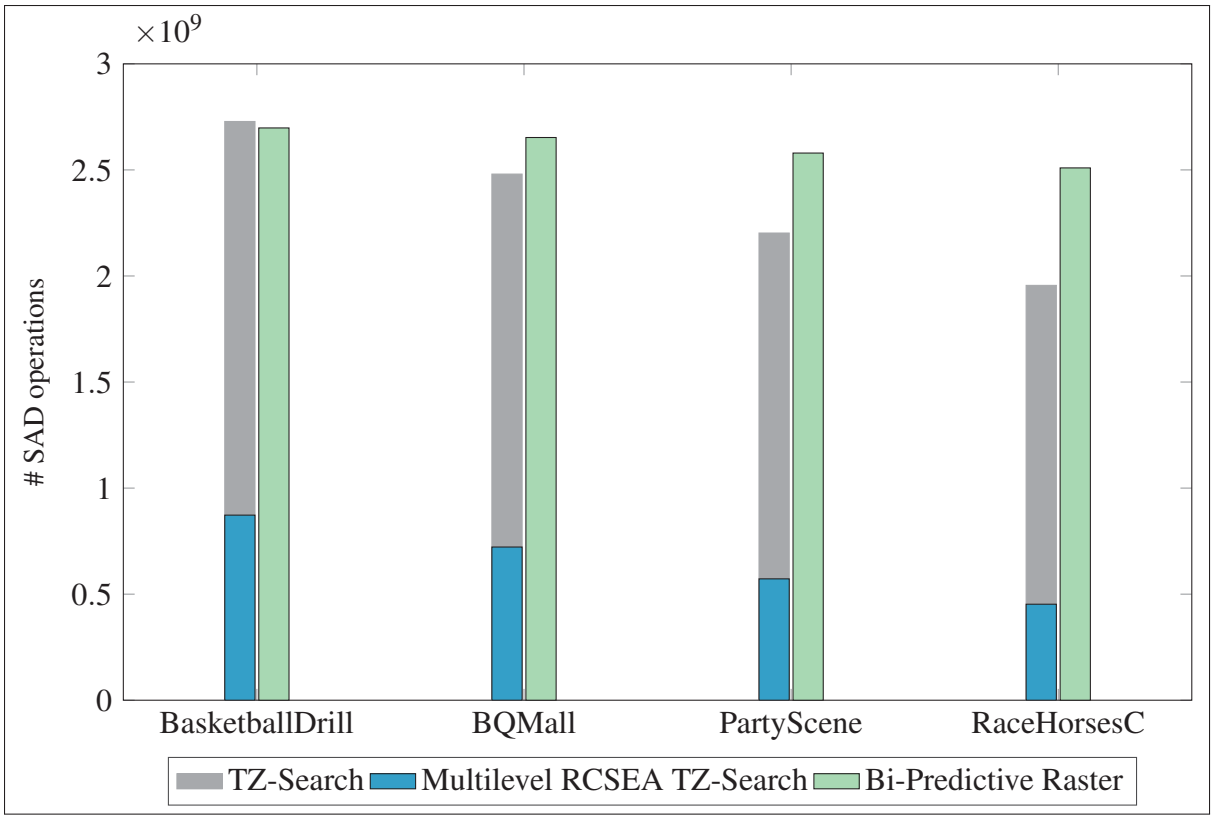


Figure 7.5 Comparison of the number of SAD operations performed by the HM. After introducing ML-RCSEA, bi-predictive raster refinement now becomes the predominant source of SAD operations.

When used on an off-centered search area, a search ordering based on a static geometric pattern will not always follow the previous assumption as the center of the refinement zone might not be aligned with the predicted motion vector.

This situation is ideal for the cost-based search ordering approach we proposed in Trudeau *et al.* (2016a). This search ordering is dynamic and adapts to off-centered search areas. It guarantees that candidates are evaluated by increasing motion vector costs. As such, the previous assumption is respected, thus increasing the likelihood of finding better candidates early on.

7.5 Experimental Results and Discussion

We implemented the ML-RCSEA in both versions 12.1 and 16.8 of the HM reference encoder (McCann *et al.*, 2014). Version 12.1 is only used in section 7.5.5. The test conditions and software configurations used in our experiments conform to the common test conditions and software reference configurations (Bossen, 2013). Our tests mainly focus on video sequences from classes B (1920×1080) and C (832×480), for the LD and the RA main profiles. The only change to the standard configuration files was to disable the FEN. We disabled the latter only to simplify implementation considerations, but FEN and SEA are compatible.

We use the encoding time reduction ratio (ETRR) and integer level motion estimation time reduction ratio (IMETRR) metrics defined in (Hu and Yang, 2014), such that:

$$\text{ETRR} = \frac{\text{ET}_{\text{HM}} - \text{ET}_{\text{AP}}}{\text{ET}_{\text{HM}}}, \tag{7.16}$$

where ET_{HM} is the encoding time of the HEVC HM reference encoder and ET_{AP} is the encoding time of the approach being evaluated. Similarly for integer-level motion estimation:

$$\text{IMETRR} = \frac{\text{IMET}_{\text{HM}} - \text{IMET}_{\text{AP}}}{\text{IMET}_{\text{HM}}}, \tag{7.17}$$

where IMET_{HM} is the integer-level motion estimation time of the HEVC HM reference encoder and IMET_{AP} is the integer-level motion estimation time of the approach being evaluated.

7.5.1 ML-RCSEA in TZ-Search

This section describes the impact of ML-RCSEA on the TZ-Search algorithm. In table 7.1, we present the percentage of SAD operations avoided by ML-RCSEA. In fig. 7.6, we compare the number of SAD operations performed by the TZ-Search algorithm with and without ML-RCSEA for each frame of the *BasketballDrill* sequence encoded using the LD Main profile and a QP of 22.

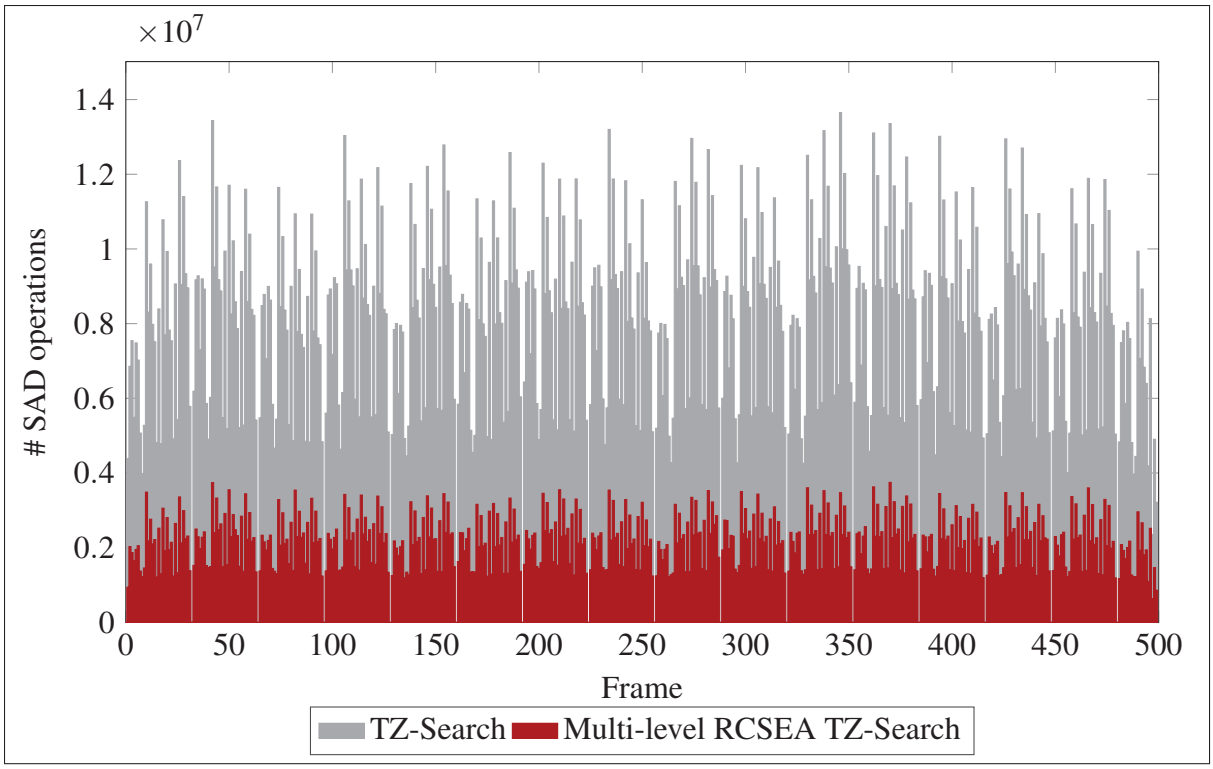


Figure 7.6 Comparison of the number of SAD operations performed by TZ-search with and without ML-RCSEA for the BasketballDrill sequence, QP=22 and using the LD Main profile.

Table 7.1 Percentage of SAD operations saved by ML-RCSEA in the TZ-Search (LD Main profile).

	QP			
Sequence name	22	27	32	37
BasketballDrill	70.94%	73.32%	76.32%	79.36%
BQMall	69.89%	72.39%	75.24%	77.93%
PartyScene	59.56%	60.26%	62.06%	64.46%
RaceHorsesC	71.94%	73.63%	75.45%	77.21%
Average	68.08%	69.90%	72.27%	74.74%

For each sequence in table 7.1, the percentage of SAD operation savings increases with the QP. This is caused by the fact that the weighting coefficient (λ in Eq. (2.4)) is dependent on the QP, as it reflects the desired trade-off between rate and distortion. As the QP increases,

the distance from the current best candidate plays an increasingly important role in transitive elimination.

7.5.2 Double-check Mechanism for RCSEA in TZ-Search

As explained in section 7.3, the motion vector cost must be taken into consideration when performing RCSEA. However, it often does not alter the outcome of transitive elimination. We proposed the double-check mechanism for RCSEA in section 7.3, which only computes the motion vector cost when an impact on transitive elimination is possible.

For quantitative results specific to the TZ-Search algorithm, we measured the percentage of cases where the double-check mechanism saved motion vector cost computations. Table 7.2 presents our findings. On average, in common test conditions (QPs: 22, 27, 32 and 37) the double-check mechanism avoided 58%, 53%, 47% and 40% motion vector cost computations, respectively. These motion vector cost computation did not impact transitive elimination nor did they impact the encoding process.

Table 7.2 Motion vector cost computational savings of the double-check mechanism implemented in TZ-Search with ML-RCSEA (LD Main profile).

	QP			
Sequence name	22	27	32	37
BasketballDrill	59.31%	53.91%	47.65%	41.34%
BQMall	59.76%	55.02%	49.08%	42.73%
PartyScene	52.23%	46.95%	40.03%	32.47%
RaceHorsesC	62.20%	57.49%	51.54%	45.54%
Average	58.37%	53.34%	47.07%	40.52%

It easily follows that the motion vector cost computational savings decrease as the QP increases. This is caused by the relationship between the weight coefficient (λ in Eq. (2.4)) and the QP. Increasing the weight of the motion vector cost in relation to the ADS makes it a discriminant factor in rate-constrained transitive elimination (Eq. (3.15)). Concretely, as the QP increases,

the motion vector cost needs to be computed more often. However, even when the QP is high, the double-check mechanism still avoids a non-negligible number of useless operations.

In regards to ETRR, the impact is negligible as motion vector costs computations represent less than 1% of the overall encoding. However, in the context where the motion vector costs are approximated (i.e. HEVC) and since RCSEA requires the computation of less than half of the motion vector costs; this could allow the use of a more precise, yet more complex approximation of motion vector costs contributing to increasing the rate-distortion performance.

7.5.3 Cost-Based Search Ordering for Bi-Predictive Search

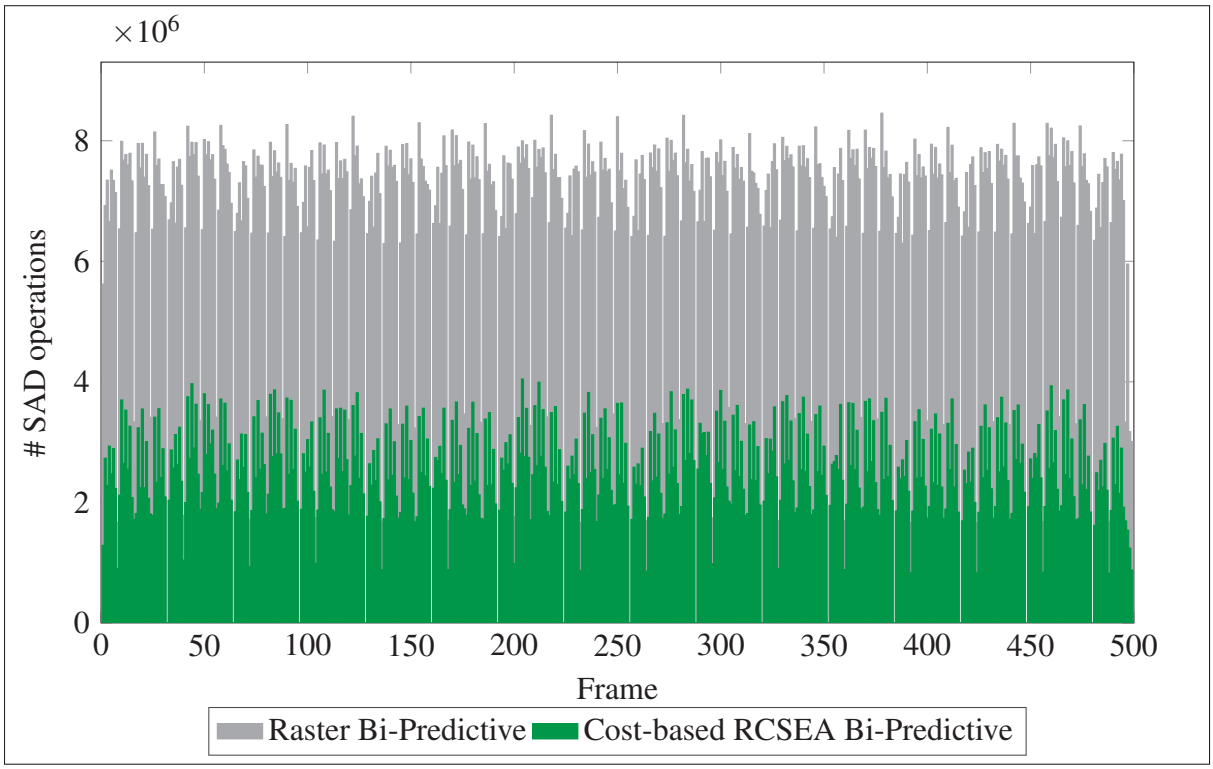


Figure 7.7 Comparison of the number of SAD operations performed by bi-predictive raster refinement compared with a cost-based RCSEA refinement for the BasketballDrill sequence, QP=22 and using the LD Main profile.

As shown in fig. 7.5, after applying the ML-RCSEA to TZ-Search, the bi-predictive raster search becomes the predominant source of SAD operations in the HM reference encoder. As explained in section 7.4, we propose to use a cost-based search ordering to maximize the efficiency of transitive elimination in the bi-predictive refinement search area. Table 7.3 shows the percentage of SAD operations saved by the proposed approach compared to an unmodified HM.

Table 7.3 Percentage of SAD operations saved by RCSEA with cost-based search ordering in bi-predictive refinement (LD Main profile).

	QP			
Sequence name	22	27	32	37
BasketballDrill	54.85%	56.12%	61.02%	67.96%
BQMall	59.96%	63.94%	68.18%	71.89%
PartyScene	50.83%	50.87%	51.80%	53.46%
RaceHorsesC	42.40%	45.46%	50.07%	55.62%
Average	52.01%	54.10%	57.77%	62.24%

As in sections 7.5.1 and 7.5.2, the SAD operation savings increase as the QP increases. Once again, this is caused by the weighting coefficient λ. Even if the refinement search area is small (9×9), the motion vector cost remains important. Because the predicted motion vector is not always the center of the search area, the area itself is small but the motion vector costs might not be. Also as explained in (Trudeau *et al.*, 2016a) the cost-based for ordering maximizes the impact of the rate constraint on transitive elimination.

7.5.4 Minimal SAD Savings Threshold

In section 7.1 we argued that the SAD operation savings of ML-RCSEA outweighed the pre-computational costs required for transitive elimination. We showed that for a given block size, computing the sum buffers required $\frac{4}{3}$ SAD operations per block size to encode the frame. The ETRR gains in tables 7.5 and 7.7 confirm this fact. This being said, in table 7.4 we show that the minimal SAD savings threshold is easily surpassed by the proposed solution.

Table 7.4 Average SAD operations saved per frame for the proposed solution for both TZ-Search and bi-predictive search using the LD Main profile. The required threshold is approximately 55000.

	QP			
Sequence name	22	27	32	37
BasketballDrill	12486437	12048817	11562327	11024823
BQMall	11321875	10893245	10442808	9990449
PartyScene	9268459	8776640	8210266	7675516
RaceHorsesC	18453584	17729079	16448261	14671255
Average	12882589	12361945	11665916	10840511

Recall from section 7.1 that the minimal SAD savings threshold for a class-C sequence is approximately 55000 SAD operations. The SAD savings are nearly 200 times larger than the minimum SAD threshold.

7.5.5 Comparison with State-Of-The-Art Methods

Starting from version 16 of the HM reference encoder, an important bug was fixed in the TZ-Search algorithm (Tourapis *et al.*, 2015). This fix considerably reduces the number of SAD operations considered by the TZ-Search algorithm and lowers its execution time. In order to compare with (Hu and Yang, 2014), we implemented the proposed solution in the same version of the HM reference encoder as the authors (i.e. version 12.1).

Tables 7.5 and 7.6 compare the ETRR, IMETRR and BD-Rate of the CIME method (Hu and Yang, 2014) with the proposed solution (i.e. ML-RCSEA for TZ-Search, double-check mechanism for RCSEA in TZ-Search and cost-based search ordering for ML-RCSEA for bi-predictive search). We introduce the bi-predictive refinement time reduction ratio (BRTRR). The BRTRR is taken into account in the ETRR but not in the IMETRR.

By comparing table 7.5 and table 7.7, we notice that the sequences where the CIME method produces the higher BD-Rate penalties are the sequences where we show the biggest speedups. For these sequences, the proposed solution offers both a good speedup and BD-Rate savings. For example, the proposed solution achieves almost 8% encoding time speed up and 1.1%

Table 7.5 Comparison of the proposed solution against CIME (Hu and Yang, 2014) with HM-12.1 for LD-Main (part 1)

		ETRR		BD-Rate(Y)	
Class	Sequence name	CIME	**Proposed**	CIME	**Proposed**
B (1920×1080)	BasketballDrive	16.62%	18.70%	1.36%	0.0207%
	Kimono	16.13%	23.87%	1.11%	-0.0930%
	ParkScene	11.05%	11.43%	0.60%	0.0088%
	Cactus	10.18%	16.79%	0.82%	0.0090%
	BQTerrace	10.65%	8.06%	0.31%	0.0288%
C (832×480)	BasketballDrill	11.79%	16.36%	1.63%	-0.0138%
	BQMall	11.38%	14.46%	0.64%	0.0219%
	PartyScene	9.21%	8.28%	0.72%	-0.0555%
	RaceHorsesC	20.46%	23.42%	1.58%	-0.1181%
	Average	13.05%	15.71%	0.97%	-0.0212%

Table 7.6 Comparison of the proposed solution against CIME (Hu and Yang, 2014) with HM-12.1 for LD-Main (part 2)

		IMETRR		BRTRR
Class	Sequence name	CIME	**Proposed**	**Proposed**
B (1920×1080)	BasketballDrive	73.47%	64.72%	12.96%
	Kimono	73.01%	69.02%	48.73%
	ParkScene	64.83%	54.37%	35.30%
	Cactus	62.54%	66.32%	26.11%
	BQTerrace	70.22%	50.71%	2.61%
C (832×480)	BasketballDrill	69.86%	62.19%	41.65%
	BQMall	69.03%	62.57%	49.02%
	PartyScene	65.10%	49.93%	34.15%
	RaceHorsesC	79.78%	65.81%	37.77%
	Average	69.76%	60.63%	32.03%

BD-Rate improvement over CIME for the Kimono sequence. For BasketballDrill the speedup is approximately 5% and the BD-Rate is improved by 1.6%. Overall, the proposed solution reduces by an extra 3% the encoding time without the 1% BD-Rate penalty.

Table 7.7 Savings for the proposed solution when compared with HM-16.8 for Low Delay Main.

		Low Delay			
Class	Sequence name	**ETRR**	IMETRR	BRTRR	BD-Rate(Y)
B (1920×1080)	BasketballDrive	11.73%	57.08%	17.35%	-0.0056%
	Kimono	13.65%	56.58%	45.10%	-0.0896%
	ParkScene	6.15%	41.73%	35.44%	-0.0032%
	Cactus	10.61%	59.21%	23.36%	-0.0104%
	BQTerrace	5.68%	45.49%	9.28%	-0.0522%
C (832×480)	BasketballDrill	8.55%	48.65%	40.08%	0.0462%
	BQMall	9.49%	49.13%	50.93%	0.0949%
	PartyScene	3.97%	32.99%	35.39%	-0.0258%
	RaceHorsesC	11.50%	54.46%	34.18%	0.0051%
	Average	9.04%	49.48%	32.35%	-0.0045%

Table 7.8 Savings for the proposed solution when compared with HM-16.8 for Random Access Main profiles.

		Random Access			
Class	Sequence name	**ETRR**	IMETRR	BRTRR	BD-Rate(Y)
B (1920×1080)	BasketballDrive	8.70%	58.68%	12.03%	0.0177%
	Kimono	7.08%	49.25%	45.55%	-0.0144%
	ParkScene	5.44%	31.27%	41.23%	-0.0122%
	Cactus	8.42%	58.71%	31.83%	-0.0045%
	BQTerrace	4.34%	36.70%	24.83%	0.0317%
C (832×480)	BasketballDrill	8.10%	46.81%	46.02%	0.0048%
	BQMall	8.86%	43.95%	56.73%	-0.0079%
	PartyScene	4.30%	28.69%	39.36%	-0.0103%
	RaceHorsesC	9.03%	51.52%	39.46%	-0.1899%
	Average	7.14%	45.06%	37.45%	-0.0206%

7.5.6 Detailed Time Savings

Tables 7.7 and 7.8 present the ETRR, IMETRR, BRTRR and the BD-Rate of the proposed solution when compared to an unmodified HM 16.8 reference encoder software, for profiles LD and RA main. As explained these results are slightly lower than those presented in tables 7.5 and 7.6, as fixes (Tourapis *et al.*, 2015) were made to the TZ-Search algorithm between versions 12.1 and 16.8 of the HM reference encoder software.

Overall, the ETRR was reduced by 9.04% and 7.14% for LD and RA respectively. For LD, this is achieved by an average combined savings of 49.48% for IMETRR and 32.35% for BRTRR. For RA, this is achieved by an average combined savings of 45.06% for IMETRR and 37.45% for BRTRR. Notice that for both LD and RA, the average impact on BD-Rate is slight improvement. This is caused by the cost-based search ordering used for bi-predictive refinement. When multiple minima exist in the refinement zone, the raster search will take the first one it finds, whereas the proposed approach is rate-biased and will take the one with the smallest motion vector cost. For the current block, this could have a slight positive or negative effect on rate-distortion because the cost function used during ME is only an approximation of the true rate-distortion ratio of the block. However, over the entire frame, this rate bias is slightly more effective as it can improve motion vector prediction.

The concepts presented in this chapter considerably speedup the TZ-Search algorithm and the raster scan used for bi-predictive search.

CONCLUSION

In this thesis, we presented an overview of a modern video encoder followed by a detailed description of temporal prediction. Our literature review covers both motion estimation algorithms and SEAs. We presented our contributions which can be summarized as follows:

A cost-based search ordering pattern

by using this pattern, the ME algorithm will follow the increasing rate rule and avoid weakening the SEA. Our simulation results demonstrate that, on average, for the H.264 reference software encoder, the amount of SAD operations is reduced by 2.86%. For smaller block sizes, this can exceed 10%.

The sorted subset approach

This dynamic search ordering is SEA-optimal and allows for early-termination. Our simulation results demonstrate that, on average, for the HEVC reference software encoder, the amount of SAD operations is reduced by 3.66%. For smaller block sizes, the average rises to 8.06%.

The fast cost-based search ordering algorithm

We proposed a new model to build dynamic search orderings for BMA. From this model, we developed a fast algorithm capable of producing cost-based search orderings that are unaffected by asymmetric distributions of MV costs and off-centered search areas. We showed that this new approach decreases the number of SAD operations by approximately 3%. We also proposed a new early-termination criterion for BMA using a cost-based search ordering. This new optimization only requires performing 36% and 46% of block-matching loop iterations for Random Access and Low Delay respectively. Our experiments show that the proposed solution is more than five times faster than the HEVC HM encoder in full search mode, with the same BD-

PSNR. When compared to an HEVC HM encoder that would implement RCSEA, the proposed solution remains superior, with a 1.4x speedup.

The enhanced rate constraint

By reusing information from the partitioning ME algorithms, this rate constraint significantly outperforms the normal rate constraint. Our experiments show that, on average, when this optimization is combined with the RCSEA in the HEVC HM encoder reference software, the number of SAD operations drops by 94.9%, resulting in a speedup of 6.13x in full search mode.

ML-RCSEA This derivative of the SEA is designed to be used with the suboptimal ME algorithm implemented in the HEVC reference software encoder. When compared to the reference encoder, our experiments show that the proposed solution reduces the motion estimation time by approximately 45% contributing to an average encoding time reduction of about 7% without increasing the BD-Rate.

Looking back, we assert that our objectives have been achieved. We have developed improvements to SEA for both optimal and suboptimal ME algorithm. The insights obtained in this research effort allowed us to propose concepts like the increasing rate rule, SEA-optimal ME, and the enhanced rate constraint. As such, our work has generated a better understanding of the solution space for ME algorithms.

In light of the work of Seidel *et al.* (2016), that is based on the contributions of this research effort, we are confident that, although it is now the end of this endeavor, our contributions have the potential to drive subsequent research initiatives and to be implemented in modern video encoders.

LIST OF REFERENCES

Bjøntegaard, Gisle. 2001. *Calculation of average PSNR differences between RD-curves*. Technical Report VCEG-M33. 13th Meeting: Austin, Texas, USA : Video Coding Experts Group (VCEG) of ITU-T, 1–4 p.

Bossen, Frank. 2013. *Common test conditions and software reference configurations*. Technical Report JCTVC-L1100. 12th Meeting: Geneva : Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, 1–4 p.

Cai, Jing and W. David Pan. 2010. "Fast Exhaustive-Search Motion Estimation Based on Accelerated Multilevel Successive Elimination Algorithm with Multiple Passes". In *Proceedings of the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. p. 1190–1193.

Chen, Zhibo, Jianfeng Xu, Yun He, and Junli Zheng. 2006. "Fast integer-pel and fractional-pel motion estimation for H.264/AVC". *Journal of Visual Communication and Image Representation*, vol. 17, n° 2, p. 264–290.

Cisco. 2016. "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020". http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. Accessed: 2016-09-08.

Coban, Muhammed Z. and Russell M. Mersereau. 1998. "A Fast Exhaustive Search Algorithm for Rate-Constrained Motion Estimation". *IEEE Transactions on Image Processing*, vol. 7, n° 5, p. 769–773.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 2001. *Introduction to algorithms*. ed. 2nd. Cambridge, Massachusetts : McGraw-Hill, 1180 p.

Gao, X. Q., C. J. Duanmu, and C. R. Zou. 2000. "A Multilevel Successive Elimination Algorithm for Block Matching Motion Estimation". *IEEE Transactions on Image Processing*, vol. 9, n° 3, p. 501–504.

Girod, Bernd. 1993. Motion compensation: Visual aspects, accuracy, and fundamental limits. Sezan, M. I. and Reginald L. Lagendijk, editors, *Motion Analysis and Image Sequence Processing*, p. 125–152. Springer US. ISBN 978-1-4613-6422-1. doi: 10.1007/978-1-4615-3236-1.

He, Zhongli and Ming L. Liou. 1997. "A High Performance Fast Search Algorithm for Block Matching Motion Estimation". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, n° 5, p. 826–828.

Hearn, Donald and M. Pauline Baker, 2004. *Computer graphics with OpenGL*. Pearson Custom Computer Science Series. ed. 3rd. Upper Saddle River, NJ, USA : Pearson Prentice Hall.

Hosur, Prabhudev I. and Kai-Kuang Ma. 1999. "Motion Vector Field Adaptive Fast Motion Estimation". In *Proceedings of the 1999 International Conference on Information, Communications and Signal Processing*.

Hu, Nan and En-Hui Yang. 2014. "Fast Motion Estimation Based on Confidence Interval". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, n° 8, p. 1310–1322.

ISO/IEC JTC 1/SC 29/WG11. 2015. *ITU-T recommendation H.265: High efficiency video coding*. ITU-T H.265 (V3) International Telecommunications Union. 317 p.

ITU-T SG16 Q.6 and ISO/IEC JTC 1/SC 29/WG11. 2003. *ITU-T recommendation H.264: Advanced video coding for generic audiovisual services*. Technical Report International Telecommunications Union.

Jain, Jaswant R. and Anil K. Jain. 1981. "Displacement Measurement and Its Application in Interframe Image Coding". *IEEE Transactions on Communications*, vol. 29, n° 12, p. 1799–1808.

Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. 2013. *H.264/AVC JM Reference Software* (Version 18.5) [Computer Software]. Retrieved from http://iphome.hhi.de/suehring/tml/.

Li, Renxiang, Bing Zeng, and Ming L. Liou. 1994. "A New Three-Step Search Algorithm for Block Motion Estimation". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, n° 4, p. 438–442.

Li, W. and E. Salari. 1995. "Successive Elimination Algorithm for Motion Estimation". *IEEE Transactions on Image Processing*, vol. 4, n° 1, p. 105–107.

Lim, Keng-Pang, Gary J. Sullivan, and Thomas Wiegand. 2006. *Text description of joint model reference encoding methods and decoding concealment methods*. Technical Report JVT-R095. Thailand : Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG.

McCann, Ken, Chris Rosewarne, Benjamin Bross, Matteo Naccari, Karl Sharman, and Gary J. Sullivan. 2014. *High efficiency video coding (HEVC) test model 16 (HM 16) improved encoder description*. Technical Report JCTVC-S1002. 19th Meeting Strasbourg, France : Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11.

Musmann, Hans Georg, Peter Pirsch, and Hans-Joachim Grallert. 1985. "Advances in Picture Coding". *Proceedings of the IEEE*, vol. 73, n° 4, p. 523–548.

Richardson, Iain E., 2010. *The H.264 advanced video compression standard*. ed. 2nd. West Sussex, United Kingdom : John Wiley & Sons Ltd, 316 p.

Seidel, Ismael, Luiz Henrique Cancellier, José Luís Güntzel, and Luciano Agostini. 2016. "Rate-Constrained Successive Elimination of Hadamard-Based SATDs". In *Proceedings of the 2016 IEEE International Conference on Image Processing*. p. 2395-2399.

Sullivan, Gary J. and Thomas Wiegand. 1998. "Rate-Distortion Optimization for Video Compression". *IEEE Signal Processing Magazine*, vol. 15, n° 6, p. 74–90.

Sullivan, Gary J., Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. "Overview of the High Efficiency Video Coding (HEVC) Standard". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, n° 12, p. 1649–1668.

Takagi, Koichi, Yasuhiro Takishima, and Yasuyuki Nakajima. 2003. "A Study on Rate Distortion Optimization Scheme for JVT Coder". p. 914–923.

Tham, Jo Yew, Surendra Ranganath, Maitreya Ranganath, and Ashraf Ali Kassim. 1998. "A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, n° 4, p. 369–377.

Tourapis, Alexis M. 2002. "Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation". *Visual Communications and Image Processing*, vol. 4671, n° May 2002, p. 1069–1079.

Tourapis, Alexis M., Oscar C. Au, and Ming L. Liou. 1999. "A High Performance Algorithm for Fast Block Based Motion Estimation". In *Proceedings of the 1999 Picture Coding Symposium*. p. 121–124.

Tourapis, Alexis M., Oscar C. Au, and Ming L. Liou. 2002. "Highly Efficient Predictive Zonal Algorithms for Fast Block-Matching Motion Estimation". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, n° 10, p. 934–947.

Tourapis, Alexis M., Yeping Su, David Singer, Joel Sole, Dmytro Rusanovskyy, S. Lee, D. Bugdayci, A. Ramasubramonian, M. Karczewicz, Chad Fogg, Alberto Duenas, and Frank Bossen. 2015. *HM reference software bug fixes and enhancements to address the HDR/WCG CfE*. Technical Report JCTVC-U0040. 21st Meeting: Warsaw, PL : Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, 1–4 p.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2014. "Rate Distortion-Based Motion Estimation Search Ordering for Rate-Constrained Successive Elimination Algorithms". In *Proceedings of the 2014 IEEE International Conference on Image Processing*. (Paris, France 2014), p. 3175–3179.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2015a. "An Adaptive Search Ordering For Rate-Constrained Successive Elimination Algorithms". In *Proceedings of the 2015 IEEE International Conference on Image Processing*. (Québec, Canada 2015), p. 207–211.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2015b. "Method and System for Rate-Constrained Search Ordering". US Patent App. 14/609,324.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2016a. "Cost-Based Search Ordering for Rate-Constrained Motion Estimation Applied to HEVC". *SUBMITTED TO IEEE Transactions on Circuits and Systems for Video Technology*.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2016b. "Sub-Partition Reuse for Fast Optimal Motion Estimation in HEVC Successive Elimination Algorithms". In *Proceedings of the 2016 IEEE International Conference on Image Processing*. p. 2003-2007.

Trudeau, Luc, Stéphane Coulombe, and Christian Desrosiers. 2016c. "Methods and Systems for Determining Motion Vectors in a Motion Estimation Process of a Video Encoder". US Patent App. 15/009,938.

Turaev, Vladimir G., 2010. *Quantum Invariants of Knots and 3-manifolds*. De Gruyter studies in mathematics. New York : De Gruyter, 592 p.

Wang, Yao, Jörn Ostermann, and Ya-Qin Zhang, 2001. *Video processing and communications*. ed. 1st. Upper Saddle River, NJ, USA : Prentice Hall, 595 p.

Wiegand, Thomas, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. 2003. "Overview of the H.264/AVC video coding standard". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, n° 7, p. 560–576.

Zhu, Ce, Wei-Song Qi, and Wee Ser. 2005. "Predictive Fine Granularity Successive Elimination for Fast Optimal Block-Matching Motion Estimation". *IEEE Transactions on Image Processing*, vol. 14, n° 2, p. 213–221.

Zhu, Shan and Kai-Kuang Ma. 2000. "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation". *IEEE Transactions on Image Processing*, vol. 9, n° 2, p. 287–290.