Rate-Distortion-Complexity Optimization of Video Encoders with Applications to Sign Language Video Compression

Rahul Vanam

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2010

Program Authorized to Offer Degree: Electrical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Rahul Vanam

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Co-Chairs of the Supervisory Committee:

_____

Eve A. Riskin

_____

Richard E. Ladner

Reading Committee:

_____

Eve A. Riskin

_____

Richard E. Ladner

_____

Maya R. Gupta

Date:_____

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Rate-Distortion-Complexity Optimization of Video Encoders with Applications to Sign Language Video Compression

Rahul Vanam

Co-Chairs of the Supervisory Committee:
Professor Eve A. Riskin
Electrical Engineering

Professor Richard E. Ladner
Computer Science and Engineering

Applications for video compression have been growing over the years due to the availability of higher network bandwidth, lower cost of memory, and faster processor speed. Some new applications include real-time videoconferencing and video streaming. Most current cell phones come equipped with a video camera, and have the ability to capture a video and playback a recorded video. An emerging area of video compression is mobile videoconferencing, which is enabling the Deaf to communicate in American Sign Language (ASL) using video cell phones. Video encoders developed for PCs cannot be readily used for mobile phones, due to mobile phones' low processor speeds. In this thesis, we present algorithms for improving the speed of the H.264 video encoder on both PC and cell phone platforms by selecting encoder parameters that jointly trade off encoding speed and video quality at different bitrates.

We also apply our algorithms to a region-of-interest based video encoder specific to ASL. This encoder jointly optimizes for ASL intelligibility and bitrate. The parameters chosen by our algorithms are demonstrated to significantly improve the encoding speed at a given ASL intelligibility for different bitrates, on both PC and cell phone platforms.

ASL videoconferencing on a cell phone drastically reduces its battery life. To extend the cell phone battery life, we detect the signer's activity on the cell phone, and use it to

control the backlight of the cell phone and the encoding frame rate. This approach improves the battery life by up to 54 minutes. Finally, we present a rate control method that allows the encoder bitrate to adapt to time-varying network bandwidth. We show that when this encoder operates with suitably chosen parameters, it yields both higher encoding speed and ASL intelligibility on a cell phone over a constant bitrate encoder.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ABC: Adaptive Backlight Control

ABR: Adaptive Bitrate

ASL: American Sign Language

CABAC: Context Adaptive Binary Arithmetic Coding

CAVLC: Context Adaptive Variable Length Coding

CBR: Constant Bitrate

CLSA: Controlled Local Search Algorithm

DCT: Discrete Cosine Transform

DIA: Diamond motion search

DPSPA: Dominant Parameter Setting Pruning Algorithm

ESA: Exhaustive Search Algorithm used in motion estimation

GBFOS: Generalized Breiman, Friedman, Olshen, and Stone

GOP: Group of Pictures

H.264/AVC: Video compression standard

HEX: Hexagon motion search

HR: Hyperarea Ratio

MOPSO: Multiobjective Particle Swarm Optimizer

MSE: Mean Squared Error

PSNR: Peak Signal-to-Noise Ratio

QCIF: Quarter Common Intermediate Format

ROI: Region-of-Interest

ROPA: Recursive Optimal Pruning Algorithm

SAD: Sum of Absolute Difference

SSD: Sum of Squared Difference

VFR: Variable Frame Rate power saving scheme

VQ: Vector Quantization

VSR: Variable Spatial Resolution power saving scheme

ACKNOWLEDGMENTS

I am enormously grateful to my advisors Eve Riskin and Richard Ladner for their guidance and for giving me an opportunity to work on interesting problems. I am very grateful to Eve for her mentorship, kindness, and for giving me helpful advice on my Ph.D. studies and job search. Richard has been very approachable, and his suggestions and feedback have helped my research greatly. I am grateful to Sheila Hemami for giving me important comments on my work. I am grateful to my other committee members, Maya Gupta, Michael Cohen, Agnieszka Miguel, and Tatiana Toro for their constructive comments.

My work would not have been possible without the collaborations from my colleagues Jaehong Chon, Frank Ciaramello, Neva Cherniavsky, Anna Cavendar, Dane Barney, and Loren Merritt. I am indebted to Jaehong for helping me understand the MobileASL code, and Frank for our joint work on optimizing the Cornell ASL video encoder. I would like to thank Jessica Tran, Tressa Johnson, and Joy Kim for helpful discussions.

I would like to thank my colleagues in the EE 423 lab for their friendship: Pascal Clark, Cameron Colpitts, Brian King, Xing Li, Danny Luong, Patrick McVittie, Nicole Nichols, Scott Phillips, Steve Schimmel, and Elliot Saba. I enjoyed their company during weekly lab lunches and potlucks. I would like to thank my well-wishers and friends from the Vedanta Society for their support and encouragement, and making me feel at home in Seattle: Swamis Bhaskarananda, Avikarananda, and Atmajayananda, Dr. Allen Freedman, Devra Freedman, David Oliver, Charles Wirth, and Ashwini Sharma.

Finally, I would like to thank my parents. Without their love, support, and encouragement, I would not have been able to complete my graduate studies. I am very grateful to my mother for motivating me to pursue graduate studies early on in my life.

This work was funded by the National Science Foundation under grants CCF-0514353 and IIS-0811884.

DEDICATION

To my parents, Sharada and Chandrasekhar

Chapter 1

INTRODUCTION TO VIDEO COMPRESSION

Video encoders are used to compress digital video to allow for easy storage and transmission. Many common applications such as digital video broadcast, Youtube, internet video chat, etc. use video compression. Current digital cameras, camcorders and video cell phones are equipped with in-built video encoders. The video encoders are now available not only as software plugins, but also as customized hardware and firmware for numerous electronic devices. In this chapter, we give an overview of H.264/AVC [5], the current most advanced video compression standard. We also describe the x264 encoder [6], and our MobileASL videoconferencing application for cell phones [2]. Finally, we present the outline of this thesis.

A video consists of series of images called *frames*. The playback speed of frames is specified by the *frame rate* and is measured in frames per second (fps). The number of pixels in each frame is specified by the height and width of a frame, and is referred to as resolution. In a color video, each pixel is composed of three components: one luminance or luma (Y) and two chrominance or chroma components (Cb, Cr).

Often compression is performed on the raw video by sub-sampling the two chrominance components as this does not result in much visual difference. The 4:2:0 is one such format, where the chrominance components are sub-sampled by two in both horizontal and vertical direction. A raw 4:2:0 video with 30 fps and 176×144 resolution requires about 9 Mb/s. This bitrate is quite high considering that current wireless transmission channels operate at much lower bandwidth. A video can be compressed to lower its bitrate while preserving content and quality.

Video compression standards define a bitstream syntax for the decoder. A bitstream conforming to a standard's syntax will give the same decoded output when using any standard

compliant decoder. The standard does not specify the encoder, therefore giving freedom for development of algorithms that can improve the speed, quality and compression rate of the encoder. Examples of international video compression standards are H.120, H.261, MPEG-1, MPEG-2, H.263, VC-1 and H.264.

Certain image compression techniques are extended to video compression. For example, when a frame has no similarity with its previous frames, like during a scene-change or the first frame of a video, image compression schemes are used. Many image compression schemes use the Discrete Cosine Transform (DCT) to frequency transform blocks of pixels, followed by variable length coding scheme. This coding method is referred to as *intra-frame coding*.

In video, there is often temporal redundancy, as a frame can have similar content with frames occurring closer in time. Video encoders exploit this redundancy by finding for each block of pixels in the current frame, a best match of block of pixels in previously encoded frames to represent it. This is called *inter-frame coding*. Inter-frame coding involves searching over previously encoded frames in a process called *motion estimation*. The best matched block can be efficiently identified by a *motion vector*. It gives both the spatial and temporal displacement of the best matched block with respect to the current block. The encoded frames being searched are called *reference frames*. The encoder uses the motion vector to predict the current block being encoded, and this process is called *motion compensation*.

1.1 Overview of the H.264/AVC encoder

Recent video encoders use a hybrid of motion compensation and image-coding methods [7,8]. In this section, we provide an overview of the H.264 or MPEG-4 advanced video coding (AVC) encoder. A more detailed discussion of this standard can be found in [1, 5, 8–10]. The H.264 was standardized by both the ITU-T and ISO/IEC in 2003 [5] and it yields an average bitrate savings of about 50% over MPEG-2 for the same video quality [1]. The different components of a H.264 encoder are shown in Figure 1.1.

In H.264, encoding is done per block of 16×16 pixels called a *macroblock*. A group of macroblocks may be grouped into a *slice*, which can be independently decoded at the

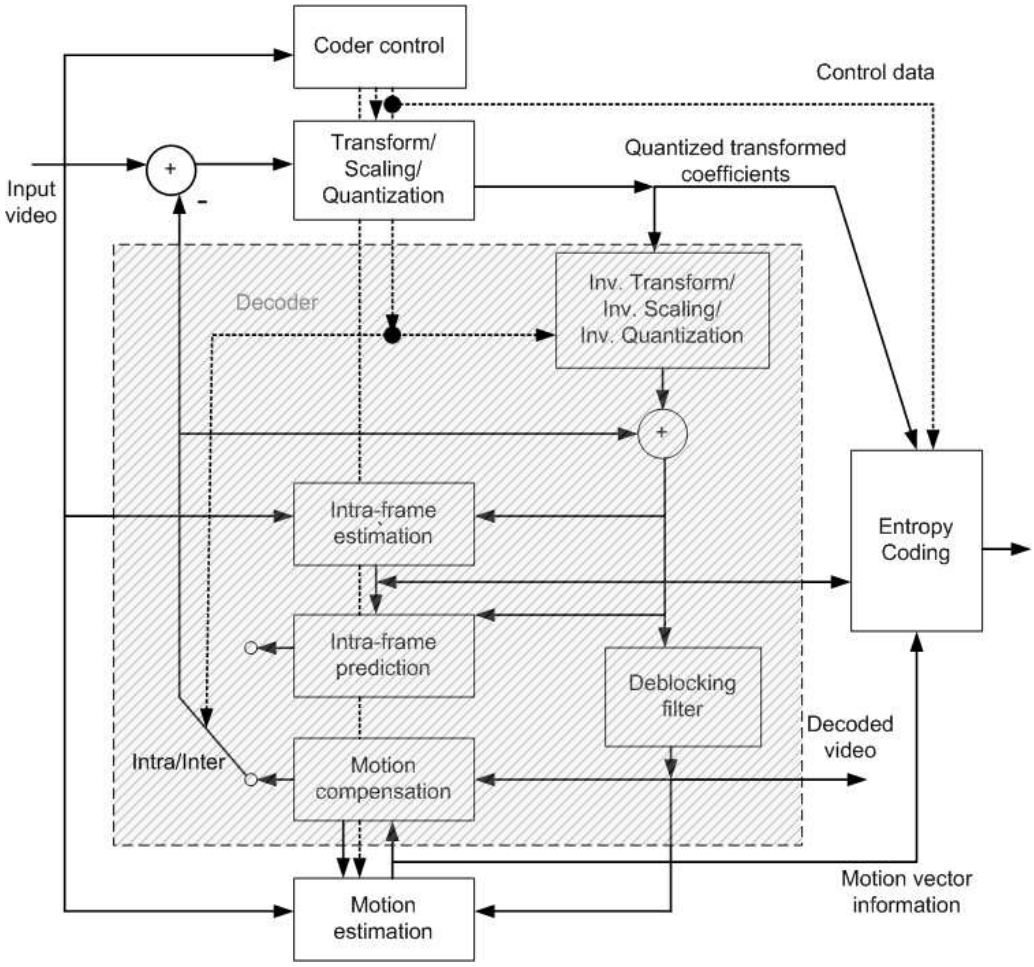
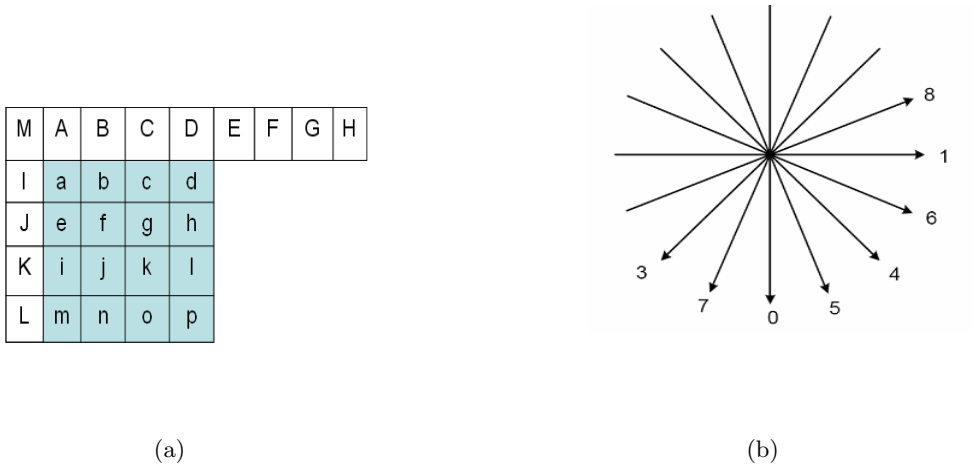


Figure 1.1: H.264/AVC encoder [1]. The shaded region includes components that are common for both the H.264 encoder and decoder.

decoder. When slices are not specified, a frame corresponds to a slice.

Intra-coding of luma samples are done on either a block of 4×4 pixels or 16×16 pixels. Figure 1.2(a) shows 4×4 luma samples a–p that are to be intra coded using samples A–



(a) (b)

Figure 1.2: (a) Intra luma 4×4 prediction on samples a-p using samples A-M that are already encoded and (b) the eight directional modes of Intra luma 4×4 [1].

M that have already been encoded. The intra luma 4×4 has nine different modes, which includes one DC mode and eight modes corresponding to a specific direction shown in Figure 1.2(b). Intra 4×4 modes are useful for coding details more efficiently.

Intra-coding for 16×16 blocks is done using one of the four modes: vertical, horizontal, DC and plane mode. The first three modes are similar to the corresponding modes in intra 4×4. The plane mode uses neighboring top and left samples to predict current samples [10]. Intra prediction for chroma samples is the same as intra luma 16×16, except it is done for 8×8 blocks.

In inter coding, a macroblock can be partitioned into smaller subblocks, allowing motion estimation on smaller blocks, resulting in better prediction. A 16×16 block can be partitioned into 16×8, 8×16, and 8×8 subblocks. An 8×8 block can be further partitioned into 8×4, 4×8, and 4×4 blocks. Figure 1.3 shows different partition sizes allowed in H.264. The H.264 standard allows up to 16 previously encoded frames as reference frames (known as P-frames). In a B-frame, a block can be predicted from both a temporally occurring previous frame and a future frame that have been encoded. In an example illustrated in Figure 1.4, the B frame is predicted by previous I and P-frames, and a future P-frame.

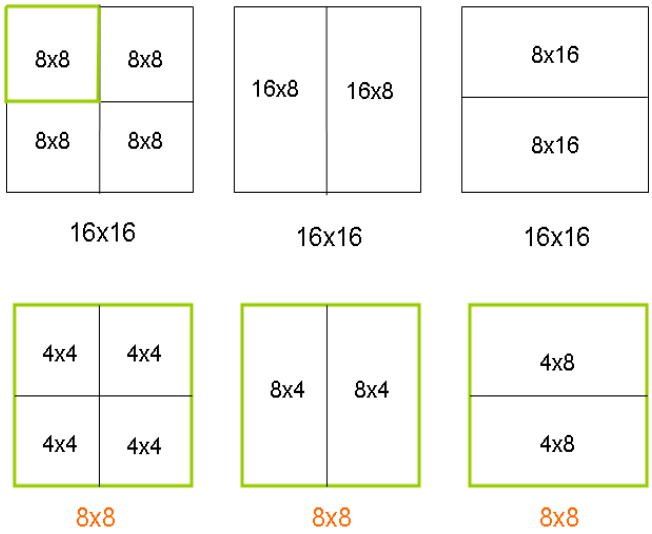


Figure 1.3: Macroblock partitions in H.264/AVC.

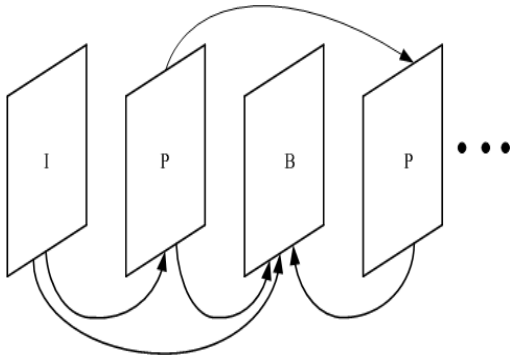


Figure 1.4: Example of intra (I), predictive (P) and bi-predictive (B) frames.

To improve prediction, H.264 allows motion estimation at half and quarter pixel positions. This is useful especially when objects in a frame have undergone fractional pixel displacement. Sub-pixel motion estimation first finds the integer pixel position that results

in the best match. These integer pixels are filtered using a 6-tap finite impulse response filter, both horizontally and vertically to obtain samples in half-pixel position. The samples at quarter pixel position are then obtained by averaging over samples at integer and half pixel positions [1]. Motion estimation is time-consuming because the search is performed over different fractional pixel positions, for different partition sizes, and over different reference frames.

Motion estimation finds the best motion vectors for each partition size or *mode*. The different modes are compared and the mode that results in the least distortion for a given rate is chosen for further encoding. This process is referred to as *mode decision*.

The difference or residual between the original block and intra or inter-frame prediction is frequency transformed by a DCT. The DCT can efficiently compact the residue into fewer non-zero coefficients. In H.264, an integer transform similar to DCT is used [11]. This has advantage of lowering the decoder complexity and avoiding the mismatch in the inverse transformed output due to floating-point arithmetic. The H.264 allows only a 4×4 transform while the fidelity range extension to H.264 allows both 4×4 and 8×8 transform [12].

The quantization is controlled by a quantization parameter (QP). The quantization step size doubles whenever the QP increases its value by six. An increase in the quantization step size increases the number of zeros of quantized transformed coefficients, thereby decreasing the bitrate.

The quantized transformed coefficients are scanned using a zigzag pattern that converts a two dimension matrix into one dimensional vector. This vector of quantized coefficients typically has few non-zero coefficients. They can be entropy coded either using context adaptive variable length coding (CAVLC) or context adaptive binary arithmetic coding (CABAC). CABAC provides higher compression than CAVLC, but at the cost of increased complexity.

The encoder includes components that are common with the decoder as shown by the shaded region in Figure 1.1. It ensures that prediction is identical at both the encoder and decoder. At the encoder, the quantized and transformed coefficients are inverse quantized and scaled to recover the residue. The residue is then added to the prediction, which is filtered using a deblocking filter. The deblocking filter is part of the H.264 standard. The

deblocking filter removes the block-edges and improves the quality of the decoded frames. The decoded frames are stored for prediction of subsequent encoded frames [8].

1.2 Rate-distortion Optimization

Each mode in the encoding process provides a bitrate and distortion. The goal of an encoder is to choose a mode to minimize the distortion D such that the bitrate R is within a given rate constraint R_t. The optimization problem is given by the following equation [7]

$$\min D, \text{ subject to } R < R_t. \tag{1.1}$$

This can be formulated as a joint rate-distortion (R-D) optimization given as

$$\min J = D + \lambda R, \tag{1.2}$$

where λ is the Lagrange multiplier. A solution to Equation 1.2 for a given λ corresponds to an optimal solution to Equation 1.1 for a given R_t [7].

To find the best motion vector in motion estimation, the R-D cost given in Equation 1.2 is used. Computing the actual distortion between the reconstructed and original block is time consuming due to the encoding and decoding process. Therefore, the distortion is computed as the sum of the absolute differences (SAD) between the original block and the block in the reference frame. For motion estimation, Equation 1.2 is reformulated as

$$\min J_{Motion} = SAD + \lambda_{Motion} R_{Motion}, \tag{1.3}$$

where R_{Motion} corresponds to the bits required for the motion vector and the reference frame index.

In the mode decision, the final mode is based upon the R-D cost similar to Equation 1.2. The distortion is the sum of the squared difference (SSD) between the original block and the reconstructed block. The R-D optimization for the mode decision is formulated as

$$\min J_{Mode} = SSD + \lambda_{Mode} R_{Mode}, \tag{1.4}$$

where R_{Mode} corresponds to the bits associated with choosing a given $Mode$. We refer to J_{Mode} as the actual R-D cost in future chapters.

1.3 x264 Encoder

In this section, we describe input parameters of the x264 encoder, an open source implementation of the H.264 encoder [6]. The x264 encoder has been compared with different commercial H.264 encoders and it was found to provide the best quality in terms of PSNR, DCT based video quality metric, and structural similarity metric [13]. In [14], the x264 encoder was compared with the JM reference encoder (ver 10.2) [15] and x264 was shown to be 50 times faster while providing bitrates within 5% for the same PSNR. We use the x264 encoder to test our algorithms.

The x264 encoder has several input parameters that determine the bitrate and distortion of the encoded video, and the speed of the encoder. We refer to a vector of encoder parameter choices as a *parameter setting*. An example of a parameter setting is the use of one reference frame; a partition size of 4×4 and 8×8 for predictive blocks; quarter pixel motion estimation; and CABAC for entropy coding. We describe some input parameters of the x264 encoder below.

1.3.1 Number of Reference Frames (`ref`)

This refers to the number of reference frames for forward prediction. It ranges from 1 – 16 frames. A higher number of reference frames can result in a better match during motion estimation, thereby resulting in a smaller residual. However, a higher number of reference frames requires a higher encode time.

1.3.2 Partition Size (`part`)

This specifies the partition size allowed for intra (I), predictive (P) and bi-predictive (B) macroblocks. For intra macroblocks, I4×4, I8×8 and I16×16 modes are allowed. I16×16 is included by default even when not specified to the x264 encoder. For predictive macroblocks, we have the choice of P4×4 and P8×8. In x264, use of P4×4 implicitly considers P4×8

and P8 × 4, and use of P8 × 8 implicitly considers P8 × 16, P16 × 8 and P16 × 16. For B macroblock type, we consider B8 × 8 that implicitly considers B8 × 16 and B16 × 8. One or more of the following partition size options can be specified to the encoder: P8 × 8, P4 × 4, I8 × 8, I4 × 4 and B8 × 8. Including more partition sizes improves the rate-distortion performance of the encoder. However, it increases the encode time.

1.3.3 Sub-pixel Motion Estimation (`subme`)

The x264 encoder selects candidate block types for the mode decision based upon the rate-distortion cost computed during motion estimation. The `subme` option specifies the number of iterations of half pixel and quarter pixel diamond search that has to be performed on all candidate block types. The resulting block type having the least cost is then coded. The `subme` option also specifies the number of iterations of half and quarter-pixel diamond refinement search to be performed on the final partition size. The `subme` has seven options, each of increasing complexity and improved coding efficiency which are listed below.

1. `subme` = 1: An integer pixel motion search is performed on all candidate block types and the resulting best motion vector is refined using a half-pixel motion search.

2. `subme` = 2: It is the same as `subme` = 1, but it applies both half and quarter pixel motion search for refinement.

3. `subme` = 3: It applies half pixel search on all candidate block types and then applies quarter pixel search for refinement.

4. `subme` = 4: It is the same as `subme` = 3, but uses two iterations of quarter pixel search for final refinement.

5. `subme` = 5: It is the same as `subme` = 4, except that it uses one iteration of half and quarter pixel motion search for all candidate block types.

6. `subme` = 6: It uses two iterations of half and quarter pixel search for all candidate block types and enables rate-distortion optimization for block types in I and P frames.

7. `subme` = 7: In addition to the process in `subme` = 6, it uses rate distortion optimization for refining the final inter or intra mode selected. In inter mode, it does a half pixel and quarter pixel refinement using the actual R-D cost. While in intra mode, it computes the R-D cost for different intra modes to determine the best mode. This option is time consuming, but can provide the best rate-distortion performance.

1.3.4 Quantization Method (`trellis`)

There are three different quantization methods in x264 that are based upon rate-complexity tradeoffs. The first method is based on the uniform deadzone quantization (called Trellis-0) and the other two methods are based on the trellis approach (called Trellis-1 and Trellis-2). There is an increase in both complexity and bitrate savings, by using higher order implementations of trellis.

Trellis-1 and Trellis-2 differ in the quantization approach used during mode decision, but they both use CABAC for entropy coding. There are two steps in the mode decision:

1. Computation and comparison of the R-D cost of the different candidate block types. The R-D cost is obtained from motion estimation using Equation 1.3, where the sum of the absolute Hadamard transformed difference is used instead of SAD.

2. The block having the least R-D cost from the previous step is chosen as the final block type to be encoded. The final block is further refined by computing the actual R-D cost using Equation 1.4.

Trellis-1 uses uniform quantization in step (1) and trellis in step (2) of mode decision. Trellis-2 uses the trellis in both steps of the mode decision and therefore has higher complexity.

We describe the trellis using a directed graph shown in Fig 1.5, where each column represents 16 DCT coefficients of the 4x4 luma block denoted by $C0, C1, \ldots, C15$ [16]. Each DCT coefficient can be coded with one of 8 possible CABAC contexts as shown by the rows in Figure 1.5. The number of coefficients greater than one (NumLg1) and equal to one (NumEq1) determine the CABAC context. The nodes of the graph are labeled as

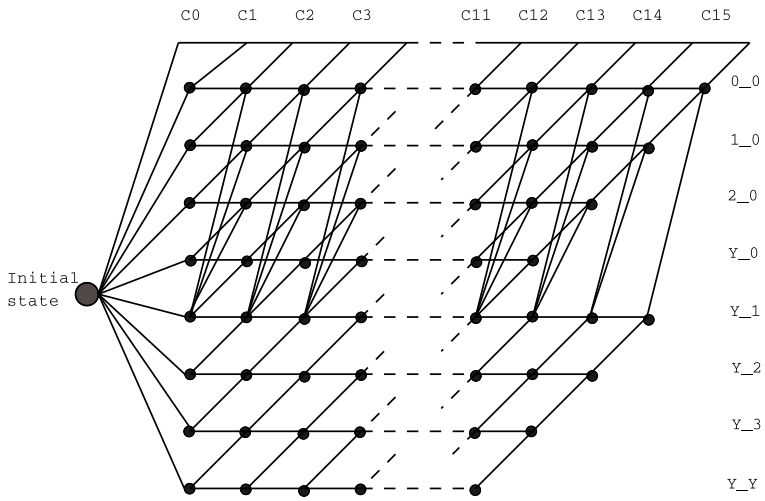


Figure 1.5: Trellis quantization in x264.

NumEq1_NumLg1. Each node belonging to a row in Fig 1.5 has the same NumEq1_NumLg1 and its values are shown by the entries in the right most column of Fig 1.5. The transition between these nodes indicates a transition between states.

When NumLg1 > 0, the CABAC context does not consider NumEq1, and we represent the state as `Y_i` in Fig 1.5, where $0 \leq i \leq 3$. When NumLg1 > 3, we represent the state as `Y_Y`. The trellis is implemented as a dynamic program, where any two states of the same (DCT index, CABAC context) pair are considered the same, and only the one with the least R-D cost is retained. The algorithm is described below:

1. The different states are compared based on the R-D cost and the state with the least cost is chosen.

2. Transition between states: A new coefficient is encoded onto one of the partial encodings. First, the rate corresponding to CABAC and the distortion of the chosen quantized level for the new coefficient are calculated. The CABAC context is then updated depending on the level coded. The R-D cost corresponding to the new coefficient is added to the total R-D cost of the current state. If a state is reached that has already been reached through a different path in the trellis, then their R-D costs

are compared and the path with the least cost is retained.

3. Dijkstra's algorithm [17] is used to find the path with the least R-D cost.

4. Optimizations: Some coefficient values are unlikely to occur. Therefore, they are omitted from the search without measurable quality loss. The coefficient is first quantized by rounding it to the nearest integer. If the *level* or quantized coefficient equals zero, the coefficient is chosen to be zero, and its column can be omitted from the directed graph given in Fig 1.5. Otherwise, step (2) is repeated for both *level* and $(level - 1)$. For example, if the unquantized coefficient is 1.8, it is quantized to a $level = 2$. Then step (2) is applied for both $level = 2$ and $(level - 1) = 1$, and the horizontal transition corresponding to $level = 0$ is omitted.

1.3.5 DCT Size (`dct`)

The x264 encoder either uses 4×4 DCT or both 4×4 and 8×8 DCT.

1.3.6 Mixed Reference (`mixed-refs`)

This option allows the use of independent reference frames for each 8×8 partition of a macroblock. When a higher number of reference frames is used, mixed-refs can improve coding efficiency but increases complexity. When `mixed-refs` is not used, reference frames are chosen per macroblock.

1.3.7 Motion Search Method (`ms`)

In motion estimation, search is performed on reference frames to find block of pixels that best matches the current block of pixels. This process can be computationally expensive. The x264 encoder provides four options for integer pixel motion search and they are listed in order of increasing complexity and quality: diamond search of radius one (DIA) [18], hexagon search of radius two (HEX) [19], uneven multihexagon (UMH) [20] and exhaustive search (ESA) [21].

The x264 encoder defines default options for its encoder parameters. They are used in the absence of user specified options. Specifically, by default the following options are used: one reference frame; $P8 \times 8$, B8 $\times$ 8, $I8 \times 8$, $I4 \times 4$ partition size; `subme` = 5; and `trellis` = Trellis-0. In this thesis, this parameter setting is referred to as the *default parameter setting*.

1.4 The MobileASL System

Web based videoconferencing has become a popular and cost effective means to communicate. Most of these videoconferencing applications are software-based, do not require expensive specialized hardware, and operate on most computers having a webcam. These applications use real-time video encoders to compress and packetize video for transmission.

The MobileASL project [22] uses video compression to enable Deaf users to communicate in American Sign Language (ASL) using video cell phones over low bandwidth network. Video telephone and internet-based video chat services are already popular among Deaf people. The MobileASL project extends videoconferencing to mobile cell phones, giving Deaf people the mobility and access to communicate using American Sign Language. Mobile videoconferencing is already available in Sweden, Japan, and other countries, but video quality is poor and jerky, and often has significant delay.

MobileASL uses the x264 video encoder described in Section 1.3 for video encoding and ffmpeg [23] for decoding [2]. The MobileASL team has ported the codec (encoder and decoder) to an HTC TyTN II cell phone that has the Windows Mobile 6.1 operating system, and a front facing camera. Like most videoconferencing applications, MobileASL allows the user to see the other person in addition to himself/herself on the screen, and this is illustrated in Figure 1.6. It also allows users to text and send SMS. The application can operate both on a Wi-Fi and 3G data network and has cross-network communication capability. That is, a video call can be established between two cell phones, with one cell phone on a Wi-Fi network and the other on a 3G network. It also provides an online contact list to the user, similar to a phone book containing user names and phone numbers. The contact list is automatically and periodically updated on a server to map a user to a network IP number.

Figure 1.6: MobileASL running on a HTC TyTN II cell phone [2]. Neva Cherniavsky (on the right) and I are shown talking to each other. The red circle and the arrow points to the front facing camera.

1.5 Contributions

The biggest challenge in MobileASL has been to run the encoder at a high enough frame rate on the cell phone. Video compression is very computationally intensive, and most current cell phone processors are not fast enough to run the encoder in real-time. The x264 encoder can be sped up on a cell phone by choosing parameter settings that take fewer computations at a cost of reduced quality; appropriate selection of parameter settings can ensure both good quality and fewer computations. We propose fast offline algorithms for finding such parameter settings using training videos, and storing the parameter settings and their associated encoding time in a lookup table. When a test video has to be encoded, a parameter setting is chosen from the lookup table based on the current available computational resources on the device. This approach improves the encoder speed on both PC and cell phone platforms over the x264 default parameter setting, with little difference in video quality. Chapter 2 describes our fast parameter settings selection algorithms.

In Chapter 3, we present the metrics used to evaluate the performance of our algorithms. We apply our algorithms to the x264 encoder and H.263+ encoder [24], and compare their performance to other parameter setting selection methods and the x264 default parameter setting. In Chapter 4, we run our algorithms on cell phones for ASL videos and evaluate their performance. We analyze the cross-platform performance of our algorithms, and the performance of our algorithms when there is a mismatch between ASL signers in training and test videos.

In Chapter 5, we apply our algorithms to an ROI-based video encoder specific to ASL [25], and compare the intelligibility, encoding speed, and bitrate performance to the x264 default parameter setting on both PC and cell phone platforms.

Running MobileASL on the cell phone quickly drains battery power thereby shortening the cell phone life. In Chapter 6, we present methods that save battery power usage when running MobileASL on the cell phone. These methods result in longer battery life when compared to earlier power saving methods.

Bandwidth in wireless networks such as Wi-Fi and 3G is time varying. In Chapter 7, we present a bitrate adaptive x264 encoder that adapts the bitrate per frame to the available bandwidth. We show that by operating this encoder with parameter settings selected by our algorithms, we get improvement in encoding speed and ASL intelligibility over the constant bitrate x264 encoder. Finally in Chapter 8, we conclude and provide future directions for research.

Chapter 2

DISTORTION-COMPLEXITY OPTIMIZATION OF THE H.264 ENCODER

2.1 Introduction

During the standardization of H.264 the main focus was to improve its rate-distortion (R-D) performance over earlier video coding standards. As a result, the H.264 standard supports many new features that improve its R-D performance but at the cost of increased computational complexity. Some of the complex features supported by the H.264 standard include multiple reference frames, quarter pixel motion estimation and compensation, CABAC, etc. These features do not allow the H.264 encoder to run in real-time on low speed mobile devices.

In this chapter, we discuss prior work in complexity optimization of video encoders including the H.264 encoder. We describe six different distortion-complexity optimization algorithms for the H.264 encoder, which include our four algorithms. We review the generalized Breiman, Friedman, Olshen, and Stone (GBFOS) algorithm [26] which is the basis for three of our algorithms. Finally, we describe the multi-objective particle swarm optimizer [27] and formulate it for distortion-complexity (D-C) optimization of the H.264 encoder.

2.2 Motivation

The H.264 video encoder compresses a raw video to a specified bitrate, using parameters defined by the user. In a videoconferencing application running on a cell phone, it is critical that the encoder runs fast and gives good quality. Since parameter settings affect both encoding speed and video quality, we would like to select parameter settings that trade off well between the two quantities.

Since we have prior knowledge of the available network bandwidth, we use it as our

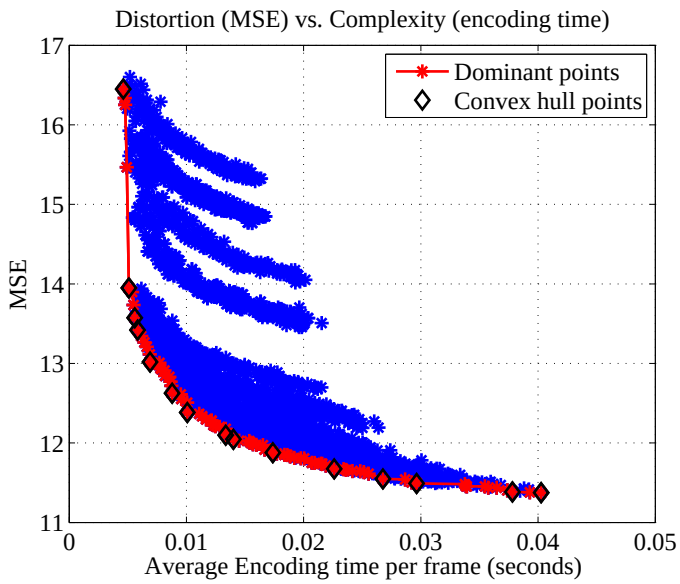


Figure 2.1: Distortion (MSE) vs. complexity (average encoding time).

bitrate. We use the average mean squared error (MSE) of the luma component as the distortion and the average encoding time per frame as the complexity corresponding to a particular parameter setting. Each point in Figure 2.1 gives the distortion and complexity corresponding to a certain parameter setting. A point in the distortion-complexity plot is *dominant* if there exist no other point having both a lower distortion and complexity. As shown in Figure 2.1, a point may not lie on the convex hull, but can still be dominant. Optimal parameter settings can be found offline by running the encoder for all possible parameter settings and obtaining a plot as in Figure 2.1, and selecting the parameter settings corresponding to the dominant points. This approach is referred to as *exhaustive search*, and it is very time consuming even with a fixed bitrate and current fast computers.

As an example, if we have 16 reference frames; 7 sub-pixel motion estimation methods; 10 partition sizes for intra and inter prediction; and 3 quantization approaches, the exhaustive search will require $16 \times 7 \times 10 \times 3 = 3360$ encodings. Applying these encodings to 10 quarter common intermediate format (QCIF) videos with an average encoding time of 65 s per video per parameter setting on a 2.8 GHz Linux computer will result in total search time of

$10 \times 3360 \times 65\text{s} = 25$ days. (The parameters used in this example are described in Section 1.3.) The search time would further increase with more variable encoder parameters, larger video resolution, longer videos, and a larger number of training videos.

Although the exhaustive search time can be reduced by parallel computing, it does not scale well with additional variable parameters. For example, when running an exhaustive search in parallel on 10 computers the search time would reduce to 2.5 days. However, if we included four additional parameters with 32 possible choices in the above search, the search time on 10 machines would increase to 32×2.5 days $= 80$ days. This illustrates the need for fast parameter settings selection algorithms. We have developed four fast parameter settings selection algorithms that are significantly faster than an exhaustive search but have similar performance. We describe our algorithms later in this chapter.

2.3 Related Work

Joint rate-distortion-complexity (R-D-C) optimization has been proposed for still image coding in [28, 29]. In these methods, the optimization is performed by augmenting the complexity constraints to the traditional R-D optimization using Lagrange optimization. In [29], a quadtree-based DCT codec is considered, and the complexity optimization is done on the inverse DCT operation at the decoder. In [28], R-D-C optimization is performed on a tree-structured vector quantizer. It reduces the encoder complexity by using a simpler distortion measure.

R-D-C or R-D optimization of motion-compensated transform-coded video compression is complex. The complexity is due to the large number of parameter selections for each frame, the many conditional decisions made in encoding, and the dependent coding of multiple frames. Even intelligently derived R-D optimization algorithms have a search complexity on the order of 10^{12} to 10^{14} to find R-D optimal frame coding parameters [30].

Complexity optimization of video encoders has been addressed in [31–33]. However, these methods are largely *ad hoc* and heuristic, and no actual optimization is performed. The optimization involves including only a subset of coding parameter choices and enforcing algorithmic simplification. The effect of each parameter choice and simplification on both performance and complexity reduction are quantified. In [34], an initial attempt towards

complexity optimization is made. In this approach, the encoding choices are limited, and a local search is performed on them to minimize complexity under a distortion constraint. It assumes that distortion is convex in the coding parameters and rate is not considered.

Distortion-complexity optimization of the H.263 encoder has been proposed in [35], where they use an adaptive mode control for choosing the motion estimation, sub-pixel accuracy and DCT pruning on a per-frame basis. In multimedia transmission, R-D-C adaptation of the bitstream to receiver constraints and network conditions has been discussed in [36]. In [37], for the low complexity H.264 decoder, bitstreams with I-frames or I and P-frames were found to have better D-C performance over bitstreams with B-frames, for the same bitrate.

2.4 Comparison with other Algorithms

Existing work on complexity optimization of the H.264 encoder can be classified into software offline methods [38]; software online (i.e., real-time) methods; and hardware optimization methods [39–48]. Software online methods can be further classified into those that adapt to computational or power resources [49–56] and non-adaptive methods [57–61]. Most of these algorithms modify one or more encoder components to achieve either speed improvement or power saving. Our approach differs from prior work since we do not modify the encoder but optimize it by selecting suitable encoder parameter settings.

The complexity optimization of the motion estimation and mode decision processes in [57–61] do not adapt to computational resources. A joint online adaptive rate-distortion-complexity (R-D-C) optimization scheme is presented in [49], where the product of complexity controlled Lagrange multiplier and complexity parameter is added to the R-D cost of motion search and mode decision.

For low power mobile devices, different online schemes have been proposed to adapt the encoder to power availability [50–53, 55]. He *et al.* introduced a complexity scalable video coder that uses a parametric power-rate-distortion (P-R-D) model [50]. This P-R-D optimized encoder has been tested only on a PC and not on a cell phone. The optimization is performed by varying the number of sum of absolute difference computations in motion estimation, the number of DCT computations, and the frame rate. However, such paramet-

ric models have large computational overhead. In [51], He *et al.* designed a P-R-D model for energy optimization of generic video encoders on portable devices and showed that it provides significant power savings in delay-tolerant applications. Su *et al.* proposed a joint power-distortion optimization scheme under a power constraint for the H.264 encoder [53]. During motion estimation, the additional search to refine the integer pixel motion vectors and the use of sub-pixel motion estimation are skipped based on available computational resources. The mode decision is approximated by predicting the macroblock partition size based upon the partition size used in the spatial-temporal neighborhood.

Lu *et al.* present a complexity-distortion model that controls the number of non-zero macroblocks and the number of sum of absolute differences [54]. However, it does not control other encoder parameters such as sub-pixel motion estimation method, partition size, and the number of reference frames. Agrawal *et al.* proposed adapting the encoder to the available battery power by controlling the bits transmitted or frames dropped [55]. However, decreasing bitrate and dropping frames can lower video quality. Li *et al.* developed a wireless transmission system that assigns more bits and transmission power to the foreground than to the background [52]. This ensures that the foreground has better decoded quality. However, they do not adapt the encoder complexity to the available battery power.

An offline scheme for R-D-C optimization of the H.264 encoder at high bitrates (1.5 Mb/s) was presented in [38], where the R-D-C optimization is reduced to R-C optimization. They optimize integer motion estimation and do not optimize any other encoder components. Also, they do not test for lower bitrates and progressive video. This offline approach has been extended to an online approach in [56] by using a complexity prediction model.

There has been prior work in developing hardware architecture or hardware specific solutions to speed up the H.264 encoder. It includes H.264 encoder optimization and H.264 encoder implementation for specific processors [39–41] or programmable firmware [62]. In addition, hardware architecture solutions have been proposed for different modules of the H.264 encoder [42], such as for motion estimation [43], intra prediction [44], transformation [45], CABAC [46], CAVLC [47], and deblocking filter [48]. Although hardware or platform specific optimization can speed up the encoder, it does not provide the flexibility of including newer algorithms or tools to the encoder.

Our approach is similar to [35], where parameter settings are chosen such that the H.263 encoder is optimized for D-C. However, they use variable bitrate and exhaustive search for finding the parameter settings offline, which can be time consuming. We have found no prior work in the area of fast parameter settings selection algorithms. In this thesis, we propose fast offline parameter settings selection algorithms that jointly optimize the H.264 encoder for distortion-complexity without altering the encoder. Our algorithms can be applied to any video encoder which has multiple parameters that each has more than one choice. Because there is no component in our algorithms that is specific to low resolution and low bitrate, our algorithms should also work well at higher bitrates and higher resolution videos.

2.5 The GBFOS Algorithm

Breiman et al. developed an algorithm for classification and regression trees, which traded off the number of leaves with probability of error for a classification tree and mean squared error for a regression tree [63], [64]. Chou et al. generalized this algorithm to a wider range of problems, and this algorithm is called the GBFOS algorithm [26]. A real-valued function of trees such as average distortion or rate is called a *tree functional* [26]. The GBFOS algorithm requires the tree functionals to be monotonic and linear or affine.

Let S be a subtree of a complete tree T that shares the same root as shown in Figure 2.2, and it is denoted by $S \preceq T$. Let u_1 and u_2 be two monotonic tree functionals, where u_1 is monotonically increasing, and u_2 is monotonically decreasing. The GBFOS algorithm minimizes the tree functional $J(S) = u_1(S) + \lambda u_2(S)$ over all $S \preceq T$, where the parameter $\lambda = -\Delta u_2/\Delta u_1$, and Δu_1 and Δu_2 are the change in u_1 and u_2, respectively, due to pruning off all branches from any node within the tree T. The parameter λ can be interpreted as a Lagrangian multiplier and as a slope of the distortion vs. rate curve (u_1 is associated with average rate and u_2 with average distortion). The GBFOS algorithm prunes off branches of a tree in order of increasing λ to find its subtrees having optimal distortion-rate performance. These optimal subtrees are nested [4], [64].

We illustrate the GBFOS algorithm with an example. Given a codebook or tree T, the GBFOS algorithm finds all codebooks or nested subtrees S having the same root node as T, and having R-D points that lie on the lower convex hull. For example, in Figure 2.3(a) '*'

represent all possible codebooks or nested subtrees of T, and in Figure 2.3(d) $S_0, S_1, S_2, S3$ are the GBFOS algorithm codebooks [3]. The first GBFOS codebook corresponds to the entire tree T, which is given by the rightmost point in Figure 2.3(a). This codebook has the lowest distortion and the highest rate. The root node t_0 corresponds to the leftmost point, since it has the highest distortion and the lowest rate. During each GBFOS iteration, the space to be examined is limited to the rectangular region bounded by the root node and the current subtree as illustrated in Figure 2.3(a), since rate is monotonically increasing and distortion is monotonically decreasing, with decreasing tree depth. During each iteration, the slopes from the current subtree to all subtrees that could be created by pruning one internal node are computed, and the subtree corresponding to the minimum slope is selected as it lies on the lower R-D convex hull. In Figure 2.3(b), the different lines corresponds to the different slopes that are compared. The solid line corresponds to the minimum slope. In future iterations, the search space is further reduced as illustrated by the shaded rectangular region in Figure 2.3(c). The GBFOS iteration continues until the root node is reached.

The GBFOS algorithm has also been used for optimal bit allocation for classified vector quantizer (VQ) with multiple classes [4]. In the fixed bitrate case, each class has a codebook with the same number of codewords. However, for the variable rate case, each class consists of different sized codebooks as shown in Figure 2.4. Therefore, using different sized codebooks for different classes will result in unequal bit allocation among classes, resulting in variable bitrate. The distortion-rate curves for different classes are obtained together with their slopes λ. The algorithm compares slopes, finds the class with the least slope and deallocates its bits. The resulting choice of codebooks for each class will have average distortion and rate that lie on the lower convex hull of the R-D plot.

2.6 GBFOS algorithm for D-C optimization of the H.264 encoder

We apply the GBFOS algorithm used in R-D optimization of classified VQ with multiple classes to the D-C optimization of the H.264 encoder. In the H.264 encoder, we have M parameters instead of M classes. Each H.264 parameter has different options with different D-C performance, similar to each class having codebooks of different sizes (and different R-D performance). Therefore, the H.264 parameters can be illustrated as a tree shown in

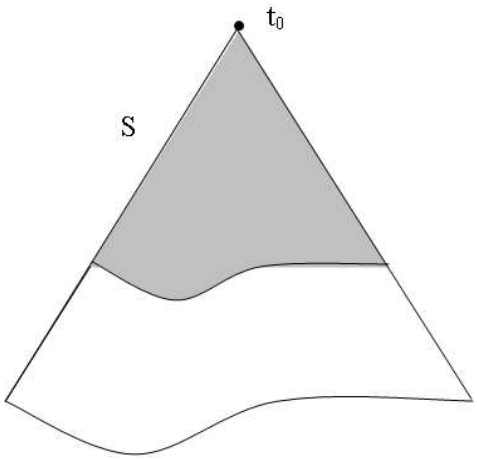


Figure 2.2: T is a whole tree with root t_0. S is a pruned subtree of T.

Figure 2.5. Instead of choosing codebooks from each class that result in overall optimal R-D performance, our objective is now to find H.264 parameter options that will result in lowest distortion for a given complexity.

In classified VQ with multiple classes, the tree functionals rate and distortion are independent, monotonic, and linear. For the H.264 encoder, the tree functionals encoding time and mean squared error for different parameters are clearly not independent, but we shall make an assumption that they are, which will result in parameter settings that are not necessarily optimal. We approximate by varying one of the encoder parameters while setting other parameters to their highest option, and attribute the resulting distortion-complexity to the variable parameter alone.

We describe two approaches that use the GBFOS algorithm for obtaining encoder parameter settings. The first approach, which we call the *GBFOS-basic* algorithm, requires only one iteration of encodings. The second algorithm requires more than one iteration of encodings, and we call this the *GBFOS-iterative* algorithm [65]. Both of our algorithms use the same preprocessing step. We found that the GBFOS-basic algorithm is very similar to

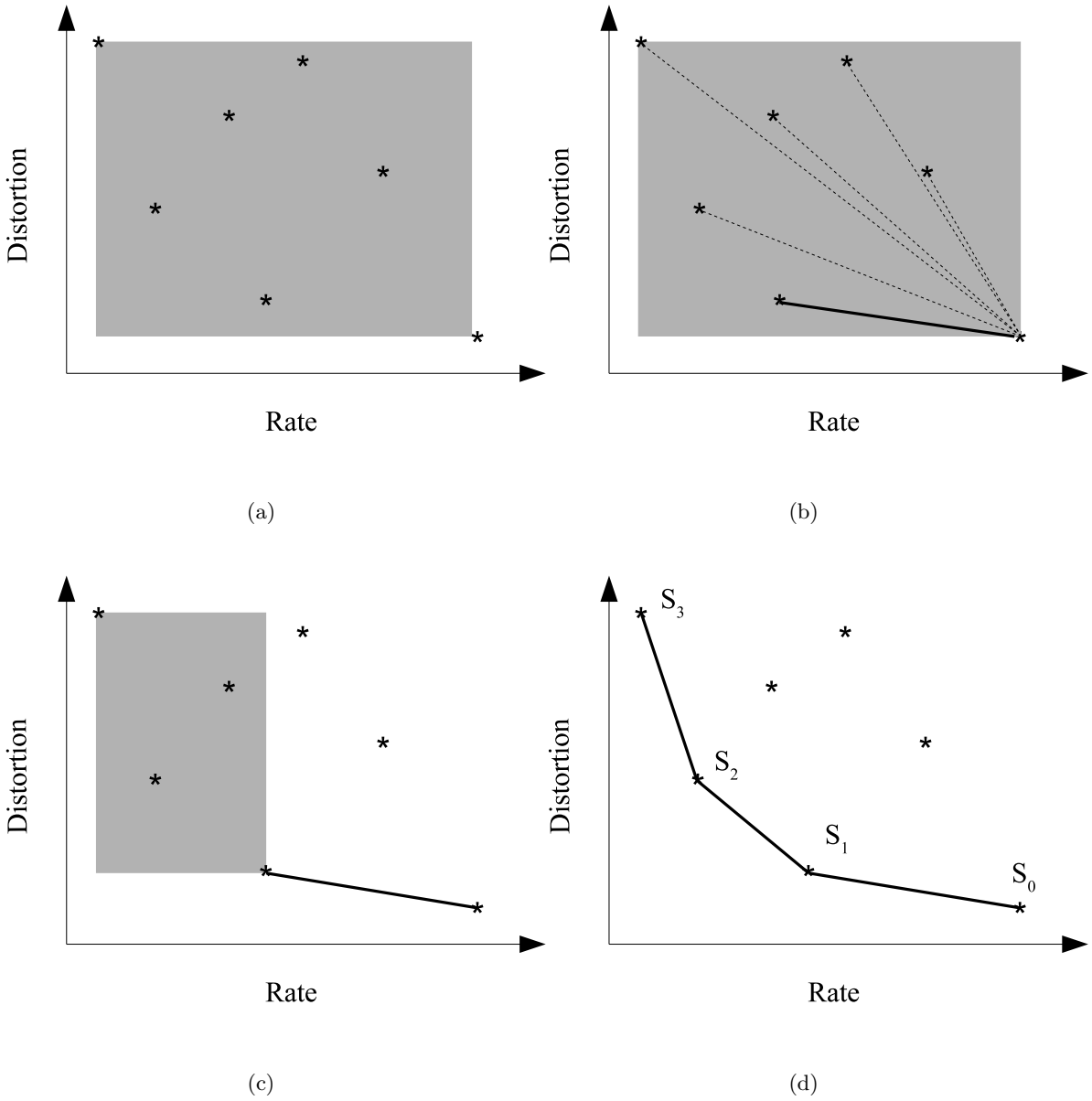


Figure 2.3: The GBFOS algorithm. (a) The leftmost point is the root node, and the rightmost point the current subtree. A '*' corresponds to a pruned nested subtree of the current subtree. The shaded rectangular region is the search space. (b) During each iteration, the slopes are compared between the current subtree and pruned subtrees. The next GBFOS codebook corresponds to the pruned subtree having the least slope indicated by the solid line. (c) Future iterations have smaller search area. (d) The GBFOS codebooks are S_0, S_1, S_2, S_3. [3]

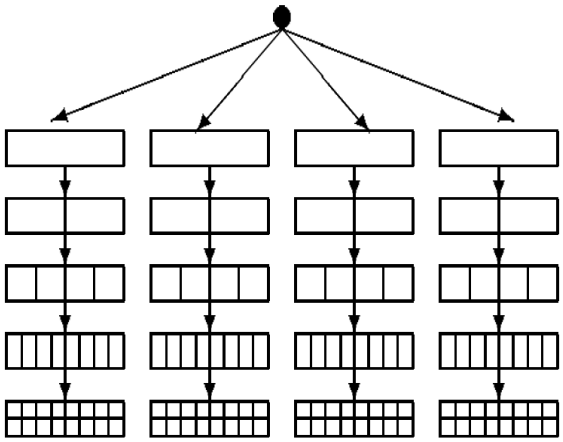


Figure 2.4: $M-1$ tree model for optimal bit allocation [4].

the multiple choice knapsack problem greedy (MCKP-greedy) algorithm [66].

The first parameter setting generated by our algorithms has the highest complexity. As successive parameter settings are generated, their complexity decreases. On the D-C plot in Figure 2.9, the parameter settings would start at the rightmost point on the plot and end at the leftmost point on the plot.

We explain our GBFOS algorithms with an illustrative example using the x264 encoder having four variable parameters, namely `ref`, `part`, `subme`, and `trellis`, which have been described in Chapter 1. We use $\{1,\ldots,M\}$ to denote a set of M encoder parameters, with each parameter i having n_i options $\{1,\ldots,n_i\}$. We denote a parameter setting as a vector $\mathbf{m}=(m_1,\ldots,m_M)$. An example of an encoder parameter setting is the vector $(1,1,1,3)$ that specifies an H.264 encoder with one reference frame; a partition size of P8 $\times$ 8; the fastest sub-pixel motion estimation scheme; and the use of trellis algorithm for quantization and CABAC for entropy coding. If $\mathbf{X}=\{\mathbf{m}_1,\ldots,\mathbf{m}_i,\ldots,\mathbf{m}_k\}$ is a set of parameter settings, then parameter setting $\mathbf{m}_i$ is dominant if there exists no parameter settings in $\mathbf{X}$ having both lower encoding time and higher PSNR.

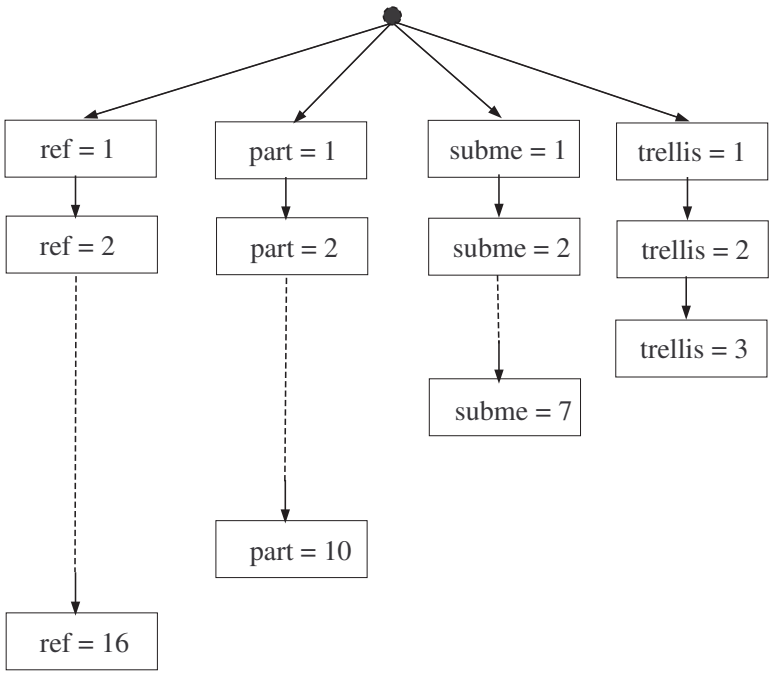


Figure 2.5: Illustrating the H.264 encoder parameters as a tree model. A class is replaced by a parameter, and VQ codebooks are replaced by parameter options.

2.6.1 Preprocessing Step

In this step, the D-C plot for each parameter is obtained by varying the parameter over all options while setting other parameters to their highest option. For example, to obtain the D-C plot for `ref` in Figure 2.6 (a), we set `part` $= 10$, `subme` $= 7$ and `trellis` $= 3$, and vary `ref` from 1 to 16. This results in 16 parameter settings $\mathbf{m}_{1,j} = (j, 10, 7, 3)$, for $1 \leq j \leq 16$, each of which is used to encode all the training videos. For each parameter setting $\mathbf{m}_{1,j}$, we use the average MSE over the training set as the distortion $d_1(\mathbf{m}_{1,j})$ and the average encoding time as the complexity $c_1(\mathbf{m}_{1,j})$. (In general, for each parameter i, we vary the parameter setting $\mathbf{m}_{i,j} = (n_1, \ldots, n_{i-1}, j, n_{i+1}, \ldots, n_M)$, for $1 \leq j \leq n_i$, and use it to encode training videos to obtain its corresponding distortion $d_i(\mathbf{m}_{i,j})$ and complexity $c_i(\mathbf{m}_{i,j})$.) Figure 2.6 gives the D-C plots for `ref`, `part`, `subme` and `trellis` obtained by running an x264 encoder on ASL videos at 30 kb/s.

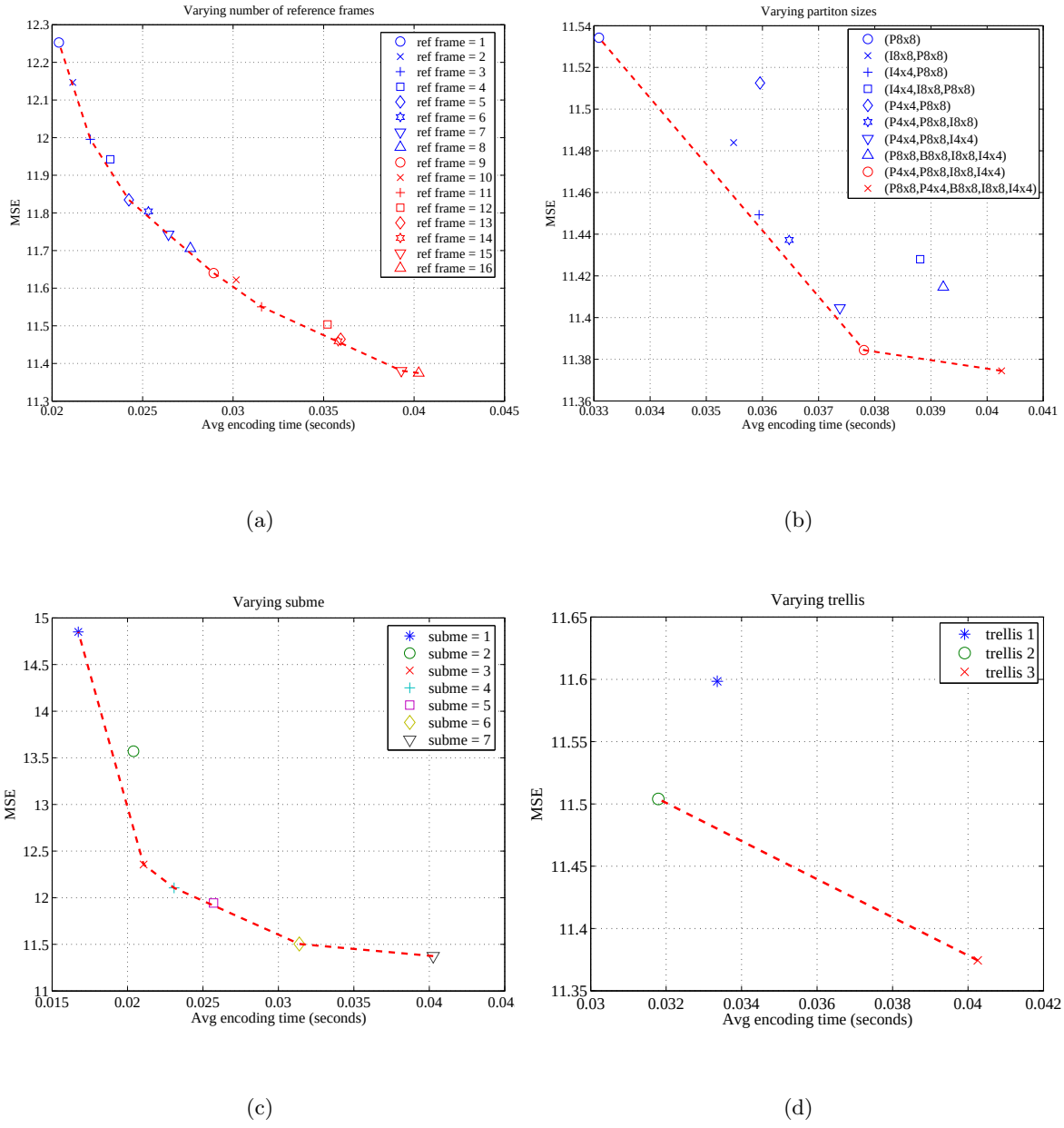


Figure 2.6: Distortion-complexity plots obtained by varying (a) number of reference frames, (b) partition sizes (the options are listed in the legend), (c) subme values and (d) trellis options. The convex hull points of each plot are shown connected by a dashed line.

We denote the number of points on the D-C convex hull of parameter i as l_i. (The number of convex hull points l_i may be less than the total number of parameter options n_i.) For example, in Figure 2.6(b), the parameter `part` has three points on the convex hull ($l_2 = 3$), although it has 10 options. We reindex the points on the convex hull of parameter i from 1 to l_i and store its original indices in vector $\mathbf{orig}_i$. We denote the parameter setting of a convex hull point by $\mathbf{q}_{i,j} = (n_1, \ldots, n_{i-1}, j, n_{i+1}, \ldots, n_M)$, where $1 \leq j \leq l_i$. In Figure 2.6(b), the parameter settings on the convex hull are given by $\mathbf{q}_{2,j} = (16, j, 7, 3)$, where $1 \leq j \leq 3$, corresponding to the three different partition size options (P8 × 8), (P8 × 8, P4 × 4, I8 × 8, I4 × 4), and (P8 × 8, P4 × 4, B8 × 8, I8 × 8, I4 × 4). Because the selected partition sizes were the first, ninth and tenth options in Figure 2.6(b), $\mathbf{orig}_2 = (1, 9, 10)$ and $\mathbf{orig}_2(l_2) = \mathbf{orig}_2(3) = 10$.

We define the slope between two points on the D-C convex hull as

$$S_i(\mathbf{q}_{i,j}, j) = abs\left(\frac{d(\mathbf{q}_{i,j}) - d(\mathbf{q}_{i,j-1})}{c(\mathbf{q}_{i,j}) - c(\mathbf{q}_{i,j-1})}\right), \tag{2.1}$$

where $abs()$ means absolute value. The parameter settings $\mathbf{q}_{i,j}$ and their corresponding slopes are used in the initialization step of both the GBFOS-basic and iterative algorithms.

2.7 GBFOS-Basic Algorithm

The GBFOS-basic algorithm uses only the D-C plots generated in the preprocessing step. Pseudocode for the GBFOS-basic algorithm is given in Figure 2.7. The first parameter setting is found in step 1, by selecting the least MSE option on the D-C convex hull of each parameter. Continuing our example, in Figure 2.6, these parameters would be `ref` = 16, `part` = 10, `subme` = 7 and `trellis` = 3, i.e. $\mathbf{p} = (16, 10, 7, 3)$. Next, the following steps are repeated until there are no slopes left to prune:

1. Compare the slopes on the convex hull of the D-C plots to find the parameter with the least slope.

2. Prune the least slope. In Figure 2.6, `part` has the least slope, so we prune it from 10 to 9.

1. Obtain $\mathbf{p} = (\mathbf{orig}_1(l_1), \ldots, \mathbf{orig}_i(l_i), \ldots, \mathbf{orig}_M(l_M))$ from $\mathbf{q}_{i,j}$, where $1 \leq i \leq M$ and $1 \leq j \leq l_i$.

2. Repeat until $\mathbf{p} = (\mathbf{orig}_1(1), \ldots, \mathbf{orig}_M(1))$:

   (a) Choose parameter i^* that results in the lowest slope $S_{i^*}(\mathbf{q}_{i^*,l_{i^*}}, l_{i^*})$ using Equation (2.1).

   (b) Set $l_{i^*} = l_{i^*} - 1$.

   (c) Determine new $\mathbf{p} = (\mathbf{orig}_1(l_1), \ldots, \mathbf{orig}_{i^*}(l_{i^*}), \ldots, \mathbf{orig}_M(l_M))$.

Figure 2.7: The GBFOS-Basic algorithm.

3. Find the new parameter setting $\mathbf{p}$. In our example, $\mathbf{p} = (16, 9, 7, 3)$.

For both the GBFOS-basic and iterative algorithms, when a parameter has only one D-C point left, it is no longer considered for pruning. We post-process the GBFOS-basic parameter settings by selecting only those parameter settings whose associated D-C points are dominant.

2.8 GBFOS-Iterative Algorithm

While the GBFOS-basic algorithm uses one set of D-C plots as shown in Figure 2.6, the GBFOS-iterative algorithm generates new D-C plots during each iteration to result in new parameter settings. In the first iteration, it uses the D-C plots generated by the preprocessing step but it provides the possibility of finding improved parameter settings in subsequent iterations.

The first two parameter settings of the GBFOS-iterative algorithm are the same as for the GBFOS-basic algorithm. The following steps are repeated until there are no slopes left to prune:

1. Regenerate the D-C plots for the parameters not pruned in the previous iteration. In our example, we prune `part` in the first iteration and in the second iteration we obtain

new D-C plots for `ref`, `subme` and `trellis` using the new value of `part`. For `ref`, we set `part` = 9, `subme` = 7 and `trellis` = 3 and vary `ref` from 1 to 16 to obtain the new D-C plot. (If `ref` was pruned in an earlier iteration to say 9, then we vary `ref` from 1 to 9.)

2. Compare the slopes for all the encoder parameters.

3. Prune the least slope. In our example, we prune `subme` from 7 to 6.

4. Find the new parameter setting, which in our example is $\mathbf{p} = (16, 9, 6, 3)$.

Pseudocode for the GBFOS-iterative algorithm is given in Figure 2.8. In step 2c(ii) of Figure 2.8, the number of points on the convex hull l_i' may be less than l_i. Therefore, the parameters are reindexed from 1 to l_i'.

2.9 Dominant parameter setting pruning algorithm

The GBFOS-basic and GBFOS-iterative algorithms choose parameter settings whose D-C performance are close to the optimal parameter settings obtained from exhaustive search. Unfortunately, both the GBFOS-basic and GBFOS-iterative algorithms yield only a few parameter settings. Figure 2.9 illustrates the GBFOS-basic points and shows the regions where the GBFOS-basic points are sparse.

Since current cell phones have low computational capability, running the high complexity x264 encoder causes frames to be dropped, resulting in a low frame rate. To ensure that the video is encoded at a higher frame rate, one would need to choose a very low complexity parameter setting, which would deteriorate video quality. Therefore, it is important to choose the right parameter setting that can satisfy an encode time constraint and yield higher PSNR. In the absence of a parameter setting for a given encode time, the encoder must use a lower complexity/lower PSNR GBFOS-basic parameter setting.

An encoder could achieve a given average encode time per frame by using different GBFOS-basic parameter settings over different frames. This is referred to as *time-sharing*. Time-sharing may not be feasible in a scenario where different entropy coding schemes have

1. Obtain $\mathbf{p} = (\mathbf{orig}_1(l_1), \ldots, \mathbf{orig}_i(l_i), \ldots, \mathbf{orig}_M(l_M))$ from $\mathbf{q}_{i,j}$, where $1 \leq i \leq M$ and $1 \leq j \leq l_i$.

2. Repeat until $\mathbf{p} = (\mathbf{orig}_1(1), \ldots, \mathbf{orig}_M(1))$ in step 2(b):

   (a) Find i^* that results in the lowest slope $S_{i^*}(\mathbf{q}_{i^*,l_{i^*}}, l_{i^*})$ using Equation (2.1).

   (b) Set $l_{i^*} = l_{i^*} - 1$. New $\mathbf{p} = (\mathbf{orig}_1(l_1), \ldots, \mathbf{orig}_{i^*}(l_{i^*}), \ldots, \mathbf{orig}_M(l_M))$.

   (c) For each $i \neq i^*$ and $l_i > 1$:

      i. Obtain $d_i(\mathbf{m}_{ij})$ and $c_i(\mathbf{m}_{ij})$ for $\mathbf{m}_{ij} = (\mathbf{orig}_1(l_1), \ldots, \mathbf{orig}_{i-1}(l_{i-1}), j, \mathbf{orig}_{i+1}(l_{i+1}), \ldots, \mathbf{orig}_M(l_M))$, where $1 \leq j \leq \mathbf{orig}_i(l_i)$ and $j \neq \mathbf{orig}_i(l_{i+1})$. Obtain the D-C plot and convex hull points of parameter i.

      ii. Reindex the parameters of the convex hull points from 1 to l_i'. Reconstruct the vector $\mathbf{orig}_i$ of length l_i', which contains original indices corresponding to $(1, \ldots, l_i')$.

      iii. Set $l_i = l_i'$.

Figure 2.8: The GBFOS-Iterative algorithm.

to be used for different frames. In related work to avoid time-sharing, Kiang *et al.* presented the Recursive Optimal Pruning Algorithm (ROPA) [67] for finding more tree-structured vector quantizer codebooks compared to those generated by the GBFOS algorithm [26]. ROPA guarantees the minimum area under the rate-distortion curve corresponding to a codebook.

To avoid time-sharing in our problem, we propose two different algorithms for finding additional parameter settings over the GBFOS-basic algorithm. One of them is similar to ROPA while the other performs a local search between two adjacent GBFOS-basic algorithm parameter settings. These additional parameter settings allow the encoder to operate at a higher PSNR.

Similar to ROPA, the DPSPA generates more parameter settings over the GBFOS-basic algorithm, while still including the GBFOS-basic parameter settings. The DPSPA

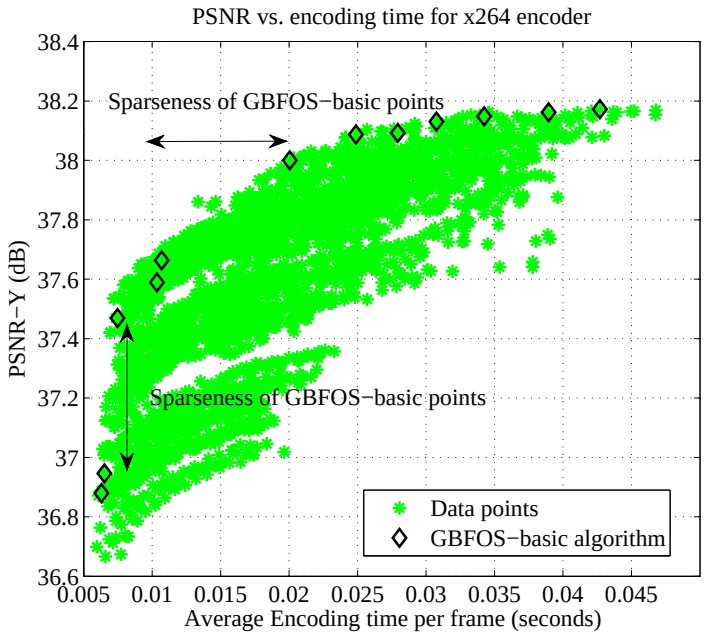


Figure 2.9: Distortion (PSNR) vs. complexity (average encoding time). Each data point here corresponds to a parameter setting.

differs from the GBFOS-basic algorithm in the slope pruning stage because it considers the dominant non-convex hull parameter settings, as shown by points C and D in Figure 2.10.

The algorithm is:

1. Preprocessing step. Generate the D-C plot for each parameter using the procedure given in the preprocessing step of Section 2.6.1, and find the D-C convex hull parameter settings and compute their slopes using Equation (2.1). For each D-C plot, we find regions that can have dominant points as shown by the shaded region in Figure 2.10. Points outside this region are discarded since they are dominated by the convex hull points. For each pair of convex hull points, find the dominant points that lie in its shaded region. Figure 2.10 is the D-C plot for the `subme` parameter containing convex hull points A and B and their dominant non-convex hull points C and D. Let the points A, B, C, and D correspond to parameter settings $(16, 10, 7, 3)$, $(16, 10, 4, 3)$,

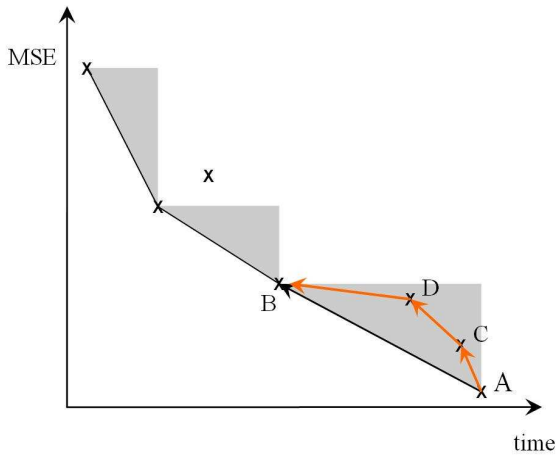


Figure 2.10: Dominant Parameter Setting Pruning Algorithm (DPSPA). The dominant non-convex hull points are C and D. Path traversed by the DPSPA is indicated by red arrows and the path of the GBFOS-basic algorithm is indicated by the black arrow.

$(16, 10, 6, 3)$, and $(16, 10, 5, 3)$, respectively.

2. The first parameter setting is the same as in the GBFOS-basic algorithm. Let $\mathbf{p}$ = A $= (16, 10, 7, 3)$ be the first parameter setting.

3. Repeat the following steps until there are no convex hull slopes left to prune:

   (a) Compare the slopes of the convex hull for different parameters and find the parameter with the least slope. For example, let `subme` in Figure 2.10 have the least slope in the first iteration, and let it correspond to the line segment AB.

   (b) In the GBFOS-basic algorithm, we would prune AB. In the DPSPA, we instead check if the slope contains dominant non-convex hull points that lie in the shaded region. If so, we generate new parameter settings by pruning each such point. For example, in Figure 2.10, we prune AC, CD, and DB to result in three new parameter settings C = $(16, 10, 6, 3)$, D = $(16, 10, 5, 3)$ and B = $(16, 10, 4, 3)$.

Step 3(b) generates additional parameter settings over the GBFOS-basic algorithm. We post-process the DPSPA parameter settings similarly to the GBFOS-basic algorithm by choosing only the non-dominated DPSPA parameter settings.

2.10 Controlled Local Search Algorithm

Our controlled local search algorithm (CLSA) is similar to the popular local search algorithm in combinatorial optimization [68]. Our algorithm performs a parameter setting search between two GBFOS-basic parameter settings. During each search iteration, new parameter settings are generated, which form the starting points for the future iterations.

Let the start and end parameter settings from the GBFOS-basic algorithm be denoted by vectors $\mathbf{p} = (p_1, \ldots, p_M)$ and $\mathbf{q} = (q_1, \ldots, q_M)$, respectively. We denote the PSNR-Y and encode time of a parameter setting $\mathbf{m}$ as $d(\mathbf{m})$ and $c(\mathbf{m})$, respectively. The function $min(.,.)$ gives the minimum of the two numbers. The resulting parameter setting generated by our algorithm is stored in set $\mathbf{B}$.

The pseudocode for our local search algorithm is given in Figure 2.11. In each iteration, we consider a new start parameter setting $\mathbf{x}$ while the end parameter setting is always $\mathbf{q}$. The subroutine expansion(.) defines the neighborhood of a parameter setting $\mathbf{x}$. We generate at most two parameter settings for each parameter in $\mathbf{x}$ by incrementing it by one and two. For example, if $\mathbf{x}$ = (1,1,1,1) then we generate parameter settings $(2, 1, 1, 1)$ and $(3, 1, 1, 1)$ by increasing the first parameter in $\mathbf{x}$.

In Step (6) of Figure 2.11, we restrict the number of parameter settings in the neighborhood of $\mathbf{x}$, otherwise the number of parameter settings generated per iteration would grow exponentially. We use the concept of dominance in pruning parameter settings in Step (6a). In Step (6b), we retain only those parameter settings in $\mathbf{E}$ that have encode time between $\mathbf{x}$ and $\mathbf{q}$, and discard the rest. In Steps (7) and (8), we discard any pruned set that does not contain $\mathbf{q}$, to avoid generating parameter settings with PSNR and encode time beyond $\mathbf{q}$.

Our algorithm searches from a lower PSNR and lower complexity parameter setting to a higher PSNR and higher complexity parameter setting. A descent algorithm could be formulated similar to Figure 2.11 with start and end parameter settings interchanged

Controlled Local Search Algorithm($\mathbf{p}$, $\mathbf{q}$)

1. Let $\mathbf{A} = \{\mathbf{p}\}$ and $\mathbf{B} = \{\mathbf{q}\}$.
2. while $\mathbf{A} \neq \emptyset$
3. Choose $\mathbf{x} \in \mathbf{A}$
4. $\mathbf{B} = \mathbf{B} \cup \{\mathbf{x}\}$
5. $\mathbf{E}$ = expansion($\mathbf{x}$).
6. Remove from $\mathbf{E}$ any $\mathbf{y} \in \mathbf{E}$ that satisfies either conditions

   (a) $\mathbf{y}$ is dominated by any element in $\mathbf{E} \cup \mathbf{B}$.
   (b) $c(\mathbf{y}) > c(\mathbf{q})$ or $c(\mathbf{y}) < c(\mathbf{x})$.

7. if ($\mathbf{q} \in \mathbf{E}$)
8. $\mathbf{A} = \mathbf{A} \cup \mathbf{E} - \mathbf{B}$.
9. return $\mathbf{B}$

expansion($\mathbf{x}$)

1. $\mathbf{G} = \emptyset$
2. for $1 \leq i \leq M$: $\mathbf{G} = \mathbf{G} \cup \{\mathbf{b}_{2i-1}\} \cup \{\mathbf{b}_{2i}\}$,
   where $\mathbf{b}_{2i-1} = (x_1, \ldots, min(x_i + 1, n_i), \ldots, x_M)$ and $\mathbf{b}_{2i} = (x_1, \ldots, min(x_i + 2, n_i), \ldots, x_M)$.
3. return $\mathbf{G}$

Figure 2.11: Controlled local search algorithm.

and modifying the expansion(.) subroutine to decrease each parameter option. In our experiments, we find that the approach in Figure 2.11 finds more parameter settings in the low and mid encode regions while a descent algorithm finds more parameter settings in the high encode time region. Because our target application is real-time encoding over cell phones, we search from lower PSNR/complexity to higher PSNR/complexity to find more points in the low and mid encode time regions.

2.11 Multiobjective particle swarm optimizer

A multiobjective optimization algorithm optimizes several objective functions simultaneously. Our distortion-complexity optimization is a multiobjective optimization problem since it involves two objective functions: the MSE, and encoding time. As a comparator to the four algorithms presented in this thesis, we consider the multiobjective particle swarm optimizer (MOPSO). We specifically consider the MOPSO algorithm developed by Coello *et al.* [27, 69] because it has comparable performance to the nondominated sorting genetic algorithm-II, the Pareto archive evolutionary strategy, and the microgenetic algorithm for multiobjective optimization, but is faster [27].

We formulate our D-C multiobjective optimization problem as follows and apply the MOPSO algorithm: find $\mathbf{m} = (\texttt{ref}, \texttt{part}, \texttt{subme}, \texttt{trellis})$ which satisfies the following inequality constraints: $1 \leq \texttt{ref} \leq 16$; $1 \leq \texttt{part} \leq 10$; $1 \leq \texttt{subme} \leq 7$; and $1 \leq \texttt{trellis} \leq 3$, and optimize $d(\mathbf{m})$ and $c(\mathbf{m})$.

In MOPSO, the user specifies the initial number of D-C points or *particles* and the number of iterations. The initial particle positions are randomly chosen. During each iteration, particle positions are updated by a *velocity* step. The algorithm stores the nondominated particle positions in a repository. Each particle's best position so far is stored in a buffer, and the velocity and position of the particle are updated using knowledge of its best position so far and the previous global best solution. Due to the randomness associated with the algorithm during the initial particle generation and velocity update stages, the final solution differs with each run of the algorithm.

The similarity between MOPSO and our algorithms is that they both generate points on the Pareto Front. The Pareto Front points are dominant among a given set of points, which are a subset of exhaustive search points. The MOPSO finds the Pareto Front heuristically by randomly searching through the distortion-complexity space using the concept of dominance. Our algorithms find the Pareto Front by comparing the distortion-complexity slopes of different parameters. These slopes approximate a Lagrange multiplier used to combine $d(\mathbf{m})$ and $c(\mathbf{m})$ to a single cost function. In Chapter 3, we will show that the Pareto Fronts generated by MOPSO and our algorithms are not identical.

2.12 Summary

In this chapter, we presented four new parameter settings selection algorithms: the GBFOS-basic and GBFOS-iterative algorithms, DPSPA, and CLSA. The last two algorithms generate additional parameter settings over the GBFOS-basic algorithm. We described the MOPSO and applied it to our D-C optimization problem. We illustrated these algorithms using four x264 encoder parameters. In the following chapter, we compare the performance of these algorithms using different training and test videos, and different video encoders.

Chapter 3

PERFORMANCE OF OUR ALGORITHMS ON X264 AND H.263+ ENCODERS

3.1 Introduction

In this chapter, we present metrics to evaluate the performance of our algorithms presented in the previous chapter. We compare our algorithms with exhaustive search, multiobjective particle swarm optimizer (MOPSO), and the x264 default parameter setting. We present an ROI-based motion search method as a new x264 parameter to improve motion estimation speed for ASL videos. We include this as an additional parameter in our parameter setting space, and apply our optimization algorithms to select the parameter settings. Finally, we test our algorithms on the H.263+ encoder [24].

3.2 Performance Metrics

We denote the parameter settings from an exhaustive search as $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{N_{opt}}$, and the parameter settings generated by algorithm A as $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N_A}$. Both categories of parameter settings are ordered in increasing encoding time. Algorithm A represents a fast algorithm. Let $c(\mathbf{p}_i)$ and $\mathrm{MSE}(\mathbf{p}_i)$ denote the average encoding time and the mean squared error of the luma component of a parameter setting $\mathbf{p}_i$, respectively. The PSNR of the luma component of a parameter setting $\mathbf{p}_i$, averaged over an entire video is then defined as

$$\mathrm{PSNR}(\mathbf{p}_i) = 10 \log_{10} \frac{255^2}{\mathrm{MSE}(\mathbf{p}_i)}, \tag{3.1}$$

where $\mathrm{MSE}(\mathbf{p}_i)$ is the mean squared error of the luma component averaged over all frames in the video.

The following four metrics are used to compare the performance between the parameter settings from algorithm A and exhaustive search:

1. Maximum PSNR difference ($\mathrm{Max_A}$). This metric gives the maximum PSNR difference between parameter settings from algorithm A and exhaustive search. For each

parameter setting $\mathbf{p}_i$, we first find the parameter setting $\mathbf{r}_{k_i}$ that has nearly the same encoding time and its index is obtained as follows

$$k_i = \underset{1 \le j \le N_{opt}}{\arg\min} \left(c(\mathbf{r}_j) - c(\mathbf{p}_i) \ge 0\right), \tag{3.2}$$

The maximum PSNR difference is

$$\text{Max}_\text{A} = \max_{1 \le i \le N_A} (\text{PSNR}(\mathbf{r}_{k_i}) - \text{PSNR}(\mathbf{p}_i)). \tag{3.3}$$

2. Mean PSNR difference. This is defined as

$$\text{Mean}_\text{A} = \frac{1}{N_A} \sum_{i=1}^{N_A} (\text{PSNR}(\mathbf{r}_{k_i}) - \text{PSNR}(\mathbf{p}_i)), \tag{3.4}$$

where index k_i is obtained using Equation (3.2).

3. Hyperarea ratio (HR) [70]. This metric measures the spread of our algorithm's D-C points relative to the dominant points of the exhaustive search. A lower spread indicates that the algorithm's D-C points are clustered together, which means the encoding time range of the parameter settings is restricted. For example, if all the parameter settings have high encoding time, the encoder cannot choose a low complexity parameter setting appropriate for low processor speed, thereby resulting in slow encoding.

   We first define the hyperarea for algorithm A as

$$\text{HA}_\text{A} = \sum_{i=1}^{N_A} \text{MSE}(\mathbf{p}_i)(c(\mathbf{p}_i) - c(\mathbf{p}_{i-1})), \tag{3.5}$$

   where $c(\mathbf{p}_0) = 0$. Similarly, we compute the hyperarea for the dominant points HA_opt. The hyperarea ratio is defined as

$$\text{HR}_\text{A} = \frac{\text{HA}_\text{opt}}{\text{HA}_\text{A}}. \tag{3.6}$$

We would like the HR to be close to one, since this would mean a good D-C spread of the parameter settings. If the HR is greater than one, the parameter settings have a good D-C spread but have not converged to the dominant points.

4. Number of parameter settings generated. We seek to generate more parameter settings since they increase the possibility of finding higher PSNR parameter settings that satisfy an encoding time constraint.

3.3 Results for the H.264 Encoder

In our tests, we use video sequences from three data sets. One of the data sets contains 15 QCIF video sequences from [71] that are often used in MPEG codec evaluation. We will refer to them as the Standard data set. The other two data sets are American Sign Language videos created at the University of Washington, with one set (ASL-1) containing 10 video clips at QCIF resolution and the other set (ASL-2) containing 10 video clips at 320×240 resolution. The ASL-1 data set contains different videos from the ASL-2 data set. Table 3.1 lists videos in each data set. These video clips are encoded using x264 (March 26, 2006 version) [6] at 30 frames per second (fps). We choose three target bitrates: 30 kb/s, 150 kb/s and 300 kb/s. The average number of frames for ASL-1, ASL-2 and Standard test sets are 1073, 1687 and 720 frames, respectively. Tests are conducted on a Linux machine with a 2.8 GHz Intel CPU.

We consider the four variable parameters described in Section 2.6 and vary each parameter over their entire range. We use the following constant parameters in our encodings: 8×8 and 4×4 DCT; one B-frame; IPBPBP... group-of-picture structure; uneven multihexagon motion search; and spatial motion vector prediction for direct mode. The other parameters are set to their default options.

3.3.1 Training data results

We compare the performance of our algorithms to exhaustive search and MOPSO using the three data sets and three bitrates. Figure 3.1 shows the D-C curves for the ASL-1 data set at 30 kb/s and we find that GBFOS-basic and GBFOS-iterative points are

Table 3.1: List of videos.

ASL-1 data set	ASL-2 data set	Standard data set
Accident	BadLuckDitch	akiyo
Childhood	ScaryMovie	claire
DayAtSchool	CaptionedMovies	foreman
FavoriteActors	School	mother
FoodILike	DaughtersGraduation	bridge-close
Graduation	BadLuckLadder	grandma
MakeUpJob	WaterWorld	news
MonthInOregon	SistersCooking	bridge-far
Theater	FancyRestaurant	coastguard
VolleyBall	Maryland	highway
		salesman
		carphone
		container
		mobile
		silent

close to the dominant points. The performance of different algorithms for different data sets are given in Tables 3.2, 3.3 and 3.4. Figure 3.2 illustrates the performance of the GBFOS-basic algorithm, dominant parameter setting pruning algorithm (DPSPA), and MOPSO for the ASL-1 data set at 30 kb/s. The DPSPA generates additional parameter settings over the GBFOS-basic algorithm. Because the PSNR performance and hyperarea ratio for the GBFOS-basic algorithm and DPSPA are very similar, we only report the number of parameter settings for DPSPA in Table 3.2. From Table 3.2, the maximum PSNR difference for both the GBFOS-basic algorithm and the DPSPA is 0.58 dB. Their mean PSNR difference is significantly less, indicating that most points lie close to the dominant points. The GBFOS-iterative algorithm has a lower maximum PSNR difference

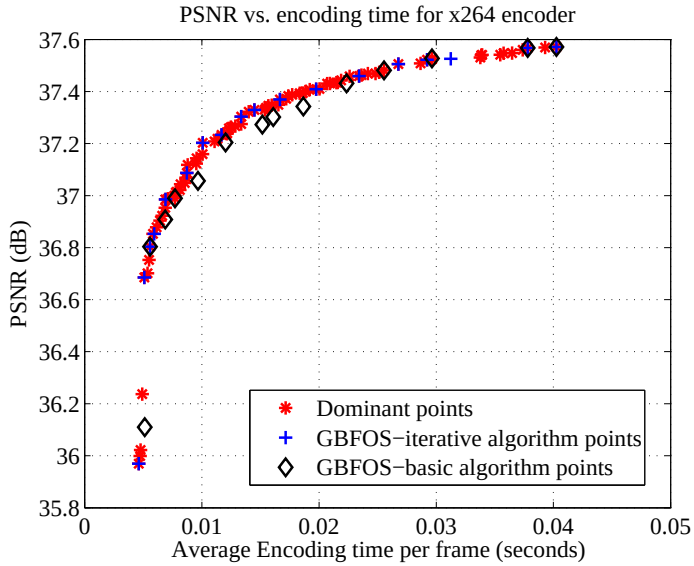


Figure 3.1: Distortion-complexity plot comparing GBFOS-basic and iterative points with the dominant points from exhaustive search for the ASL-1 data set at 30 kb/s.

of 0.36 dB, and for the ASL-1 data set it generates points that almost lie on the dominant points as shown in Table 3.2(c).

Since the parameter settings generated by the MOPSO algorithm differ during each run, we run it 100 times, find the 95% confidence interval for the four metrics, and report them in Table 3.2(b). The worst case 95% confidence interval for the maximum PSNR difference is [0.39 dB 0.45 dB], corresponding to the ASL-2 data set at 30 kb/s. The confidence interval for the mean PSNR difference is low indicating the MOPSO points often lie close to the dominant points. Compared to other algorithms, the MOPSO often generates fewer parameter settings and has lower HR indicating that its D-C points have lower spread. This is illustrated in Figure 3.2, where MOPSO points were obtained from a single run of the algorithm.

We next compare the number of encodings required by different algorithms. Let $n_1, ..., n_M$ be the number of options of each of the M parameters. Then the number of encodings re-

Table 3.2: Results corresponding to the ASL-1 data set. (a) Both the GBFOS-basic algorithm and DPSPA have similar performance and take 33 encodings. (b) The 95% confidence interval results for MOPSO. MOPSO also takes 33 encodings. We would like the Hyperarea ratio (HR) to be close to one. (c) Performance of the GBFOS-iterative algorithm.

Bitrates	GBFOS-basic algorithm				DPSPA
	PSNR diff (dB)		HR	# of points	# of points
	Max	Mean			
30 kb/s	0.58	0.07	0.995	14	24
150 kb/s	0.21	0.05	0.975	10	11
300 kb/s	0.45	0.03	1	18	25

(a)

Bitrates	MOPSO			
	PSNR diff (dB)		HR	# of points
	Max	Mean		
30 kb/s	[0.31 0.43]	[0.09 0.12]	[0.62 0.67]	[12 13]
150 kb/s	[0.26 0.34]	[0.10 0.12]	[0.67 0.77]	[9 10]
300 kb/s	[0.28 0.36]	[0.08 0.10]	[0.61 0.69]	[12 13]

(b)

Bitrates	GBFOS-iterative algorithm				
	PSNR diff (dB)		HR	# of points	# of
	Max	Mean			encodings
30 kb/s	≈ 0	≈ 0	0.99	18	223
150 kb/s	≈ 0	≈ 0	0.99	17	197
300 kb/s	0	0	0.99	20	235

(c)

Table 3.3: Training data results for the ASL-2 data set. (a) Both the GBFOS-basic algorithm and DPSPA have similar performance. (b) The 95% confidence interval results for MOPSO. (c) Performance of the GBFOS-iterative algorithm.

Bitrate	GBFOS-basic algorithm				DPSPA
	PSNR diff (dB)		HR	# of points	# of points
	Max	Mean			
30 kb/s	0.33	0.1	0.98	7	11
150 kb/s	0.49	0.13	1.01	15	20
300 kb/s	0.15	0.03	0.99	17	23

(a)

Bitrate	MOPSO			
	PSNR diff (dB)		HR	# of points
	Max	Mean		
30 kb/s	[0.39 0.45]	[0.15 0.17]	[0.86 0.94]	[9 9]
150 kb/s	[0.26 0.31]	[0.12 0.14]	[0.67 0.73]	[9 9]
300 kb/s	[0.19 0.24]	[0.07 0.08]	[0.60 0.67]	[10 11]

(b)

Bitrate	GBFOS-iterative algorithm				
	PSNR diff (dB)		HR	# of points	# of
	Max	Mean			encodings
30 kb/s	0.03	≈ 0	0.99	13	127
150 kb/s	0.01	≈ 0	≈ 1	13	118
300 kb/s	0.22	0.02	≈ 1	22	218

(c)

Table 3.4: Training data results corresponding to the Standard data set. (a) Both the GBFOS-basic algorithm and DPSPA have similar performance. (b) The 95% confidence interval results for MOPSO. We would like the Hyperarea ratio (HR) to be close to one.(c) Performance of the GBFOS-iterative algorithm.

Bitrate	GBFOS-basic algorithm				DPSPA
	PSNR diff (dB)		HR	# of points	# of points
	Max	Mean			
30 kb/s	0.12	0.04	0.98	9	10
150 kb/s	0.22	0.05	≈ 1	14	18
300 kb/s	0.27	0.06	0.99	14	19

(a)

Bitrate	MOPSO			
	PSNR diff (dB)		HR	# of points
	Max	Mean		
30 kb/s	[0.27 0.32]	[0.09 0.11]	[0.65 0.70]	[10 11]
150 kb/s	[0.16 0.2]	[0.08 0.09]	[0.37 0.44]	[10 11]
300 kb/s	[0.22 0.25]	[0.09 0.1]	[0.71 0.85]	[11 13]

(b)

Bitrate	GBFOS-iterative algorithm				
	PSNR diff (dB)		HR	# of points	# of
	Max	Mean			encodings
30 kb/s	0.18	0.04	0.99	12	155
150 kb/s	0	0	≈ 1	22	285
300 kb/s	0.36	0.04	0.99	15	229

(c)

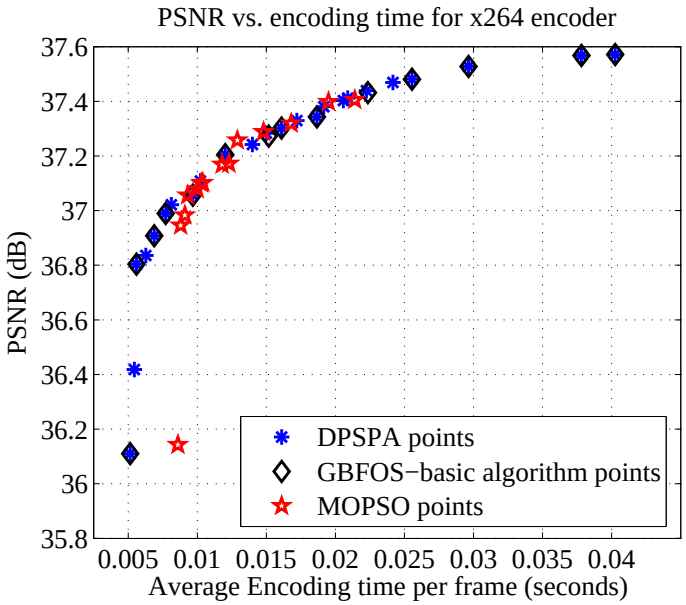


Figure 3.2: Distortion-complexity plot comparing GBFOS-basic algorithm, DPSPA and MOPSO for the ASL-1 data set at 30 kb/s.

quired per training video for searching the dominant points exhaustively is

$$N_{exhaustive} = \prod_{i=1}^{M} n_i, \tag{3.7}$$

while for the GBFOS-basic algorithm, it is

$$N_{GBFOS} = 1 - M + \sum_{i=1}^{M} n_i. \tag{3.8}$$

The difference between $N_{exhaustive}$ and N_{GBFOS} can be quite large, even when using few parameters. For example, using our four H.264 parameters, we obtain $N_{exhaustive} = 3360$ and $N_{GBFOS} = 33$, meaning the GBFOS-basic algorithm takes about 1% of the number of encodings required for exhaustively searching the dominant points. The DPSPA always takes the same number of encodings as the GBFOS-basic algorithm. The number of encodings required for the GBFOS-iterative algorithm depends upon the data set. In the worst case, it requires the same number of encodings as exhaustive search. However, from our experiments, using different bitrates and data sets, we found the maximum number of

encodings is 285, which is only 8.5% of the number of encodings required by exhaustive search.

The number of encodings required for MOPSO is the product of the number of iterations and the number of initial points, both of which are user-specified. We choose these MOPSO parameters so that MOPSO has the same number of encodings as the GBFOS-basic algorithm. We shall exclude discussion on both the DPSPA and MOPSO in the rest of this chapter, since the DPSPA has similar performance to the GBFOS-basic algorithm, and MOPSO generates parameter settings that are non-deterministic, and have low D-C spread and PSNR performance similar to the other algorithms.

We next compare the performance of parameter settings generated across an entire data set vs.single video in the data set. We illustrate the comparison for ASL-1 data set at 30 kb/s in Figure 3.3 for GBFOS-basic and GBFOS-iterative algorithms. We find the two set of points very close indicating that parameter settings obtained across training data set performs equally well on individual training videos.

3.3.2 Cross-validation results

To test the robustness of the GBFOS-basic algorithm against change in video content, the parameter settings obtained from a training set are applied to a test set. We use three (training set, test set) pairs, namely (ASL-1, ASL-2), (ASL-1,Standard) and (Standard, ASL-1) for three different bitrates: 30 kb/s, 150 kb/s and 300 kb/s. We compute the hyperarea ratio, and the maximum and mean PSNR difference on the test data and list them in Table 3.5. Figure 3.4 shows that GBFOS-basic and iterative points are close to the dominant points for the (ASL-1, Standard) case at 30 kb/s. For all (training, test) pairs, the GBFOS-basic and iterative algorithms have maximum and mean PSNR difference within 0.6 dB and 0.25 dB, respectively, showing that our algorithms are robust to changes in the data set.

Table 3.5: Cross-validation for different (training, test) data pairs on a 2.8 GHz PC for the (a) GBFOS-Basic algorithm, (b) and (c) MOPSO, and (d) the GBFOS-iterative algorithm.

Bitrate (kb/s)	(Standard, ASL-1)			(ASL-1, Standard)			(ASL-1, ASL-2)		
	PSNR diff (dB)			PSNR diff (dB)			PSNR diff (dB)		
	Max	Mean	HR	Max	Mean	HR	Max	Mean	HR
30	0.33	0.1	0.96	0.12	0.05	1.03	0.33	0.15	1.14
150	0.59	0.14	1.01	0.6	0.11	0.89	0.56	0.24	1.08
300	0.45	0.15	0.79	0.30	0.07	1.39	0.29	0.1	1.06

(a)

Bitrate (kb/s)	(Standard, ASL-1)			(ASL-1, Standard)		
	PSNR diff (dB)			PSNR diff (dB)		
	Max	Mean	HR	Max	Mean	HR
30	[0.3 0.4]	[0.1 0.1]	[0.6 0.7]	[0.3 0.4]	[0.1 0.1]	[0.7 0.7]
150	[0.5 0.6]	[0.2 0.2]	[0.7 0.8]	[0.3 0.3]	[0.1 0.1]	[0.4 0.4]
300	[0.5 0.5]	[0.1 0.1]	[0.5 0.6]	[0.3 0.3]	[0.1 0.1]	[0.8 0.9]

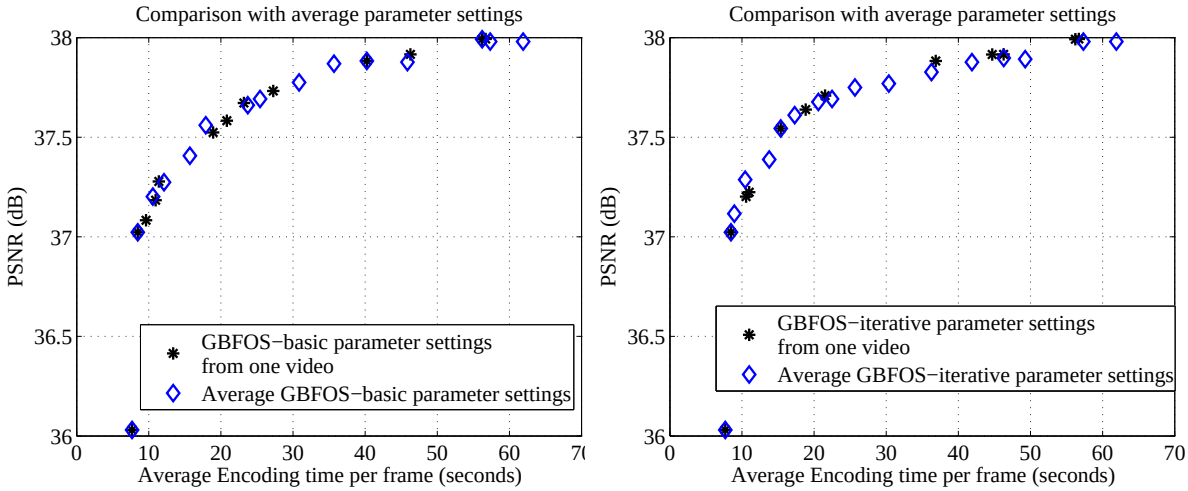
(b)

Bitrate (kb/s)	(ASL-1, ASL-2)		
	PSNR diff (dB)		
	Max	Mean	HR
30	[0.5 0.6]	[0.2 0.3]	[0.9 0.9]
150	[0.3 0.4]	[0.2 0.2]	[0.7 0.7]
300	[0.3 0.3]	[0.1 0.1]	[0.5 0.6]

(c)

Bitrate (kb/s)	(Standard, ASL-1)			(ASL-1, Standard)			(ASL-1, ASL-2)		
	PSNR diff (dB)			PSNR diff (dB)			PSNR diff (dB)		
	Max	Mean	HR	Max	Mean	HR	Max	Mean	HR
30	0.58	0.13	0.98	0.11	0.05	1.04	0.38	0.23	1.19
150	0.43	0.06	1.01	0.14	0.06	0.89	0.49	0.14	1.08
300	0.47	0.15	0.8	0.3	0.07	1.4	0.28	0.06	1.05

(d)



(a) (b)

Figure 3.3: Performance comparison between parameter settings obtained from a single video and across a data set for the (a) GBFOS-basic algorithm and (b) GBFOS-iterative algorithm, using ASL-1 data set at 30 kb/s.

3.3.3 Comparison with the default parameter setting

We compare the performance of the GBFOS-basic parameter settings with the x264 default parameter setting, which is $(1, 8, 5, 1)$ [6] on a 2.8 GHz PC. For evaluation, we pick the GBFOS parameter setting $\mathbf{p}$ that has PSNR close to the default parameter setting $\mathbf{q}$ and measure the speed gain and PSNR difference. Using the definition for PSNR() and $c()$ defined in Section 3.2, we define the speed gain as

$$S_g(\mathbf{p}, \mathbf{q}) = \frac{c(\mathbf{p}) - c(\mathbf{q})}{c(\mathbf{q})}, \tag{3.9}$$

and define the PSNR difference as

$$\Delta\text{PSNR}(\mathbf{p}, \mathbf{q}) = \text{PSNR}(\mathbf{p}) - \text{PSNR}(\mathbf{q}). \tag{3.10}$$

We use the three (training, test) pairs used previously and use five videos each from

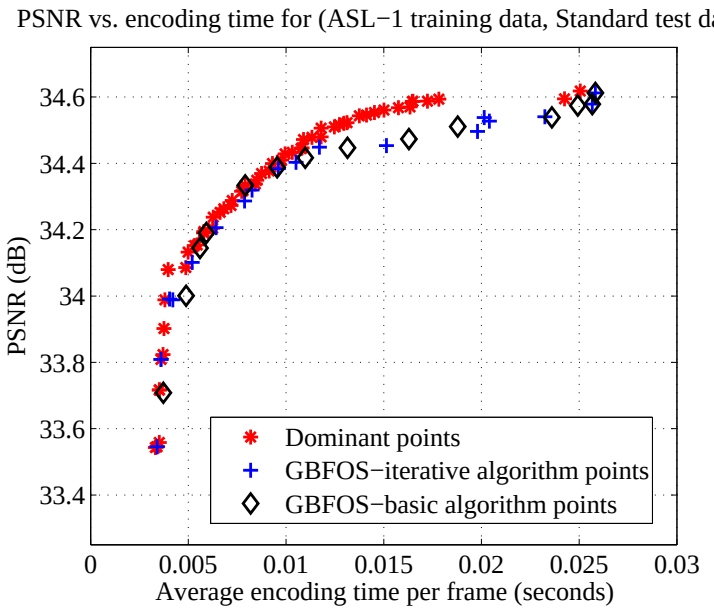


Figure 3.4: Cross-validation using ASL-1 training data and Standard test data at 30 kb/s. The dominant points are obtained from exhaustive search using Standard test data set.

the ASL-1 and ASL-2 data sets and six videos from the Standard data set. We list the results for the three bitrates and three (training, test) pairs in Table 3.6. Figure 3.5 shows that the GBFOS-basic parameter settings have better D-C tradeoff compared to the default parameter setting for the (ASL-1 training, ASL-2 test) pair at 30 kb/s. The GBFOS-basic and iterative algorithms result in a maximum speed gain of 37% and PSNR loss that is within 0.26 dB of the default setting.

3.3.4 Comparison between CLSA, DPSPA and the GBFOS-basic algorithm

In Chapter 2, we described the controlled local search algorithm (CLSA) and DPSPA, which generate additional parameter settings over the GBFOS-basic algorithm [72]. In this section, we compare the performance of the CLSA and DPSPA to the GBFOS-basic algorithm at 30 kb/s on a Linux computer with 2.8 GHz Intel CPU, and a Sprint PocketPC 6700 with a 416 MHz Intel CPU.

For our tests on the Linux machine, we use 10 videos from both the ASL-1 and ASL-2

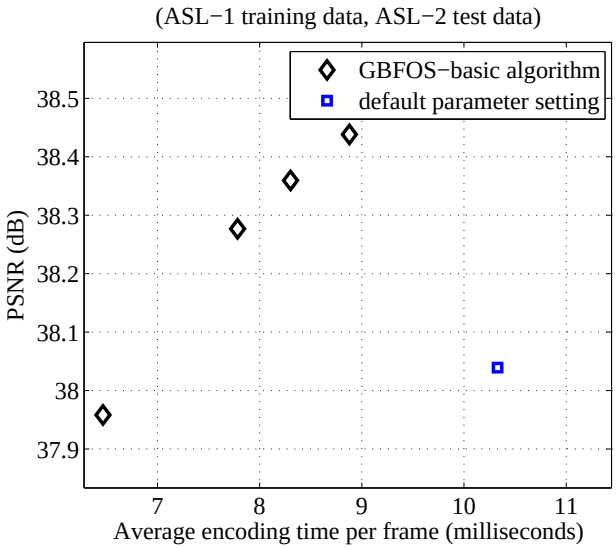


Figure 3.5: Performance of the GBFOS-basic algorithm and the x264 default parameter setting for the ASL-1 training data and ASL-2 test data at 30 kb/s on a 2.8 GHz PC.

data set at 30 fps. We perform cross-validation by randomly choosing six videos containing three of each resolution as the test data and the remaining 14 videos as our training data. We run both the CLSA and DPSPA to obtain extra parameter settings over the GBFOS-basic algorithm in the low and mid encode time regions indicated in Figure 3.6. Figure 3.6 illustrates GBFOS-basic points A and B, and additional points in between them generated by DPSPA and CLSA. (In Figure 3.6(b), point C performs less well due to the mismatch between training and test data.) From Figure 3.6, we find that CLSA often finds DPSPA points.

To evaluate the performance of our algorithm, we measure the maximum PSNR difference between each additional point generated and the lower PSNR/complexity GBFOS-basic parameter setting on test data. We use this difference since the encoder would need to operate at the lower GBFOS-basic parameter setting, if it could not find suitable parameter settings in the intermediate region. We would like the additional parameter settings generated to be spread over a given time region instead of clustering around a single parameter setting. Therefore, we measure the maximum encode time difference between adjacent

Table 3.6: Performance of (a) GBFOS-basic and (b) iterative parameter settings relative to the x264 default parameter setting on a PC for different (training, test) data.

Bitrate (kb/s)	(Standard, ASL-1)		(ASL-1, Standard)		(ASL-1, ASL-2)	
	ΔPSNR (dB)	Speed gain (%)	ΔPSNR (dB)	Speed gain (%)	ΔPSNR (dB)	Speed gain (%)
30	-0.26	27.8	-0.08	12.3	-0.08	37.4
150	0.09	17.2	-0.2	29.5	-0.18	19.7
300	-0.07	34.2	-0.17	5.7	-0.07	9.4

(a)

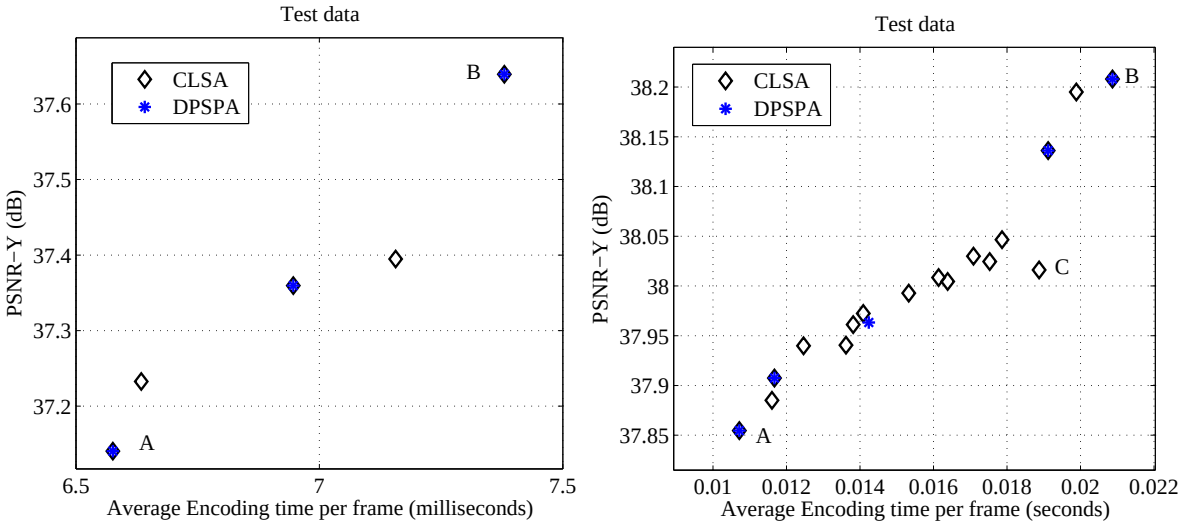
Bitrate (kb/s)	(Standard, ASL-1)		(ASL-1, Standard)		(ASL-1, ASL-2)	
	ΔPSNR (dB)	Speed gain (%)	ΔPSNR (dB)	Speed gain (%)	ΔPSNR (dB)	Speed gain (%)
30	-0.01	21	-0.08	12.3	-0.08	37.1
150	0.06	19.1	≈ 0	19.5	-0.1	22.4
300	-0.09	15.5	-0.06	31.8	-0.08	8.03

(b)

parameter settings generated by our algorithms on the test data.

Table 3.7 gives results for both CLSA and DPSPA on the Linux platform. Both algorithms yield slight improvement in maximum PSNR on the Linux platform. For the CLSA, we use the following start and end GBFOS-basic parameter settings for low encode time region {(1,3,1,2),(1,3,3,2)}, and for mid encode time region we use {(5,4,3,2),(5,4,7,2)}, respectively. The DPSPA takes the same number of encodings as the GBFOS-basic algorithm while the CLSA takes additional encodings. The CLSA takes more encodings, and yields more additional parameter settings.

For our tests on the cell phone, we use 10 QCIF videos from the ASL-1 data set. Five



(a) (b)

Figure 3.6: The parameter settings generated by CLSA and DPSPA for ASL test data on Linux platform. Points A and B are also generated by the GBFOS-basic algorithm. (a) Low encode time region and (b) mid encode time region. Point C performs less well due to the mismatch between training and test data.

of them were used in our earlier test on the Linux machine, but have a frame rate of 10 fps. The other five videos were recorded on a HTC TyTN II cell phone at 15 fps and have different signer and background compared to the other videos. We perform a leave-one-out cross-validation, by training on 9 videos and testing on one video. Since we are testing on the cell phone, we consider only three variable parameters each having fewer options. We use the `subme` options (0,1,2,3), where the option zero corresponds to integer pixel motion estimation. We use `part` options (0,1,2) which correspond to P16 $\times$ 16; P8 $\times$ 8; and I8 $\times$ 8, P8 $\times$ 8, respectively. We use all three trellis options. We use only one reference frame, no B-frames and diamond search.

Both the GBFOS-basic algorithm and the DPSPA take only 9 encodings per video on the training data. We apply the CLSA to the low and mid encode time regions by using {(1,1,0,1),(1,1,1,1)} and {(1,2,1,1),(1,2,3,1)} as the start and end GBFOS-basic parameter

Table 3.7: Results for the x264 encoder on Linux platform. Both the GBFOS-basic algorithm and DPSPA take only 33 encodings.

	Low encode time		Mid encode time	
	CLSA	DPSPA	CLSA	DPSPA
Number of encodings per video	17+33=50	33	123+33 = 156	33
Number of additional parameter settings generated	3	1	15	3
Maximum PSNR gain (dB)	0.25	0.22	0.34	0.28
Maximum encode time difference (seconds)	3×10^{-4}	4×10^{-4}	13×10^{-4}	49×10^{-4}

Table 3.8: Results for the x264 encoder on PocketPC platform. Both the GBFOS-basic algorithm and DPSPA take only 9 encodings.

	Low encode time		Mid encode time	
	CLSA	DPSPA	CLSA	DPSPA
Number of encodings per video	8+9=17	9	6+9=15	9
Number of additional parameter settings generated	1	0	3	1
Maximum PSNR gain (dB)	0.71	0	0.664	0.43
Average maximum encode time difference (seconds)	45×10^{-4}	59×10^{-4}	49×10^{-4}	73×10^{-4}

setting pairs, respectively. The results for low and mid encode time regions are tabulated in Table 3.8. The CLSA gives a maximum PSNR gain of 0.71 dB on the PocketPC platform. The CLSA again takes more encodings compared to the DPSPA, but also yields parameter settings with a higher PSNR gain. Since the DPSPA does not find any parameter settings in the low encode time region, the maximum encode time difference is the time difference between the two adjacent GBFOS-basic parameter settings. The CLSA results in fewer parameter settings for the cell phone data when compared to the Linux data, since we consider fewer parameters with fewer options.

3.4 ROI-based Motion Search for ASL Videos

In this section, we propose region of interest(ROI)-based motion search method for ASL videos for the x264 encoder and use it as an additional parameter to test our algorithms. Our ROI-based motion search improves the speed of the encoder without loss in PSNR.

Each frame in an ASL video sequence can be classified into three distinct regions, namely head, torso and background as shown in Figure 3.7. In ASL videos, the head has limited motions while the hands have larger motions that often occur in the torso region. Considering this prior knowledge, we can use different motion search methods for the different regions. We hand-label the macroblocks that belong to the head and torso regions of our data set and use the GBFOS-basic and iterative algorithms to find the ROI-based motion search methods that result in good distortion-complexity performance.

Motion estimation (ME) is the most time-consuming part of the H.264 encoder as it uses multiple prediction modes and reference frames. We shall consider the following four x264 motion search methods, which are listed in increasing complexity and have been described in Section 1.3: DIA, HEX, UMH and ESA.

In our tests, we consider 8 different region-of-interest (ROI) based motion search methods: UMH for all regions, while the other 7 methods use DIA for the background and use either DIA, HEX or UMH for the head and torso regions. In [73] and [74], it was shown that fluent signers look at the face 95% of the time when watching ASL videos. This leads us to choose a motion search method for the torso region of similar or lower complexity when compared to the motion search of the face region, thereby ensuring better quality for the face region. Therefore, we consider the following 7 (head, torso) motion search pairs: (DIA, DIA), (HEX, DIA), (HEX, HEX), (UMH, DIA), (UMH, HEX), (HEX, UMH) and (UMH, UMH). We do not consider the combination of DIA search for the head and UMH or HEX for the torso region.

Our data set includes five ASL video clips belonging to the ASL-2 data set and they are encoded at 30 fps and 30 kb/s on the same computer as in Section 3.3. The head region consists of 5×8 macroblocks (height×width) and the torso consists of 6×12 macroblocks (See Figure 3.7). We conduct an exhaustive search using the same variable parameters

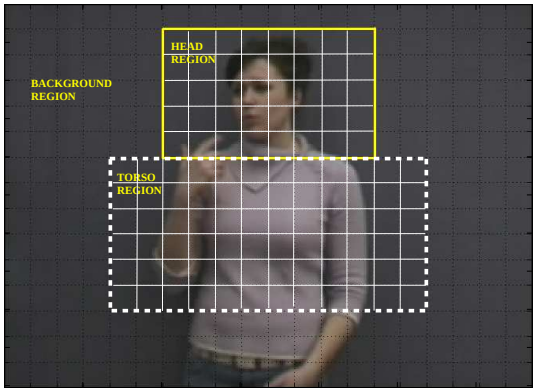


Figure 3.7: Different regions of an ASL video frame.

from Section 3.3 together with 8 ROI-based motion search options mentioned above. We plot the distortion-complexity convex hull points in Figure 3.8. We find that using ROI-based motion search improves the speed of the encoder up to 11.3% over using UMH for the entire frame. At faster encoding speeds, the ROI-based motion search chooses different search methods for different regions, but at slower speeds it chooses all-UMH and the two sets of dominant points overlap as shown in Figure 3.8. This makes sense as UMH provides higher PSNRs when longer encoding time is available.

Table 3.9: Results for the GBFOS-basic and iterative algorithms when using ROI-based motion search as one of the x264 encoder parameters.

	GBFOS-basic algorithm	GBFOS-iterative algorithm
Max PSNR diff (dB)	0.05	0.05
Mean PSNR diff (dB)	0.02	0.01
# of encodings	40	273
Hyperarea ratio	0.98	0.98
# of points	10	13

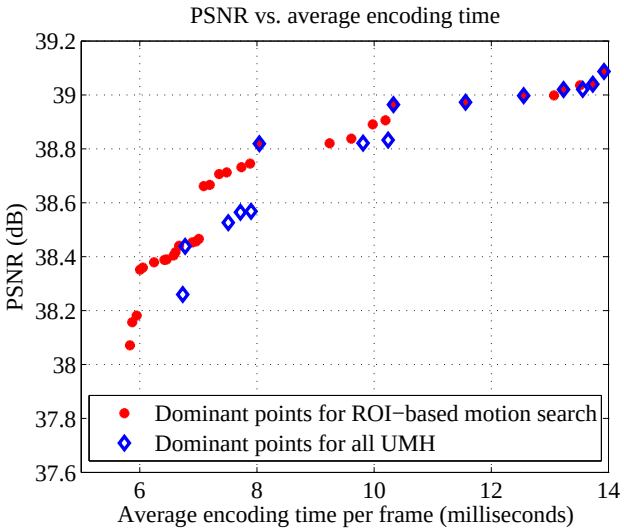


Figure 3.8: PSNR vs. average encoding time per frame for the dominant points with and without ROI-based motion search at 30 kb/s.

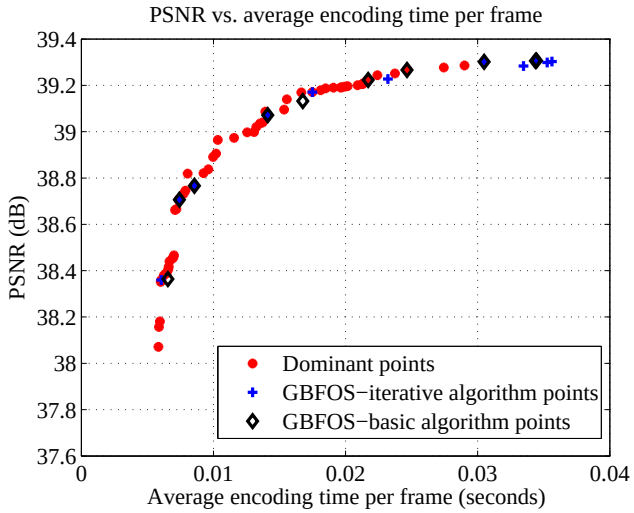


Figure 3.9: PSNR vs. average encoding time per frame for ROI-based motion search at 30 kb/s.

To find the dominant points using the five variable encoder parameters mentioned above, we require $16 \times 10 \times 7 \times 3 \times 8 = 26,880$ encodings per video sequence, using Equation

(3.7). However, using the GBFOS-basic and GBFOS-iterative algorithms, we require 40 and 233 encodings only, i.e., 0.15% and 1% of the encodings required by an exhaustive search, respectively. Figure 3.9 shows that both the GBFOS-basic and iterative points lie close to the dominant points. Table 3.9 shows that both our algorithms have a maximum PSNR difference of 0.05 dB, and also have very similar performance for other metrics. This indicates that when using the ROI-based motion search with other x264 parameters, the low complexity GBFOS-basic algorithm can be used to find the parameter settings.

3.5 Results for the H.263+ Encoder

In this section, we use both our GBFOS algorithms for choosing H.263+ encoder parameters. Many new features and modes were introduced in the H.263 standard [75] to improve its compression efficiency, resulting in the H.263+ standard [76]. H.263+ allows different source formats and provides scalability that helps improve error resilience. This standard has 12 new modes that include unrestricted motion vectors, advanced intra coding, deblocking filter, slice structure, improved PB-frames, alternative inter variable length coding (VLC), and modified quantization [24]. These modes may have two or more options. Similar to the H.264 encoder, finding the optimal parameter settings for the H.263+ encoder using exhaustive search is time consuming.

We have considered the H.263+ encoder since it is fairly different from the x264 encoder. We use the reference source code for H.263+ from [77] and enable the use of deblocking filter in our tests. We consider the following variable parameters, with their respective options given in parentheses: advanced prediction mode (on, off); advanced intra coding mode (on, off); integer pel search window (0,1,. . . ,15); unrestricted motion vector mode (H.263, H.263+, and H.263+ unlimited range); and quantization methods (arithmetic coding, and modified quantization mode with and without the use of alternate inter VLC mode).

In the advanced prediction mode, each of the four 8×8 luma subblocks belonging to a macroblock can have a motion vector. The advanced intra coding mode allows interblock prediction from neighboring intrablocks. It uses modified quantization and a separate VLC table for intrablocks. In unrestricted motion vector mode, the motion vectors are allowed to point outside the picture. However, using the H.263 option the motion vectors are

restricted to reference within a picture area. In the alternate inter VLC mode, the VLC table used for intrablocks can be used for coding interblocks as well. The modified quantization mode allows the quantizer value to be changed at the macroblock level, and allows the use of smaller quantizer step size for chrominance samples, and finally extends the range of representable quantized DCT coefficients [24].

Our test set includes 10 ASL videos clips of 600 frames each from the ASL-1 data set used in Section 3.3. We encode the videos at 30 kb/s and 30 fps and obtain our results using the same computer used in Section 3.3. For generating the marginal plots for each parameter, we consider the following parameter options to result in least MSE: advanced prediction mode = on; advanced intra coding mode = on; integer pel search window = 15; unrestricted motion vector mode = H.263+ unlimited range; and quantization method = arithmetic coding.

Figure 3.10 shows that points belonging to the GBFOS algorithms are close to the dominant points from exhaustive search. The number of encodings required for generating the dominant points, GBFOS-iterative, and GBFOS-basic points are 1536, 52 and 23, respectively. That is, the GBFOS-basic and iterative algorithms take only 1.5% and 3.4% of the number of encodings required by exhaustive search, respectively. From Table 3.10, we find that both our algorithms have similar mean and maximum PSNR difference, hyperarea ratio, and number of points. Therefore, when encoding ASL videos at a very low bitrate (30 kb/s) using the H.263+ encoder, we can just use the GBFOS-basic parameter settings.

3.6 Summary

In this chapter, we compared the performance of the GBFOS-basic and iterative algorithms, DPSPA, CLSA, MOPSO, and exhaustive search, using several performance metrics. The GBFOS-basic and iterative algorithms take only 1% and 8.5% of the time required for exhaustive search and result in a maximum and mean PSNR difference of 0.6 dB and 0.25 dB, respectively. The MOPSO generates parameter settings that are non-deterministic having low D-C spread. The DPSPA takes the same number of encodings as the GBFOS-basic algorithm but generates more parameter settings. Given two GBFOS-basic parameter settings, the CLSA generates additional parameter settings between them using additional

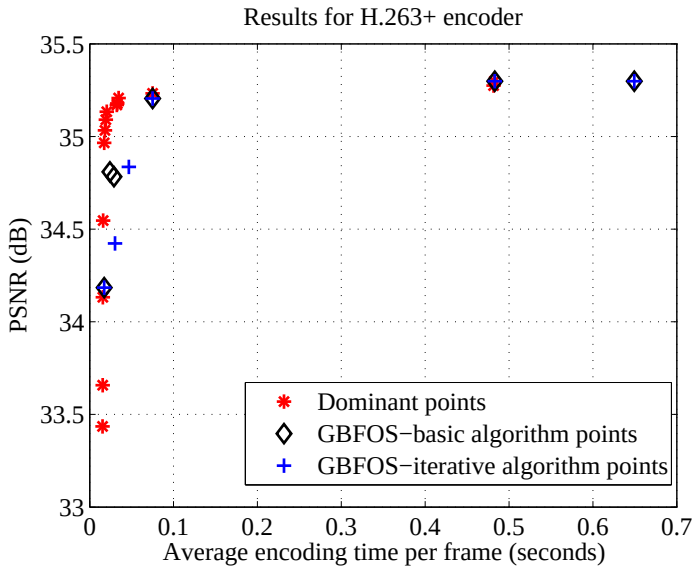


Figure 3.10: Distortion-complexity plot comparing GBFOS-basic and iterative points with dominant points from the exhaustive search for the H.263+ encoder when using ASL-1 data set at 30 kb/s.

Table 3.10: Results for the H.263+ encoder when using ASL-1 data set at 30 kb/s and 30 fps. The DPSPA generates the same points as the GBFOS-basic algorithm.

	GBFOS-basic algorithm	GBFOS-iterative algorithm
Max PSNR diff (dB)	0.78	0.78
Mean PSNR diff (dB)	0.25	0.33
# of encodings	23	52
Hyperarea ratio	0.99	≈ 1
# of points	6	7

encodings. The overall PSNR performances of the GBFOS-basic and GBFOS-iterative algorithms, DPSPA, and MOPSO are very similar. Therefore, we would recommend either the use of GBFOS-basic algorithm or DPSPA for generating parameter settings, since they take the fewest encodings. (We did not consider CLSA in this comparison, since it requires two GBFOS-basic parameter settings as its input.)

The GBFOS-basic and iterative parameter settings improve encoding speed by up to 37% over the x264 default parameter setting on a PC, with negligible loss in PSNR. The use of ROI-based motion search parameter improves the encoding speed by 11.3% over using the UMH search, with no loss in PSNR. Finally, we demonstrate the generality of our algorithms by applying them to the H.263+ video encoder, to select parameter settings that trade off well between PSNR and encoding speed. In the following chapter, we will analyze the performance of our algorithms on cell phones for encoding ASL videos.

Chapter 4

AMERICAN SIGN LANGUAGE VIDEO COMPRESSION ON CELL PHONES

4.1 Introduction

Current mobile devices such as cell phones, PDAs, and PocketPCs are equipped with video cameras and codecs that can ultimately be used for real-time video communication of ASL [78]. The compression efficiency of H.264 encoder makes it suitable for use in such mobile devices that operate at low bandwidth. Also, many cell phones have hardware H.264 decoders, which allow fast decoding. Since cell phones typically have low processing speed (on the order of hundreds of MHz, although the new HTC EVO phone has a 1 GHz processor), it is essential to use encoder parameter settings that optimize the encoder for both frame rate (speed) and quality.

An exhaustive search for finding distortion-complexity optimized parameter settings on a cell phone is infeasible due to its slow processor speed. Continuing with our example in Section 2.2, for a 416 MHz PocketPC, an exhaustive search with four variable parameters would approximately take 25 days $\times \frac{2.8 \text{ GHz}}{416 \text{ MHz}} \approx 5$ months! To speedup the process of finding parameter settings, we propose to use the GBFOS-basic algorithm which takes ≈ 5 months$\times$ $1\% = 1.5$ days.

In this chapter, we apply the GBFOS-basic algorithm to the x264 encoder for encoding ASL videos on a cell phone. We compare the performance of the GBFOS-basic algorithm with the estimated convex hull on the PocketPC. We analyze the cross-platform performance of the GBFOS-basic parameter settings. We show that the GBFOS-basic parameter settings are robust to different ASL signers used in training and test videos. Finally, we compare the performance of the GBFOS-basic parameter settings to the x264 default parameter setting on a cell phone.

4.2 Comparison with the estimated convex hull

In this section, we compare the performance of the GBFOS-basic parameter settings to an estimate of the convex hull on the Sprint PocketPC with a 416 MHz Intel processor. To save time, we estimate the convex hull parameter settings by running all possible combinations of parameters selected by the GBFOS-basic algorithm. We choose only three variable parameters (`ref`, `subme` and `part`), and use `trellis` $= 3$ for quantization method. We use a training set consisting of five 320×240 ASL videos at 10 fps, containing 100 frames each encoded at 25 kb/s. The five ASL videos belong to the ASL-2 data set described in Section 3.3 with the frame rate sub-sampled from 30 to 10 fps.

In our tests, the GBFOS-basic algorithm results in parameter settings that have four different options for `subme`, three options for `part` and 9 options for `ref`. We use all possible combinations of these parameters ($4 \times 3 \times 9 = 108$ encodings) to estimate the convex hull. However, the GBFOS-basic algorithm here takes 31 encodings only.

Figure 4.1 shows that the GBFOS-basic algorithm performs well and overlaps with most points on the estimated convex hull. Both the estimated convex hull and the GBFOS-basic algorithm require fewer encodings than an exhaustive search which requires $7 \times 16 \times 10 = 1120$ encodings.

4.3 Cross platform performance

In this section, we investigate the use of PC-based GBFOS-basic parameter settings on the cell phone and compare it with cell phone-based GBFOS-basic parameter settings. In our experiments, we use two cell phone platforms, a Sprint PocketPC used in the previous section, and a HTC TyTN-II cell phone which has a 400 MHz ARM processor. For the Sprint PocketPC, we use the videos described in Section 4.2 and use the same four x264 encoder parameters described in Section 3.3.

For the HTC cell phone, we use QCIF ASL videos at 15 fps recorded on the cell phone itself. We use five training videos and one test video at 30 kb/s. We trade off `subme` $(1, \ldots, 6)$; `ref` $(1, \ldots, 8)$; and `part` $(1, \ldots, 8)$. Figures 4.2 (a) and (b) show that the GBFOS-basic parameter settings obtained from a 2.8 GHz Linux machine perform very close to

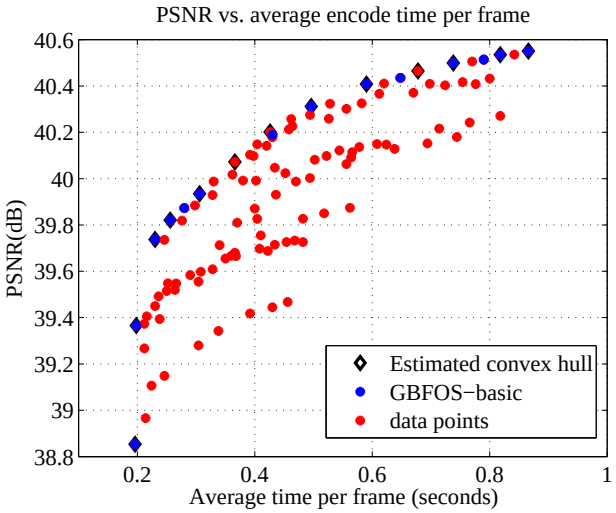


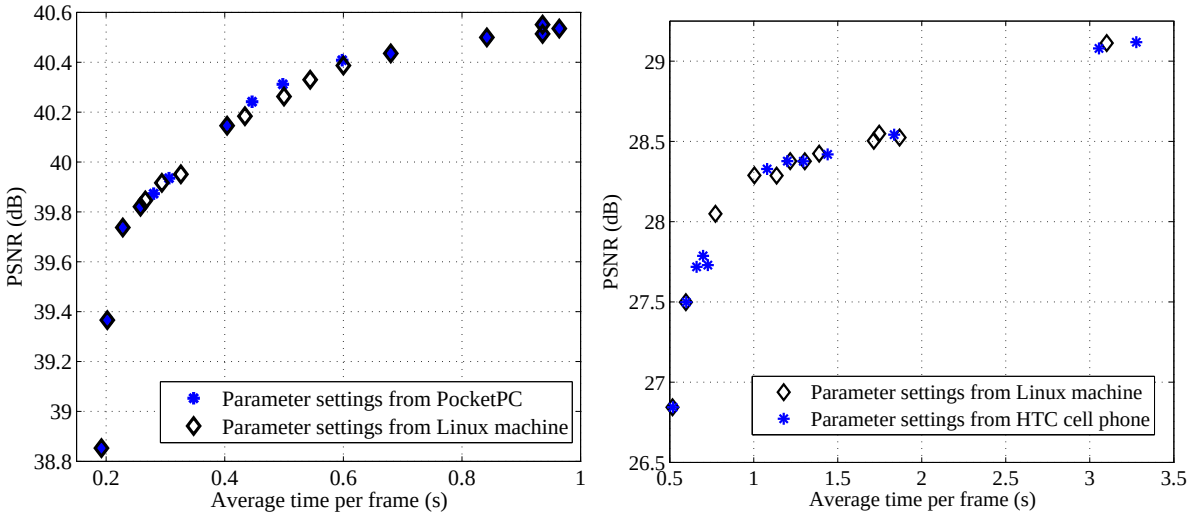
Figure 4.1: PSNR vs. average encoding time per frame on a Pocket PC for ASL-2 data set at 25 kb/s.

the cell phone parameter settings, indicating that the training time for the GBFOS-basic algorithm can be further sped up by running it on a Linux machine. Continuing with our previous example, running the GBFOS-basic algorithm on the Linux machine would take only 6 hours instead of 1.5 days on a cell phone.

4.4 Performance across different ASL signers

In this section, we investigate whether the GBFOS-basic parameter settings are robust to mismatch in signers in test and training videos. In our experiments, we use the ASL-1 data set described in Section 3.3 having Jessica as the signer, and two ASL videos with a different signer (Gina) at 15 fps. Both user category videos are QCIF resolution and we encode them at 30 kb/s. We run our tests on a 2.8 GHz PC. We train the GBFOS-basic parameter settings using the Jessica video and apply them to the Gina test video. We obtain the GBFOS-basic parameter settings from a training Gina video and apply it to the test Gina video used before.

Figure 4.3 shows points corresponding to both sets of parameter settings on the Gina test video. We find that the maximum PSNR difference is 0.2 dB, which suggests that the



(a) (b)

Figure 4.2: Cross platform performance of GBFOS-basic parameter settings. Comparing performance of Linux trained parameter settings vs. parameter settings obtained from (a) Sprint PocketPC and (b) HTC Ty TN-II cell phone. The two plots are obtained using different test videos and encoder parameters.

GBFOS-basic parameter settings are not overtrained for a specific signer in the training video, and can be used across different signers without significant loss in PSNR.

4.5 Comparison with the x264 default parameter setting

In this section, we compare the performance of the GBFOS-basic and GBFOS-iterative parameter settings with the x264 default parameter setting on a HTC TyTN-II cell phone. We use the MobileASL software described in Section 1.4.

We use ten 96×80 resolution videos from the ASL-1 data set at 10 fps. We choose five videos for training and use the rest for testing. To further speed up the training time for finding the parameter settings on the cell phone we run the GBFOS-basic and iterative algorithms on a 2.8 GHz PC. We select a GBFOS-basic parameter setting and a GBFOS-

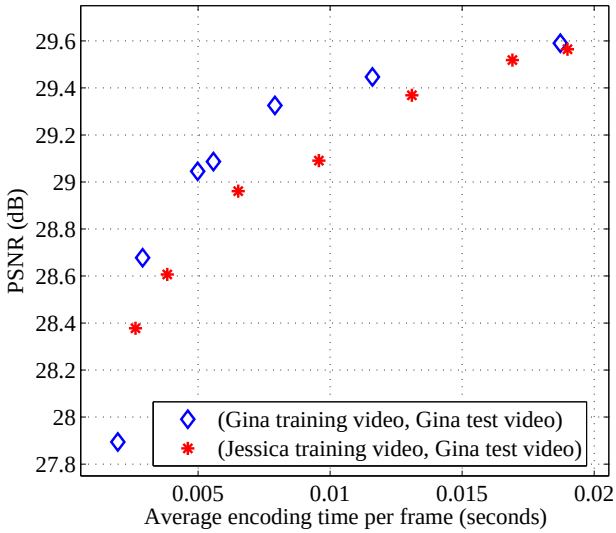


Figure 4.3: Performance of GBFOS-basic parameter settings on videos belonging to different signers on a PC.

iterative parameter setting that have similar PSNR performance as the default parameter setting on the PC, and test them on the cell phone to compare their encoding speed. For each test video, parameter setting, and bitrate we encode, transmit, receive, decode and display the video in real-time on the cell phone for sixty seconds, and measure the average encoding frame rate.

The default parameter setting results in 7 fps at 30 kb/s, and 5 fps at both 150 kb/s and 300 kb/s, while the GBFOS-basic parameter setting results in frame rates close to 10 fps at all three bitrates. For lower bitrates, the GBFOS-iterative parameter setting provides close to 10 fps. At 300 kb/s, the GBFOS-iterative algorithm chooses parameter setting having higher complexity, thereby resulting in a frame rate of 7.5 fps. Because we obtained our parameter settings from a PC, it is likely that the GBFOS-iterative algorithm chose the higher complexity parameter setting at 300 kb/s, since it does not affect the frame rate on a high speed PC. Studies have shown that a minimum of 10 fps is required for maintaining ASL intelligibility [79,80] indicating that the GBFOS-basic and iterative parameter settings

often provide frame rates necessary for ASL intelligibility.

In Table 4.1, we list the the average frame rates achieved when using the default, GBFOS-basic and GBFOS-iterative parameter settings on a HTC TyTN II cell phone. The GBFOS-basic and iterative algorithms improve the encoder speed by up to 94.1% and 89.3% which corresponds to 4.9 and 4.7 fps increase in frame rate over the default parameter setting, respectively. Since the GBFOS-basic and GBFOS-iterative parameter settings have similar performance, we use the low complexity GBFOS-basic algorithm to train for parameter settings.

The frame rates between the default parameter setting and the GBFOS-basic and GBFOS-iterative parameter settings differ. This makes a PSNR comparison difficult on a cell phone. Therefore, we instead compare the PSNR on a PC to ensure equal frame rates. We found that the PSNR differences between the default and our parameter settings were within 0.3 dB, indicating comparable quality.

4.6 Summary

In this chapter, we showed that the GBFOS-basic algorithm performs close to the estimated convex hull parameter settings on the cell phone. We showed that the GBFOS-basic parameter settings are not overtrained for a specific training platform. Therefore, we can speed up training time by running the GBFOS-basic algorithm on a PC and use the parameter settings on a cell phone. The GBFOS-basic parameter settings have been shown to be robust to mismatch in signers in training and test videos. Finally, on a cell phone the GBFOS-basic parameter setting nearly doubles the frame rate over the x264 default parameter setting with negligible loss in PSNR.

Table 4.1: Comparing the frame rates of the x264 default parameter setting with (a) GBFOS-basic parameter setting and (b) GBFOS-iterative parameter setting, on a HTC TyTN-II cell phone when using ASL-1 training and test data. The GBFOS-basic and iterative parameter settings used in this experiment are listed below.

Bitrate (kb/s)	Default (fps)	GBFOS-basic (fps)	GBFOS-basic parameter setting
30	7.2	9.7	(1, 2, 3, 2)
150	5.2	9.5	(1, 5, 1, 2)
300	5.2	10.2	(1, 1, 1, 2)

(a)

Bitrate (kb/s)	Default (fps)	GBFOS-iterative (fps)	GBFOS-iterative parameter setting
30	7.2	10	(1, 1, 3, 2)
150	5.2	10	(1, 1, 1, 2)
300	5.2	7.6	(1, 6, 3, 1)

(b)

Chapter 5

DISTORTION-COMPLEXITY OPTIMIZATION OF THE CORNELL ASL INTELLIGIBILITY-OPTIMIZED VIDEO ENCODER

5.1 Introduction

Region-of-interest (ROI) based video encoders are traditionally used to achieve bitrate savings by allocating more bits to the most relevant regions and fewer bits to other regions. They could also be used to reduce the complexity of encoding by allocating more complexity to the important regions. In this work, we shall combine these two concepts to achieve joint rate-distortion-complexity optimization.

In related work, the perceptual quality of videoconferencing was improved by reducing distortions in the user's face [81, 82]. ROI-based video compression can be extended to American Sign Language (ASL) video. For ASL video, distortion in face and hand regions affects intelligibility. This is supported by both the linguistic structure of sign language [83] (e.g. how information is conveyed) and by eye-tracking experiments [84]. Because of this unique structure, several specialized algorithms have been proposed for encoding sign language videos [84–86].

In [86], Ciaramello and Hemami developed an ASL optimized video encoder using an objective measure of intelligibility incorporated into an H.264 rate-distortion (R-D) optimization algorithm. A performance bound for the system is obtained using the Viterbi algorithm to search over all possible quantization parameters and encoding modes. For fixed levels of intelligibility, the bitrate is reduced by 60% over an R-D optimization algorithm which measures distortion as MSE. The goal of our rate-distortion-complexity optimization is to achieve as much of this R-D gain while maintaining a computational complexity appropriate for mobile devices with low processing power.

In this chapter, an ROI-optimized video encoder is presented that includes three new ROI-based parameters that allow variations in encoding complexity per-macroblock based

on its relative importance. We use the DPSPA described in Section 2.9 to search the space of encoder parameter setting, to find parameter settings that yield improvement in complexity (encoding speed), with only small decreases in intelligibility. This approach improves the encoder speed on the PC and cell phone platforms by up to 54.4% and 60.8%, respectively, with negligible loss in intelligibility when compared to the x264 default parameter setting.

The rate-distortion-complexity (R-D-C) optimization performed in this work is evaluated specifically for ASL video, but the same procedure can be applied to any class of ROI video (e.g. videoconferencing), subject to the availability of a distortion measure that reflects the relative importance of each region. In this work, the face, hands and torso are the ROI and the background is the non-ROI.

5.2 Objective Metric for ASL intelligibility

In this section, we describe the objective intelligibility metric for ASL videos developed by Ciaramello and Hemami [87]. This metric is used as a distortion measure instead of MSE. The following four components form a part of the intelligibility distortion metric:

1. Spatial distortion (d'_k).

2. Temporal increase in spatial distortion (tv_k).

3. Coding distortion in the background macroblocks (D_{NewBG}).

4. Intelligibility loss due to low frame rate($f(r_f)$).

The procedure for computing the distortion measure is described below.

Let $Y(i,j,n)$ and $\hat{Y}(i,j,n)$ correspond to the luma pixel values of the original and processed videos at spatial location (i, j) in frame n. $\bar{Y}(n)$ is the mean luma pixel value in frame n. The distortion between original and processed video at pixel location i,j and frame number n is given by

$$e_c(i,j,n) = \frac{Y(i,j,n) - \hat{Y}(i,j,n)}{\bar{Y}(n)}. \tag{5.1}$$

Let k denote the head, hands, and torso regions in a frame, and N_k denote the number of pixels in each region k. Let N be the total number of frames in the video. The average spatial distortion in each region k is computed by

$$d_k(n) = \frac{1}{N_k} \sum\nolimits_{i,j \in \text{Region } k} e_c(i,j,n)^2. \tag{5.2}$$

If distortion occurs in a small set of frames, the video can still be intelligible. To eliminate the effect of short duration distortion, a median filter of odd frame length γ is applied to $d_k(n)$ as follows

$$d'_k(n) = median\left(d_k(n - \frac{\gamma - 1}{2}), \ldots, d_k(n + \frac{\gamma - 1}{2})\right). \tag{5.3}$$

To penalize for abrupt and sustained increases in distortion, the average of the largest 5% of the positive gradients of $d'_k(n)$ is computed for region k as follows

$$tv_k = avg_{5\%}(\Delta d'_k(n)), \tag{5.4}$$

where $\Delta d'_k(n)$ is the positive gradients of $d'_k(n)$.

To reduce the complexity of encoding background macroblocks, they can be skipped. However, if a background macroblock contains a face or hand in the previous frame, and was skipped in the current frame, portions of the hand or face will be noticeable in the decoded background macroblock. This artifact can be very distracting and possibly reduce the intelligibility. To include this distortion in the objective metric, an average distortion is computed specifically for such background macroblocks ($NewBG$) as

$$D_{NewBG} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{N_k} \sum\nolimits_{i,j \in NewBG} e_c(i,j,n)^2. \tag{5.5}$$

A lower frame rate can affect intelligibility and this loss is modeled as a sigmoid function given by

$$f(r_f) = a_1(1 - e^{-e^{a_2 - a_3 r_f}}), \tag{5.6}$$

where a_1, a_2, and a_3 control lower asymptote, convergence locations and growth rate, respectively, and r_f is the instantaneous frame rate. The constants $a_1 = 10$, $a_2 = 0$, and $a_3 = 0.5$.

The final objective intelligibility measure is given by

$$D_I = \log_{10}\left(1 + D_{NewBG} + \sum_k \frac{\alpha_k}{N} \sum_{n=1}^{N} d'_k(n) + \beta_k tv_k\right) + f(r_f), \tag{5.7}$$

The values of α_k, β_k, and the parameters in $f(r_f)$ are optimized using ground truth subjective intelligibility ratings of a collection of processed ASL videos. The weights were determined to be $\alpha_{face} = 1.6$, $\alpha_{hands} = 0.5$ and $\alpha_{torso} = 0.1$. A higher weight is given to the face since distortion in the signer's face results in lower intelligibility compared to the same distortion in the signer's torso.

The objective metric developed prior to the metric given in Equation 5.7 used only weighted mean squared error computed at different regions of an ASL video, and is given by

$$D_I = \log_{10}\left(1 + \sum_k \frac{\alpha_k}{N} \sum_{n=1}^{N} d_k(n)\right). \tag{5.8}$$

The authors of this metric have observed that their new metric in Equation 5.7 can be approximated by the sum of Equation 5.8, and $f(r_f)$ given by Equation 5.6, for videos within a data set.

5.3 ASL Intelligibility Optimized Video Encoder

Ciaramello and Hemami's ASL intelligibility optimized video encoder is implemented within the x264 encoder. Each frame of the input sequence is segmented into the signer's face, hands, torso, and background, using color-based skin detection and morphological processing. The ASL-tuned distortion measure in Equation (5.7) is incorporated into a rate-distortion (R-D) optimization procedure similar to that of [88].

For a given Lagrangian λ, the parameter p that includes motion vector, partition size and quantization step size (QP) is chosen such that it minimizes the joint R-D cost of a macroblock X given by

$$J(X, p) = D(X, p) + \lambda R(X, p). \tag{5.9}$$

A consequence of using the distortion measure D_I is that more rate is inherently allocated to the important regions (i.e., face, hands, and torso). The work presented in [86] identified

a functional relationship between λ and the resulting optimal QPs. Therefore, a λ value is mapped to QP corresponding to each region. The motion vector and partition size for each macroblock are selected according to the minimum R-D cost.

The ASL video encoder has four different rate control options:

1. Single pass constant λ method. In this approach, a user-specified λ is applied to all frames. A constant λ corresponds to constant QP, which does not result in constant average bitrate. (Two different videos encoded using the same λ result in different bitrates.)

2. Single pass constant average bitrate method. In this approach, the user specifies the average bitrate. To achieve the target bitrate, the encoder updates λ in successive frames as follows:
$$\lambda(n+1) = \lambda(n) - \left(\frac{R_{target}}{R_{actual}} - 1\right), \tag{5.10}$$
where R_{target} and R_{actual} are the target and actual bits for frame n, respectively.

3. Multi-pass constant average bitrate method. Here a bisection search approach is used to find a λ for the entire video that results in the user-specified bitrate. However, it can take multiple encodings to find the suitable λ.

4. Multi-pass constant intelligibility method. It is similar to the previous approach, except that a search is performed for λ that results in the user-specified intelligibility.

We perform rate-distortion-complexity optimization using the first and the second rate control methods, since they are single pass methods.

5.4 ROI-based Complexity Allocation Parameters

Implementing the ASL intelligibility optimized video encoder as an extension of x264 allows for the use of all the encoding parameters available to x264, the selection of which provides a tradeoff between encoding complexity and R-D performance. Specifically, four encoding parameters available in the x264 are varied to achieve different R-D-C operating points:

sub-pixel motion estimation (`subme`); reference frames (`ref`); partition size (`part`); and entropy coding and quantization (`trellis`). These parameters were described in Section 1.3. For `trellis`, a fourth additional option that uses uniform quantization with CABAC is included.

Three additional encoding parameters are added to the x264 encoder to allow the encoding complexity to vary on a per-block basis, depending on whether the block belongs to the ROI or not. In H.264, as many as 12-15 different partitions need to be analyzed for a given macroblock. Our first parameter, `nonROI-part`, restricts the partitions used by the encoder for the background blocks. Since the distortion in background macroblocks have less effect on overall intelligibility, background macroblocks can be encoded with very little rate (and consequently, very high distortion). Motivated by this, the encoder is modified to have two sets of available partition types, one for the ROI blocks and other for the non-ROI blocks. For ease of integration into the pre-existing encoder structures, the `nonROI-part` has the same 8 options as `part`. (We consider only 8 `part` options instead of 10 used in Chapter 3.) This parameter allows the selection of coarser partitions for background macroblocks and finer partitions for the ROI blocks.

The second parameter, `ROI-subme`, has the same 7 options as the `subme` parameter and is applied to the ROI, while the regular `subme` option is applied to the background or non-ROI region. In addition to varying the complexity of sub-pixel motion estimation, the `subme` also varies the accuracy and complexity for R-D cost computation. The highest `subme` option computes the actual R-D cost by encoding and decoding a macroblock, while the lowest option only estimates the R-D cost from the coded macroblock. The `ROI-subme` together with `subme` allow the encoder to use the fast R-D cost estimate on non-ROI blocks, while computing the accurate R-D cost and using high complexity sub-pixel motion estimation for the ROI blocks.

The third ROI parameter addresses the complexity of the motion search. In motion-compensated video coding, motion search requires a significant portion of the total encoding time. To speed up the motion search, a ROI-based motion search parameter `ROI-MS` is included that specifies a potentially different motion search method for the ROI and non-ROI macroblocks. The `ROI-MS` uses the following three fast motion search methods provided

by x264 in the order of increasing complexity: diamond (DIA), hexagon (HEX) and uneven multihexagon search (UMH) [6]. Details of `ROI-MS` are given in Section 3.4.

5.5 Joint Rate-Intelligibility-Complexity Optimization

The set of encoding options discussed in Section 5.4 determines the achievable bitrate, distortion, and complexity. The x264 default parameter setting is the vector (`subme` = 5, `part` = 8, `ref` = 1, `trellis`= 1, `nonROI-part`=`part`, `ROI-subme`=`subme`, `ROI-MS`=0). The ROI-based parameters have been disabled, by setting `nonROI-part` and `ROI-subme` to `part` and `subme`, respectively, and `ROI-MS` to all HEX. The default parameter setting uses high complexity sub-pixel motion estimation; all possible macroblock partitions; one reference frame; and the use of the context adaptive arithmetic coder (CABAC) with uniform quantization.

An ideal video encoder will select the parameter setting which results in a compressed video that meets the target rate and complexity constraints. An exhaustive search over all possible parameter settings requires 2,508,800 encodings per video ($7 \times 10 \times 16 \times 4 \times 10 \times 7 \times 8$). Because it is impossible to perform an exhaustive search of this rate-distortion-complexity space on a mobile device, fast methods for choosing the appropriate set of encoding parameters must be used. We apply DPSPA described in Section 2.9 to the ASL video encoder to find near optimal parameter settings. Applying DPSPA over a range of target bitrates effectively provides a look-up table that specifies the parameter settings to use such that distortion is minimized while satisfying both rate and complexity constraints. The following sections give results when applying DPSPA to the ASL video encoder for two different single pass rate control methods.

5.6 Single pass constant λ rate control method

The DPSPA algorithm is applied to a set of nine training ASL videos and six test ASL videos each having 320×240 frame resolution, 200 frames and a frame rate of 15 fps. The results are reported for fixed values of λ. These experiments are conducted on a Windows XP PC having a 2.01 GHz AMD processor.

The DPSPA is executed for values of $\lambda = \{0.5, 1, 5, 40, 150\}$ corresponding to average bitrate of {215.5, 155.8, 74.2, 28.8, 16.1} kb/s for the default parameter setting on test

Table 5.1: Relative performance of DPSPA parameter setting over the default parameter setting for ASL test videos. A positive bitrate gain implies lower bitrate for DPSPA. Tests were performed on a Windows XP PC with 2.01 GHz AMD processor.

λ	Avg. rate gain	Max speed gain	Avg. speed gain	ΔD_I (dB)
0.6	2.4%	15.7%	14.4%	-0.08
1.1	0.47%	19.2%	16.8%	0
5	26.8%	21.2%	13.7%	-0.16
46	-0.49%	20.1%	15%	-0.67
150	0.66%	18%	13.7%	-0.22

videos. The ROI option `ROI-subme` is not considered in our experiment. The performance of the DPSPA parameter settings is evaluated using the relative gain in bitrate, maximum and average encoding speed improvement, and the loss in intelligibility of DPSPA parameter setting over the default parameter setting. Let $R(\mathbf{q})$, $D_I(\mathbf{q})$, and $C(\mathbf{q})$ correspond to the bitrate, intelligibility distortion measure, and encoding time of a parameter setting $\mathbf{q}$. Since we are running the test on a PC, the average frame rate is very high (approximately higher than 50 fps), resulting in $f(r_f) \approx 0$ in Equation 5.6. Therefore, we compute D_I using Equation 5.8. Let $\mathbf{p}$ and $\mathbf{q}$ represent DPSPA and default parameter settings, respectively. We then define rate gain as $\frac{(R(\mathbf{q})-R(\mathbf{p}))}{R(\mathbf{q})} \times 100$, change in intelligibility as $\Delta D_I = D_I(\mathbf{q}) - D_I(\mathbf{p})$, and speed gain $= \frac{(C(\mathbf{q})-C(\mathbf{p}))}{C(\mathbf{q})} \times 100$.

As demonstrated in Table 5.1, the DPSPA parameter settings provide average speed improvement of 15%, and average bitrate reduction of 6% with little decrease in intelligibility. A difference of approximately 1.5 dB corresponds to a statistical change in subjective intelligibility score [86]. Therefore, the average decrease in intelligibility shown in Table 5.1 will not significantly reduce the perceived intelligibility.

The reductions in complexity result from choosing an appropriate set of encoding parameters, including the use of the additional coding modes tuned for region-based coders.

Table 5.2: The default parameter setting (fixed) and DPSPA parameter settings for different values of λ.

Parameter name	Default	λ_{DPSPA}				
		0.6	1.1	5	46	150
`subme`	5	3	3	3	2	2
`ref`	1	3	1	1	1	1
`part`	8	6	8	8	3	3
`trellis`	1	0	0	2	2	2
`backgrd-part`	–	8	7	3	6	7
`ROI-MS`	–	6	5	6	6	6

Table 5.2 lists the encoder parameter settings which have been applied to the test videos. While the default parameter setting uses HEX search for the entire frame, the DPSPA parameter setting exploits the `ROI-MS` option, using either (HEX, DIA, DIA) or (DIA, DIA, DIA) for (face, torso, background), each of which have lower search complexity than the default setting. DPSPA often chooses lower complexity `subme` and `nonROI-part` compared to the default `subme`=5 and P8$\times$8, I8$\times$8, I4$\times$4 for background macroblocks. Since DPSPA generates parameter settings that trade off joint rate-intelligibility cost with encoding time, it does not always pick parameters having lower complexity than the default setting. For example in Table 5.2, for $\lambda_{DPSPA} = 0.6$, DPSPA picks three reference frames instead of one reference frame. The additional computation cost is mediated by the corresponding reduction in distortion.

5.7 Single pass constant average bitrate rate control method

Three combinations of training and test sets are created from a collection of 8 indoor ASL videos filmed on a static background, and 8 outdoor ASL videos filmed on a busy street. The segmentation into ROI and non-ROI is performed offline for each video. The three cases correspond to training and testing on only the indoor videos, only the outdoor videos,

and on both the indoor and outdoor videos. The DPSPA is applied to a set of four training ASL videos and four test ASL videos each having 176×144 frame resolution, 200 frames and a frame rate of 15 fps. These experiments are conducted on a Windows XP PC with a 2.01 GHz AMD processor and on a HTC TyTN II cell phone with a 400 MHz Qualcomm MSM7200 ARMv6 processor. Three bitrates are considered: 15, 30 and 60 kb/s. We consider all the three ROI parameters and four x264 parameters described in Section 5.4.

On a PC, due to high frame rates (> 50 fps) we compute D_I using Equation 5.8 only, and ignore the frame rate component $f(r_f)$. On a cell phone since the frame rates are lower than 13 fps, we approximately compute D_I by adding Equations 5.8 and 5.6. In Equation 5.6, $f(r_f)$ is computed using the average frame rate instead of the instantaneous frame rate.

The DPSPA parameter settings are applied to the ASL test videos and ΔD_I and speed gain are measured as described in the previous section. The intelligibility distortion measure D_I is computed using Equation 5.8. As demonstrated in Table 5.3, the DPSPA parameter settings provide average speed improvements of approximately 45% on the PC with little decrease in intelligibility. A difference of approximately 0.2 corresponds to a statistical change in subjective intelligibility score [87]. Therefore, the average increase in D_I on a PC shown in Table 5.3 will not significantly reduce the perceived intelligibility compared to the x264 default parameter setting.

On a cell phone, we compare the default parameter setting with the highest speed DPSPA parameter setting. From Table 5.4, we find that on an average the DPSPA parameter setting improves speed by 127.3%, and lowers D_I by 0.66 over the default parameter setting. Specifically for outdoor videos at 60 kb/s, using DPSPA parameter setting improves the frame rate from 4.2 fps to 10.6 fps, and lowers D_I from 1.63 to 0.44, which corresponds to a subjective score improvement from 'very difficult' to 'very easy.' We also find that with increase in bitrate, using DPSPA parameter setting improves speed and ΔD_I. This can be attributed to the slow encoding speed of the default parameter setting at higher bitrates. Similar observation was also made in Section 4.5.

Tables 5.3 and 5.4 demonstrate that for both the PC and cell phone platforms, the largest speed gain is obtained on the outdoor test videos. Because these videos were filmed on a busy street, the level of background activity is high. The x264 encoder must spend

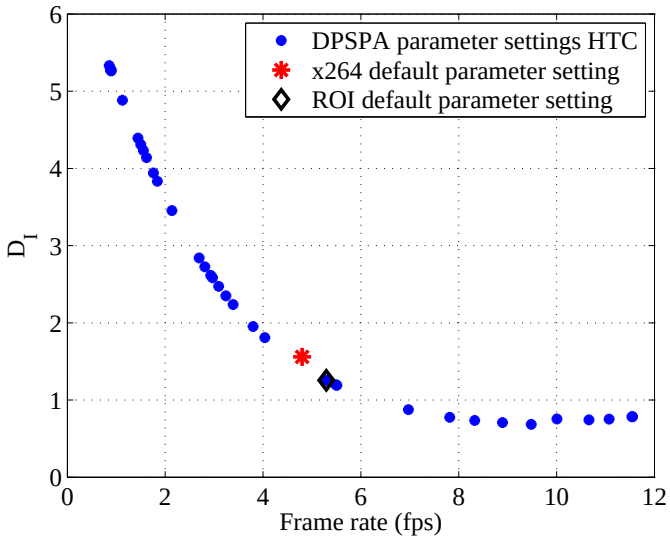


Figure 5.1: Performance of DPSPA parameter settings vs. default parameter setting on the HTC TyTN II cell phone for outdoor ASL videos at 30 kbps.

computational resources encoding these regions, whereas the ROI-optimized encoder can use very coarse, low-complexity parameter options. The overall speed improvement of the ROI-optimized encoder depends on the relative level of activity in the non-ROI.

Tables 5.3 and 5.4 compare the performance against the x264 default parameter settings, which were chosen heuristically by its developers to provide good R-D performance at a reasonable encoding speed. We apply this default parameter setting to our ROI-optimized encoder and call it the *ROI default parameter setting*. Figure 5.1 shows that both the ROI and x264 default parameter settings are not fast enough for real-time performance on the cell phone. DPSPA provides points which allow the encoder to run at or above 10 fps on the cell phone, the nominal limit for full ASL conversation [89].

DPSPA provides a collection of parameter settings which are appropriate for the specific test device on which it is run. While DPSPA can be executed on the cell phone platform, it is useful to investigate if the parameter settings generated on the PC can still approximate the D-C convex hull on the cell phone. The set of encoding parameters computed by DPSPA

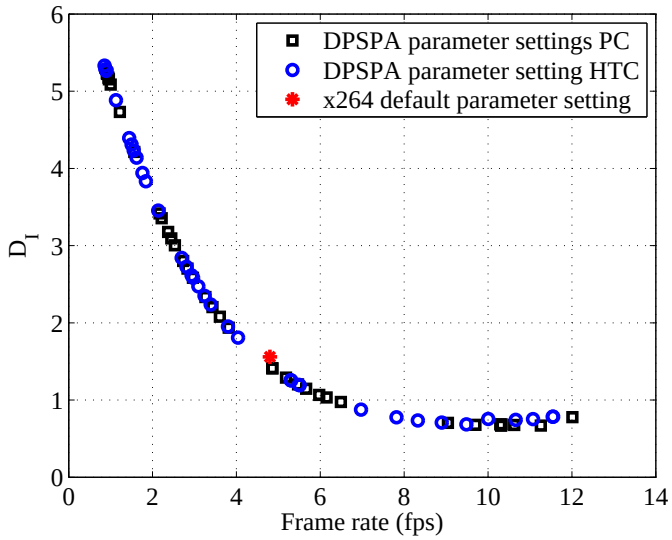


Figure 5.2: Performance of the PC-based and HTC TyTN II cell phone-based parameter settings on the cell phone for outdoor ASL videos at 30 kbps.

when run on the PC is applied to the test videos encoded on the cell phone. Despite differences in the exact parameter settings chosen, the PC-generated settings perform very close to the cell phone-generated settings. Figure 5.2 illustrates the D-C curves for the outdoor test videos at 30 kbps, comparing both collections of parameter settings. In this case, the training required for DPSPA to generate the lookup table can be done on a PC and used on the cell phone without loss in performance.

On the PC, the DPSPA often picks all `ROI-MS` options, while on the cell phone (HEX, UMH, DIA) is preferred over (UMH, HEX, DIA) and (UMH, DIA, DIA) options. This shows that on a cell phone, better intelligibility-complexity tradeoff is obtained by using higher complexity UMH for the hand macroblocks instead of the face macroblocks. Because the location of the face does not vary significantly between frames, a fast motion search algorithm (HEX) is sufficient for identifying the appropriate motion vectors. The signer's hand movements are much wider over the frame, and accurate motion vectors are identified using a higher complexity motion search (UMH).

Table 5.3: Intelligibility distortion difference (ΔD_I) and speed gain of DPSPA parameter setting over the x264 default parameter setting on a 2.01 GHz PC for different pairs of training and test videos. Negative value for ΔD_I indicates a higher intelligibility distortion for DPSPA.

Bitrate (kbps)	Indoor		Outdoor		Indoor & Outdoor	
	ΔD_I	speed gain	ΔD_I	speed gain	ΔD_I	speed gain
15	≈0	31.2%	0.03	43%	-0.01	40.8%
30	-0.05	41.3%	0.05	48.2%	-0.01	45.8%
60	-0.03	45%	0.07	54.4%	0.02	50.7%
Average	-0.03	39.2%	0.05	48.5%	≈0	45.8%

As parameter settings are generated from highest to lowest complexity by DPSPA, the `subme` option (associated with the non-ROI) first decreases from its highest to lowest option while the `ROI-subme` is retained at its highest option. Therefore, DPSPA appropriately reduces encoding complexity by choosing `subme` options that favor lower distortion of the ROI over the non-ROI, and lower complexity in the non-ROI versus the ROI.

We compare the x264 default parameter setting with the highest frame rate DPSPA parameter setting for the three bitrates on the cell phone. Each of the DPSPA parameter settings allows the encoder to operate above 10 fps. For both `subme` and `ROI-subme`, the DPSPA parameter setting chooses option 1 or 2, which has lower complexity compared to the default `subme` = 5 option. For motion search, DPSPA picks DIA search for all regions instead of the higher complexity HEX default option.

5.8 Summary

In this chapter, an ROI-based ASL video encoder that is optimized for rate-intelligibility was presented. Three new ROI-based encoder parameters were proposed for the ASL en-

Table 5.4: Intelligibility distortion difference (ΔD_I) and speed gain of DPSPA parameter setting over the x264 default parameter setting on a HTC TyTN II cell phone for different pairs of training and test videos.

Bitrate (kbps)	Indoor		Outdoor		Indoor & Outdoor	
	ΔD_I	speed gain	ΔD_I	speed gain	ΔD_I	speed gain
15	0.26	109.2%	0.59	107.1%	0.48	102.6%
30	0.55	111.4%	0.77	140.7%	0.6	137.5%
60	0.65	130%	1.2	155.1%	0.88	152.3%
Average	0.49	116.9%	0.85	134.3%	0.65	130.8%

coder that allow lower complexity options to be used for non-ROI regions, resulting in complexity gains without affecting intelligibility. The DPSPA is used to find parameter settings at different bitrates, which approximate the rate-intelligibility-complexity convex hull. We perform the rate-intelligibility-complexity optimization on the encoder when using two different single pass rate control methods. On a PC, for the encoder with the constant λ single pass rate control scheme, the DPSPA parameter setting yields a maximum speed improvement of 21.2% over the x264 default parameter setting with negligible loss in intelligibility. When using the constant average bitrate scheme, the DPSPA parameter setting yields a maximum speed improvement of 54.4% over the default parameter setting on a PC with negligible loss in intelligibility. On a cell phone, the DPSPA parameter setting yields a maximum speed improvement of 155% (corresponding to 6.4 fps frame rate improvement), while significantly decreasing the intelligibility distortion over the default parameter setting. Finally, for the ASL video encoder on the cell phone, the DPSPA parameter settings can be obtained on a PC without loss in performance.

Chapter 6

BATTERY POWER SAVING FOR ASL VIDEO COMMUNICATION OVER CELL PHONES

6.1 Introduction

Battery life is an important parameter that consumers consider when buying a cell phone. Shorter cell phone battery life defeats the mobility functionality, making the cell phone less useful. Running a computationally intensive application such as video conferencing on a cell phone affects battery life. In this chapter, we will review prior work in improving battery life of a HTC TyTN II cell phone when running the MobileASL application. We will present a new method that adaptively controls the backlight of the screen during ASL conversation on MobileASL, and show that it further improves battery life over earlier methods.

6.2 Prior work

In ASL communication, users take turns in signing, i.e., when one person is signing, often the other person is listening. In prior work, a frame is classified as signing/listening, and the listening frame is assigned lower encoding complexity, by either dropping it [90], lowering its resolution, or both [91]. This lowers the total power consumption without affecting the intelligibility of the video. We describe the two prior methods below.

6.2.1 Variable Frame Rate Method

In the variable frame rate method (VFR), a simple thresholding is used to detect whether a video frame is signing/listening [92], [90]. First, the difference between the current frame I_k and previous frame I_{k-1} is computed by

$$d(k) = \sum_{i,j} |I_k(i,j) - I_{k-1}(i,j)|. \tag{6.1}$$

Using ASL training videos, a threshold τ is determined, such that difference values $d(k) > \tau$ are labeled as signing, or otherwise labeled as listening. In the same study, this approach was shown to have a classification accuracy of 84.6%. During the listening portions of the video, the frame rate is dropped from 15 fps to 1 fps. That is, during one second of listening, only one frame is encoded and transmitted, although the cell phone camera captures 15 fps. The reduced frame rate results in choppy video during listening parts. The savings in encoding complexity results in additional battery life of 23 minutes over the average battery life of 284 minutes [90], [91]. The low computational complexity of the VFR method makes it suitable for use on a cell phone.

6.2.2 Variable Spatial Resolution Method

The variable spatial resolution method (VSR) downsamples the listening frame resolution by one-fourth, while retaining the original resolution for signing parts of the video [91]. In MobileASL, this would imply 96× 80 resolution for signing and 48×40 for listening. The downsampled frame is upsampled by two in both dimensions at the decoder to maintain the original spatial resolution. This causes the frame to be blurry during listening periods.

The VSR uses the same thresholding method as in VFR for detecting signing/listening, and provides similar battery savings as VFR. Its advantage is that the frame rate is maintained the same during both signing and listening parts of the video. Another method that combines both lower spatial resolution and frame rate for listening parts of the video was also presented in [91] and it provided battery life savings of 31 minutes (i.e., 8 minutes of improvement over VSR or VFR).

6.3 Backlight Control

Most current PDAs and cell phones are equipped with a fairly sized LCD screen, which allows the user to watch movies, browse the internet, read e-mails, etc. with ease. This is a step ahead of earlier cell phones that were restricted to making voice calls and sending text messages. The HTC TyTN II cell phone has a 320×240 resolution LCD screen. One of the drawbacks encountered with larger LCD screens is limited battery life. A study in [93]

showed that LCD backlight in a PDA consumes 50% of the total power. Therefore, lowering the use of LCD backlight should improve the overall battery life of the cell phone.

There exists prior work in LCD backlight control to extend battery life. Chang et al. developed an approach to save power by dimming the backlight of a LCD display while restoring the brightness of the screen by applying image compensation techniques [94]. The authors proposed another idea where the backlight was dimmed when low ambient light was detected [93]. A dimly lit LCD screen is visible in low ambient light or darkness. However, this scheme requires a photo sensor in the mobile device to detect the ambient light. Iyer et al. [95] use an organic LED display that uses energy proportional to the light output of the display. To save energy, they implement *dark windows* that either reduce the brightness or change colors of the non-region-of-interest. Four different dark windows are tested. Two of them are brightness control: background half and full dim, and the other two are color control: background gray and green scale.

6.3.1 Adaptive backlight control for ASL communication

In this section, we propose a battery power saving scheme that dims the LCD screen of the HTC TyTN II cell phone based on whether the MobileASL user is signing or listening. Due to turn-taking in ASL, when one person signs, the signer's backlight can be dimmed, since the other person is listening. However, the backlight of the listener is at normal brightness to allow signs to be seen clearly. This adaptive dimming of backlight is demonstrated to improve cell phone battery life.

The thresholding approach described in Section 6.2.1 is used to classify signing/listening activity. We consider three states of operation for our backlight control: signing, listening and sleep. During the signing state, the backlight is dimmed, and during the listening state the backlight is made bright. We use the Windows API DevicePowerNotify() to control the brightness of the screen. To avoid flicker due to sudden fluctuation between states, a buffer is used that stores states up to 2 seconds in the past (or 23 past frames). If more than 67% of the buffer is labeled listening, then the screen is turned bright, as it is most likely that the current state is listening. Otherwise, the screen is dimmed.

We transmit the current state to the other person's cell phone. If both users are in listening mode for more than 20 seconds (or 240 frames), it is very likely that both of them are not in front of their cell phones. The state is changed to sleep state and both screens are dimmed. If an activity is detected on either of the cell phones after the sleep mode is turned on, both screens are made bright. It is likely that one of the users is at the cell phone to continue his/her conversation, and therefore a bright screen on both ends is needed.

When both users begin to sign, both screens are turned bright. Although there is turn taking in ASL conversation, the listening person could always interrupt. So, in this case the signer's screen would turn bright allowing him/her to see other person signing. We illustrate the different operating modes of our adaptive backlight control in Figure 6.1.

We refer to MobileASL using no power savings as the default method. We compare the performance of default, VFR, adaptive backlight control (ABC), and combined implementation of ABC and VFR. The combined ABC and VFR approach encodes a video at full frame rate with screen dimmed during signing, and during listening parts it drops encoding frames to 1 fps (as in VFR) with the screen made bright. (We do not consider VSR since it has not been implemented for conversational ASL videos on the cell phone.) For each method, we run the MobileASL for 30 minutes using prerecorded conversational ASL video. During the test, the battery life on the cell phone is measured every 5 seconds and stored on the cell phone. A plot of battery life vs. time for each power saving method is linear, and it can therefore be linearly extrapolated to 0% battery life, to give us estimated battery life. This is illustrated in Figure 6.2.

We run MobileASL for each power saving scheme on two cell phones using two different ASL videos, and average the total battery life across cell phones and ASL videos. Running MobileASL using the default approach results in 198 minutes of average total battery life. The improvements in battery life when using different power saving schemes over the default approach are listed in Table 6.1. For VFR, we obtain 10 minutes of power saving compared to 23 minutes reported in [91]. The measured results are lower because the results in [91] correspond to the best case, i.e., when the entire video is classified as listening with the camera facing the wall. Since all frames are labeled as listening the video is encoded at 1 fps. Also, since the camera faces the wall, all macroblocks will be encoded using low

(a) (b)

(c)

Figure 6.1: Different modes of adaptive backlight control: (a) screen bright when current user listens and the other user signs; (b) screen dim when current user signs and other user listens; and (c) screen bright when both users sign.

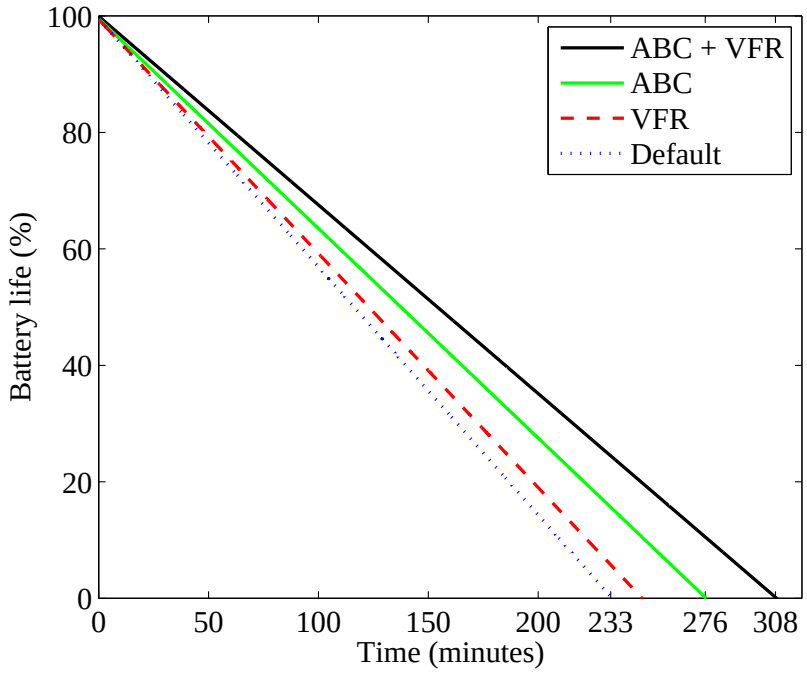


Figure 6.2: Battery life of a HTC TyTN II cell phone (HTC32) vs. time when running MobileASL using different power saving schemes using a prerecorded video (Gina.yuv).

complexity skip blocks.

Table 6.1 shows that our approach combined with VFR gives an average improvement in battery life of 54 minutes or 27.3% improvement over the default approach. This approach saves power during both signing and listening parts of the video. During signing parts it dims the screen and during listening parts it encodes only 1 fps. With only ABC, we get 30 minutes of extended battery life that corresponds to 13.8% improvement over the default approach. Although VFR is seen to give only 10 minutes of improvement over the default, when combined with ABC it helps extend the battery life by 24 minutes over ABC alone.

6.4 Summary

We presented an approach to extend battery life by controlling the backlight of the LCD screen. The brightness of the screen is either turned dim/bright depending on whether the

Table 6.1: Comparing different power saving methods for the MobileASL application relative to the default approach. The results are averaged over two prerecorded ASL videos and two cell phones.

Power saving method	Average battery life (minutes)	Additional battery life (minutes)	% increase in battery life
Default	198	0	0
VFR	208	10	5%
ABC	228	30	15.2%
ABC + VFR	252	54	27.3%

user is signing/listening. This approach provides significantly higher battery life savings over earlier methods that only lower encoding complexity during listening parts of the ASL conversation. When combining VFR with our ABC method, we obtain 54 minutes of additional battery time over the default case. A subjective study with users needs to be conducted to evaluate the usability of this approach.

Chapter 7

RATE-DISTORTION-COMPLEXITY OPTIMIZATION OF THE H.264 VIDEO ENCODER FOR TIME-VARYING NETWORKS

7.1 Introduction

In previous chapters we considered the bitrate to be constant (except for chapter 5) and performed optimization for distortion and complexity of the video encoder. However, in transmission media such as Wi-Fi and 3G networks, the bandwidth varies over time. A constant bitrate rate control scheme is a disadvantage in applications where the bandwidth is time varying, if the encoder is restricted to operate at the lowest bitrate. This motivates the use of a bitrate adaptive encoder since it allows the use of higher bitrates (and higher quality) when higher bandwidth is available.

In prior work, De Vito et al. proposed a rate control method to allow the bitrate of a group-of-pictures (GOP) to adapt to the available bandwidth [96]. However, this method is somewhat limited in practice since channel bandwidth can vary within a GOP interval. Schierl et al. used a combination of temporal scalability and bitrate switching to adapt the encoded bitrate to the available bandwidth [97]. Bitrate switching is done by having multiple bitstreams at different bitrates. Encoding multiple bitstreams on a cell phone is impractical. Temporal scalability is achieved by dropping non-referenced frames resulting in lower bitrate. Dropping a non-reference frame in ASL video could lower intelligibility since this frame could contain important information. In this chapter, we propose a rate control scheme which allows encoded bitrate to adapt on a frame-level basis.

Internet-based video chatting services provided by Skype and Google have become popular and cheap ways to videoconference between remote users. Since these services have to support time varying network bandwidths, it is very likely that they employ variable bitrate video encoders. De Cicco et al. investigated Skype's performance as a function of variation in available bandwidth, loss rate, and jitter considering Skype as a blackbox, since Skype's

algorithm is proprietary [98]. They found that Skype varies the encoded video frame rate, frame quality, frame resolution, and transmission rate based on the bandwidth, loss rate, and jitter. When there is a sudden increase in the number of lost packets, Skype increases its packet size. At lower bitrates, Skype maintains high frame rate, but lowers the frame resolution. Also, Skype adapts to both increases and decreases in available bandwidth, thereby indicating that it uses a bitrate adaptive video encoder.

To allow the MobileASL application to adapt to varying bandwidth, we modify the rate control of the x264 encoder. Our proposed rate-distortion-complexity optimization scheme is illustrated in Figure 7.1. We pre-compute GBFOS-basic parameter settings for different bitrates and complexities and store them in a lookup table. When a frame is encoded, the current bandwidth and available computational resources are used to select a parameter setting from the lookup table, resulting in higher ASL intelligibility and frame rate.

7.2 Adaptive bitrate x264 video encoder

Rate control allows selection of encoding parameters to maximize quality under bitrate and decoder video buffer constraints. The rate control in H.264 can be performed at three different levels of granularity - group of pictures level, picture level, and macroblocks level [99]. At each level, the rate control algorithm selects the quantization parameter (QP) values that determine the quantization of the transformed coefficients. As the QP increases, the quantization step size increases and the bitrate decreases. The rate control in x264 is based on the libavcodec's rate control implementation [23], which is mostly empirical. Five different rate control modes are available in x264: there are one two-pass mode and four one-pass modes [14].

In this section, we describe our modification to the x264 rate control algorithm to allow bitrate adaptation to available bandwidth. Specifically, we modify the constant bitrate (CBR) one-pass rate control method of the x264 encoder to obtain an adaptive bitrate (ABR) one-pass rate control method. We modify the rate control parameters (`wanted_bits_window` and `cplxr_sum`) that are used to determine the QP for a given frame. We refer to the available bandwidth as the target bitrate. The x264 encoder averages these rate control parameters over previous frames to meet a global average target bitrate. We

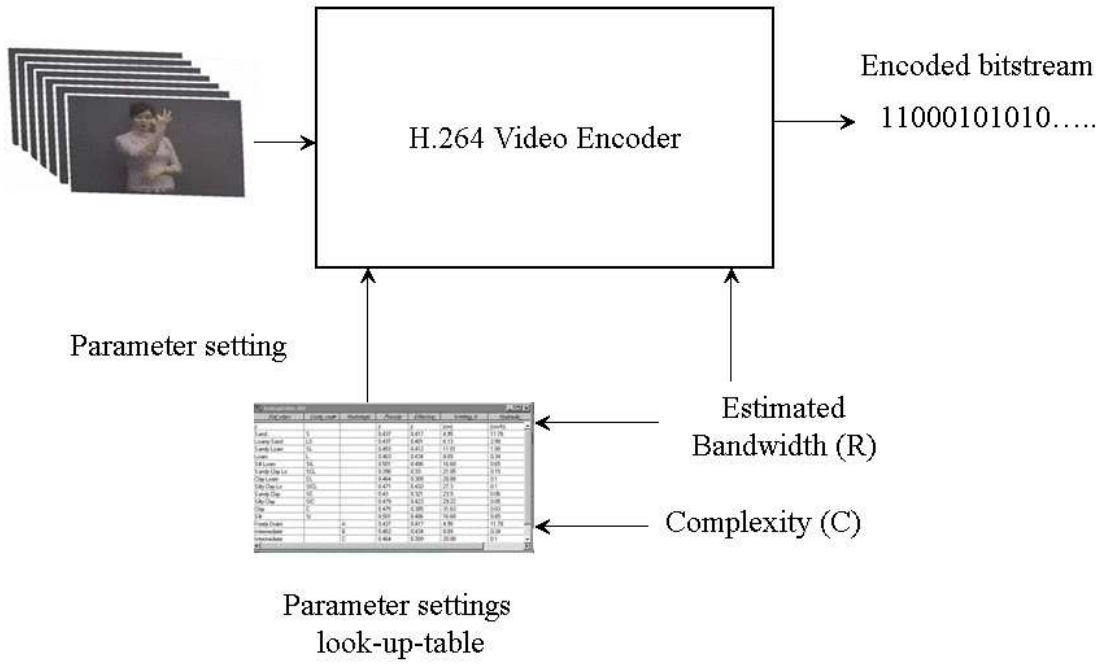


Figure 7.1: Schematic diagram of the proposed rate-distortion-complexity optimization scheme using a look-up-table containing GBFOS-basic parameter settings. R is the estimated bandwidth and C is the computational resources available to the encoder.

find that when the target bitrate for a frame decreases, the actual instantaneous bits used by the x264 encoder does not lower. Therefore, when the target bitrate decreases, we reset the rate control parameters (instead of averaging them). This modification allows x264 to meet the current target bitrate instead of a global average target bitrate. However, when the target bitrate either increases or remains unchanged, we average the rate control parameters across previous frames.

Figure 7.2: The performance of the adaptive bitrate x264 encoder to varying target bitrate.

Figure 7.2 shows the performance of our rate control scheme in response to variations in target bitrate when applied to an ASL QCIF video at 15 fps tested on a 2.01 GHz PC. We use the default parameter setting in our test. We consider this pattern for the target bitrate since it is within the range of the 3G network [100]. We clearly see that our rate control method adapts well to variations in the target bitrate.

7.3 Results

We ported the x264 encoder with our rate control method to the HTC TyTN II cell phone. Since there is no off-the-shelf available bandwidth estimation software which runs on the cell phone, we simulate a bandwidth variation pattern that operates within the allowed bandwidth range of the 3G network. For future work, an approach described in [100] could be implemented on the cell phones to measure available bandwidth. In our experiment we specifically consider bitrate variation between 40 to 100 kb/s, since bitrates above 100 kb/s do not significantly improve the quality of a QCIF ASL video.

We use two ASL QCIF videos having 700 frames each at 15 fps in our test labeled ‘indoor’ and ‘outdoor.’ We consider a target bitrate that varies with time as illustrated

in Figure 7.3. We vary the target bitrate every 50 frames or 3.34s. This bitrate variation duration is shorter than the 20 s duration considered in [101] and 100 s duration considered in [98]. We consider three different schemes to encode the videos: CBR at 40 kb/s with default parameter setting; ABR with default parameter setting; and ABR with GBFOS-basic parameter settings. For CBR, we specify a constant bitrate of 40 kb/s since this is the lowest bitrate of the given bitrate range. (Packet loss can occur if operating bitrate exceeds the available bandwidth. Therefore, we set the CBR encoder to the lowest bitrate range.) Since the x264 encoder is utilizing most of the computational resources on the cell phone, we use the lowest complexity GBFOS-basic parameter setting (`ref` = 1, `part` = 1, `subme` = 1, `trellis` = 1) in our test.

The videos are encoded on the cell phone for different rate control schemes and parameter settings on the HTC cell phone. We plot the instantaneous bitrate per frame in Figure 7.3. From Figure 7.3, we find that our adaptive rate control scheme adapts well to the varying target bitrate, while the constant bitrate method does not adapt, as expected. The average bitrate, intelligibility distortion, and frame rate for all three cases are listed in Table 7.1. Again, the outdoor video has higher intelligibility distortion than the indoor video.

When using the default parameter setting, ABR results in a very small improvement in intelligibility over CBR, although ABR has higher bitrate. This is because the distortion intelligibility metric penalizes ABR for having a lower frame rate. That is, an increase in bitrate does not significantly improve intelligibility at low frame rates. When using the GBFOS-basic parameter setting with ABR, there is both increase in frame rate and bitrate, which results in an improvement in intelligibility over the default parameter setting using the CBR method by 0.2 and 0.32 for indoor and outdoor videos, respectively. This is significant since a difference of 0.2 corresponds to a change in subjective intelligibility score of Ciaramello et al. [87]. Therefore, by using the GBFOS-basic parameter settings and ABR, we increase the average bitrate, intelligibility and frame rate.

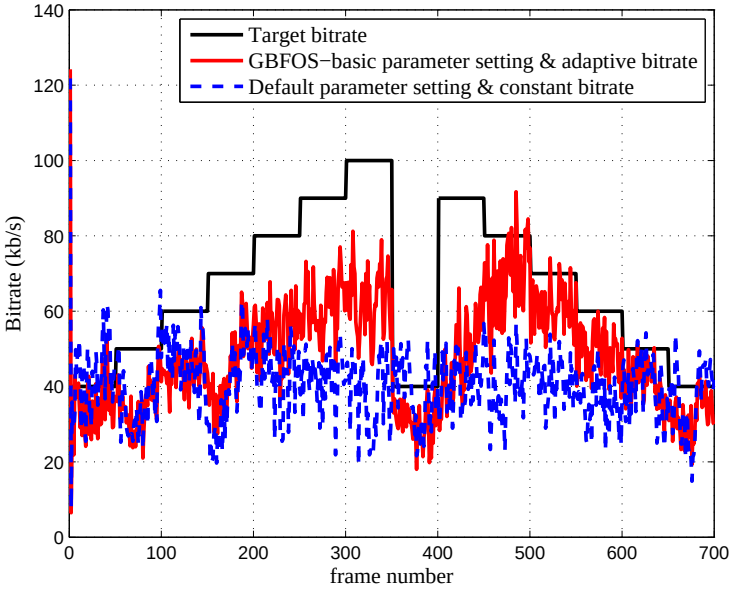


Figure 7.3: Comparison between adaptive bitrate x264 encoder using GBFOS-basic parameter setting vs. using x264 encoder with default parameter setting at 40 kb/s on a HTC TYTN II cell phone.

Table 7.1: Performance of the x264 encoder on a HTC cell phone when using different rate control methods and parameter settings, with varying target bitrate, when applied to (a) indoor video and (b) outdoor video.

Rate control method	Parameter setting	Average bitrate (kb/s)	Intelligibility distortion	Frame rate (fps)
CBR at 40 kb/s	Default	40.3	0.64	6.8
ABR	Default	47.3	0.63	6.6
ABR	GBFOS-basic	47.7	0.43	13.1

(a)

Rate control method	Parameter setting	Average bitrate (kb/s)	Intelligibility distortion	Frame rate (fps)
CBR at 40 kb/s	Default	40	1.11	5.8
ABR	Default	48	1.08	5.7
ABR	GBFOS-basic	48.4	0.79	9.4

(b)

Chapter 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this thesis, we proposed four fast offline parameter settings selection algorithms that choose parameter settings that trade off well between PSNR and encoding time. Our algorithms take less than 8.5% of the number encodings required by exhaustive search of parameter settings, but have similar PSNR performance. We compared our algorithms with the multiobjective particle swarm optimizer, which generates random parameter settings having low D-C spread. Our algorithms improve the encoding speed over the x264 default parameter setting on PC and cell phone platforms by up to 37.4% and 94.1%, respectively, with little difference in PSNR.

To show the generality of our algorithms, we applied them to the H.263+ encoder. To speed up the motion estimation for ASL video compression in x264, we introduced a new ROI-based motion search parameter, which uses different motion search methods for different regions of the video. We included this parameter with other x264 encoder parameters, and applied our parameter settings selection algorithms on them.

On a cell phone platform, our algorithms generate parameter settings whose PSNR-encoding time performance are close to the estimated convex hull parameter settings. Our parameter settings are robust to the training platform, and we have shown that the PC-based parameter settings perform well on a cell phone. Our parameter settings are also robust to different signers used in training and test videos.

We applied our algorithms to the ROI-based video encoder specific to ASL developed at Cornell University. This encoder uses an ASL intelligibility distortion metric instead of the MSE. Our parameter settings trade off well between encoding time and ASL intelligibility. On a PC and cell phone platforms, our parameter setting improves the encoding speed by up to 54.4% and 155%, respectively, over the x264 default parameter setting, with negligible

or no loss in intelligibility.

We presented two methods to extend cell phone battery life when running MobileASL. Both methods use information about whether the user is signing or listening. The first method controls the backlight of the cell phone, and the second method combines the first method with dropping the encoder frame rate to 1 fps during listening parts of the video. The combined approach yields 54 minutes additional battery life which corresponds to 27.3% improvement. Finally, we presented an adaptive bitrate rate control method that allows the x264 encoder bitrate to adapt to time-varying available bandwidth. Using our parameter settings with this encoder improves the average bitrate, intelligibility, and encoding speed over the constant bitrate x264 encoder.

Future video standards, such as High Efficiency Video Coding (also referred to as H.265), are expected to perform better than the H.264 standard [102]. It is very likely that these standards will have more tools and higher encoding complexity. Our parameter settings selection algorithms can easily be extended to future standards to jointly optimize them for rate, distortion, and complexity.

Our MobileASL software will soon be publicly available, and is expected to revolutionize the way Deaf people communicate by giving them access to video cell phones. Video relay services should be accessible through MobileASL, allowing Deaf people to use cell phones to communicate with hearing persons. Our parameter settings selection algorithms, battery life saving methods, and the rate control scheme can improve the user experience of MobileASL on cell phones by providing higher intelligibility and frame rates necessary for ASL video communication.

8.2 Future work

8.2.1 Computing available bandwidth on cell phones

In Chapter 7, we presented a bitrate adaptive rate control method for the x264 encoder, which allows the bitrate per frame to vary according to the available bandwidth. We did not measure the available bandwidth in real-time, but simulated the bandwidth variation. It will be useful to develop a bandwidth estimation algorithm for cell phone platforms,

and use the real-time bandwidth estimate as the input to our encoder. There are several publicly available bandwidth estimation tools, such as pathload [103], pathrate [104], and pathChirp [105] for Linux platforms. However, there are none available so far for cell phones. It is possible that these programs could be adapted for use on cell phones.

8.2.2 Speed control algorithm

A speed control algorithm was developed for the x264 encoder, which allows the encoder to adapt its parameter settings per frame based on the available computational resources [106]. The speed control algorithm uses a lookup table of parameter settings. In current cell phones, the x264 encoder consumes almost all of the processor resources even at lower complexity parameter settings. In the future, when cell phones with faster processors are available, the speed control together with a lookup table of our parameter settings could be used to control the encoder complexity on a frame-level basis.

8.2.3 User study for battery power savings method

In Chapter 6, we present two methods to save cell phone battery power when running MobileASL. A user study will have to be carried to determine the usability of these methods, and these methods may need to be adapted based on user feedback. The user study will involve ASL fluent signers using MobileASL with different power savings schemes, and comparing our proposed methods with the earlier methods.

We would like to combine the adaptive backlight control with variable frame rate and variable spatial resolution on the cell phone, and analyze its battery power savings. It is likely that this combined approach will give higher battery power savings. The MobileASL research group is planning to conduct a field study, where Deaf participants will be given the MobileASL phones for a few months to use and test the different power saving methods.

BIBLIOGRAPHY

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, July 2003.

[2] J. Chon, N. Cherniavsky, E. A.Riskin, and R. E.Ladner, "Enabling access through real-time sign language communication over cell phones," in *Proceedings of the 43rd Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2009.

[3] J. R. Goldschneider, "Lossy compression of earth science data," Ph.D. dissertation, University of Washington, June 1997.

[4] E. A. Riskin, "Optimal bit allocation via the generalized BFOS algorithm," *IEEE Trans. Inf. Theory*, vol. 37, no. 2, pp. 400–402, March 1991.

[5] *Advanced video coding for generic audiovisual services.* ITU-T Recommendation H.264, March 2005.

[6] x264. [Online]. Available: http://developers.videolan.org/x264.html

[7] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Process. Mag.*, vol. 15, pp. 74–90, Nov. 1998.

[8] ——, "Video compression – from concepts to the H.264/AVC standard," *Proc. IEEE*, vol. 93, pp. 18–31, Jan. 2005.

[9] I. Richardson, *H.264 and MPEG-4 video compression.* John Wiley & sons, 2003.

[10] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: tools, performance, and complexity," *IEEE Circuits Syst. Mag.*, vol. 4, pp. 7–28, Jan. 2004.

[11] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598–603, July 2003.

[12] G. J. Sullivan, P. Topiwala, and A. Luthra, "The H.264/AVC Advanced Video Coding standard: overview and introduction to the fidelity range extensions," in *Proceedings of the SPIE*, vol. 5558, Nov. 2004, pp. 454–474.

[13] *MPEG-4 AVC/H.264 video codec comparison*, CS MSU Graphics & Media Lab Video Group, December 2005. [Online]. Available: http://www.compression.ru/video/index.htm

[14] L. Merritt and R. Vanam, "Improved rate control and motion estimation for H.264 encoder," in *Proceedings of ICIP*, vol. 5, Sept. 2007, pp. 309–312.

[15] JM ver. 10.2. [Online]. Available: http://iphome.hhi.de/suehring/tml/index.htm

[16] E. Yang and X. Yu, "Rate distortion optimization of H.264 with main profile compatibility," *IEEE International Symposium on Information Theory*, pp. 282–286, July 2006.

[17] E. W. Dijkstra, *A note on two problems in connexion with graphs.* Mathematisch Centrum, Amsterdam, Netherlands, 1959, vol. 1.

[18] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Process.*, vol. 9, no. 2, pp. 287–290, 2000.

[19] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation." *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 5, pp. 349–355, 2002.

[20] X. Yi, J. Zhang, and N. Ling, "Improved and simplified fast motion estimation for JM," *JVT-P021*, July 2005.

[21] X. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. on Image Proc.*, vol. 9, no. 3, pp. 501–504, 2000.

[22] MobileASL. [Online]. Available: http://mobileasl.cs.washington.edu/index.html

[23] "ffmpeg," http://ffmpeg.org/.

[24] G. Côté, B. Erol, M. Gallant, and F. Kossentini, "H.263+: video coding at low bit rates," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 7, pp. 849–866, 1998.

[25] F. Ciaramello, R. Vanam, J. Chon, S. Hemami, E. Riskin, and R. Ladner, "Rate-intelligibility-complexity optimization for real-time video conferencing on a mobile device," in *Fifth Intl. Workshop on Video Processing and Quality Metrics for Consumer Electronics*, January 2010.

[26] P. A. Chou, T. D. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inf. Theory*, vol. 35, no. 2, pp. 299–315, March 1989.

[27] C. A. Coello Coello, G. Toscano Pulido, and M. Salazar Lechuga, "Handling Multiple Objectives With Particle Swarm Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, June 2004.

[28] J. Cardinal, "A Lagrangian optimization approach to complexity-constrained TSVQ," *IEEE Signal Process. Lett.*, vol. 7, no. 11, pp. 304–306, Nov. 2000.

[29] K. Lengwehasatit and A. Ortega, "Rate-complexity-distortion optimization for quadtree-based DCT coding," in *Proceedings of ICIP*, Sep. 2000.

[30] Y. Yang and S. S. Hemami, "Generalized rate-distortion optimizations for motion-compensated video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 6, pp. 942–955, Sept. 2000.

[31] P. L. Tai, C. T. Liu, and J. S. Wang, "Complexity-adaptive search algorithm for block motion estimation," in *Proceedings of ICIP*, Oct. 2001.

[32] J. Zhang, Y. He, S. Yang, and Y. Zhong, "Performance and complexity joint optimization for H.264 video coding," in *Proceedings of the 2003 International Symposium on Circuits and Systems*, May 2003.

[33] K. Ramkishor, P. Gupta, T. S. Raghu, and K. Suman, "Algorithmic optimizations for software-only MPEG-2 encoding," *IEEE Trans. Consum. Electron.*, vol. 50, no. 1, pp. 366–375, Feb. 2004.

[34] I. R. Ismaeil, A. Docef, F. Kossentini, and R. K. Ward, "A computation-distortion optimized framework for efficient DCT-based video coding," *IEEE Trans. Multimedia*, vol. 3, no. 3, pp. 298–310, Sep. 2001.

[35] D. N. Kwon, P. F. Driessen, A. Basso, and P. Agathoklis, "Performance and computational complexity optimization in configurable hybrid video coding system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 1, pp. 31–42, 2006.

[36] M. van der Schaar and Y. Andreopoulos, "Rate-distortion-complexity modeling for network and receiver aware adaptation," *IEEE Trans. Multimedia*, vol. 7, no. 3, pp. 471–479, June 2005.

[37] A. Ray and H. Radha, "Complexity-distortion analysis of H.264/JVT decoder on mobile devices," *Picture Coding Symposium*, December 2004.

[38] J. Stottrup-Andersen, S. Forchhammer, and S. M. Aghito, "Rate-distortion-complexity optimization of fast motion estimation in H.264/MPEG-4 AVC," in *Proceedings of ICIP*, Oct. 2004, pp. 111–114.

[39] V. Iverson, J. McVeigh, and B. Reese, "Real-time H.264/AVC codec on Intel architectures," in *Proceedings of ICIP*, 2004, pp. 757–760.

[40] M. G. Koziri, G. I. Stamoulis, and I. X. Katsavounidis, "Power reduction in an H.264 encoder through algorithmic and logic transformations," in *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, 2006, pp. 107–112.

[41] Z. Wei, K. L. Tang, and K. Ngan, "Implementation of H.264 on mobile device," *IEEE Trans. Consum. Electron.*, vol. 53, pp. 1109–1116, 2007.

[42] S.-Y. Chien, Y.-W. Huang, C.-Y. Chen, H. Chen, and L.-G. Chen, "Hardware architecture design of video compression for multimedia communication systems," *IEEE Commun. Mag.*, vol. 53, pp. 1109–1116, 2007.

[43] Y.-W. Huang, T.-C. Wang, B.-Y. Hsieh, and L.-G. Chen, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *Proceedings of ISCAS*, 2003, pp. 796–799.

[44] Y.-W. Huang, B.-Y. Hsieh, T.-C. Chen, and L.-G. Chen, "Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 3, pp. 378–401, 2005.

[45] T.-C. Wang, Y.-W. Huang, H.-C. Fang, and L.-G. Chen, "Parallel 4×4 2D transform and inverse transform architecture for MPEG-4 AVC/H.264," in *Proceedings of ISCAS*, 2003, pp. 800–803.

[46] R. R. Osorio and J. D. Bruguera, "High-throughput architecture for H.264/AVC CABAC compression system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 11, pp. 1376–1384, 2006.

[47] C. A. Rahman and W. Badawy, "CAVLC encoder design for real-time mobile video applications," *IEEE Transactions on Circuits and Systems—Part II: Express Briefs*, vol. 54, no. 10, pp. 873–877, Oct 2007.

[48] Y.-W. Huang, T.-W. Chen, B.-Y. Hsieh, T.-C. Wang, T.-H. Chang, and L.-G. Chen, "Architecture design for deblocking filter in H.264/JVT/AVC," in *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, 2003, pp. 693–696.

[49] Y. Hu, Q. Li, S. Ma, and C.-C. Kuo, "Joint rate-distortion-complexity optimization for H.264 motion search," *2006 IEEE International Conference on Multimedia and Expo*, pp. 1949–1952, July 2006.

[50] Z. He, Y. Liang, L. Chen, I. Ahmad, and D. Wu, "Power-rate-distortion analysis for wireless video communication under energy constraints," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 645–658, May 2005.

[51] Z. He, W. Cheng, and X. Chen, "Energy minimization of portable video communication devices based on power-rate-distortion optimization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 5, pp. 596–608, May 2008.

[52] D. Li, Y. Sun, and Z. Feng, "Joint power allocation and rate control for real-time video transmission over wireless system," in *Proceedings of IEEE Globecom*, vol. 4, Dec. 2005, pp. 2164–2168.

[53] L. Su, Y. Lu, F. Wu, S. Li, and W. Gao, "Real-time video coding under power constraint based on H.264 codec," *SPIE Visual Communications and Image Processing*, vol. 6508-1, 2007.

[54] M.-T. Lu, J. J. Yao, and H. H. Chen, "A complexity-aware video adaptation mechanism for live streaming systems," *EURASIP J. Appl. Signal Process.*, vol. 2007, no. 1, pp. 215–215, 2007.

[55] P. Agrawal, S. Chen, P. Ramanathan, and K. Sivalingam, "Battery power sensitive video processing in wireless networks," *The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 1, pp. 116–120, Mar. 1998.

[56] M. Wu, S. Forchhammer, and S. M. Aghito, "Complexity control of fast motion estimation in H.264/MPEG-4 AVC with rate-distortion-complexity optimization," in *Visual Communications and Image Processing 2007*, ser. Proc. SPIE, C. Chen, D. Schonfeld, and J. Luo, Eds., vol. 6508, Jan. 2007, pp. 650 824 – 650 824–11.

[57] P. Yin, A. M. Tourapis, and J. Boyce, "Fast mode decision and motion estimation for JVT/H.264," in *Proceedings of ICIP*, vol. 3, Sept. 2003, pp. 853–856.

[58] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, D. Wu, and S. Wu, "Fast mode decision algorithm for intraprediction in H.264/AVC video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 7, pp. 813–822, 2005.

[59] B. Jeon and J. Lee, "Fast mode decision for H.264," *JVT-J033*, December, 2003.

[60] Z. Chen, P. Zhou, and Y. He, "Fast integer pel and fractional pel motion estimation for JVT," *JVT-F017*, December 2002.

[61] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Proceedings of VCIP*, Jan. 2002, pp. 1069–1079.

[62] I. Werda, F. Kossentini, M. A. B. Ayed, and N. Massmoudi, "Analysis and optimization of UB video's H.264 baseline encoder implementation on Texas Instruments' TMS320DM642 DSP," in *Proceedings of ICIP*, 2006, pp. 3277–3280.

[63] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[64] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[65] R. Vanam, E. A. Riskin, S. S. Hemami, and R. E. Ladner, "Distortion-complexity optimization of the H.264/MPEG-4 AVC encoder using the GBFOS algorithm," in *Proceedings of the IEEE Data Compression Conference*, Snowbird, Utah, March 2007, pp. 303–312.

[66] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[67] S. Z. Kiang, R. L. Baker, G. J. Sullivan, and C. Y. Chiu, "Recursive optimal pruning with applications to tree structured vector quantizers," *IEEE Trans. Image Process.*, vol. 1, no. 2, pp. 162–169, April 1992.

[68] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[69] M. S. Lechuga and G. T. Pulido. Multi-objective particle swarm optimization (MOPSO). [Online]. Available: http://delta.cs.cinvestav.mx/~ccoello/EMOO/mopso.tar.gz

[70] Y. Collette and P. Siarry, *Multiobjective Optimization. Principles and Case Studies*. Springer, August 2003.

[71] YUV QCIF samples. [Online]. Available: http://www.tkn.tu-berlin.de/research/evalvid/qcif.html

[72] R. Vanam, E. A. Riskin, and R. E. Ladner, "H.264/MPEG-4 AVC encoder parameter selection algorithms for complexity distortion tradeoff," in *Proc. of the IEEE DCC*, Mar. 2009.

[73] D. Agrafiotis, C. N. Canagarajah, D. R. Bull, M. Dye, H. Twyford, J. Kyle, and J. T. Chung-How, "Optimized sign language video coding based on eye-tracking analysis," *Visual Communications and Image Proc.*, pp. 1244–1252, 2003.

[74] L. Muir and I. Richardson, "Perception of sign language and its application to visual communications for deaf people," *Journal of Deaf Studies and Deaf Education*, vol. 10, pp. 390–401, 2005.

[75] *Video Coding for Low Bit Rate Communication,*. ITU-T Recommendation H.263, March 1996.

[76] *Video Coding for Low Bit Rate Communication,*. Draft ITU-T Recommendation H.263 Version 2, Sept. 1997.

[77] H.263+ reference software. [Online]. Available: http://whkong.myrice.com/download/src/vcomp/tmn-3.2.0.tgz

[78] A. Cavender, R. Vanam, D. K. Barney, R. E. Ladner, and E. A. Riskin, "MobileASL: Intelligibility of sign language video as constrained by mobile phone technology," *Disability and Rehabilitation: Assistive Technologies, London: Taylor and Francis*, pp. 1–13, January 2007.

[79] C. M. Reed, L. A. Delhorne, N. I. Durlach, and S. D. Fischer, "A study of the tactual and visual reception of fingerspelling," *Journal of Speech and Hearing Research*, no. 33, pp. 786–797, December 1990.

[80] W. W. Woelders, H. W. Frowein, J. Nielsen, P. Questa, and G. Sandini, "New developments in low-bit rate videotelephony for people who are deaf," *Journal of Speech, Language, and Hearing Research,*, pp. 1425–1433, December 1997.

[81] D. Chai and K. N. Ngan, "Face segmentation using skin color map in videophone applications," in *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, no. 4, 1999, pp. 551–564.

[82] S. Daly, K. Matthews, and J. Ribas-Corbera, "Face-based visually-optimized image sequence coding," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, 1998, pp. 443–447 vol.3.

[83] F. M. Ciaramello and S. Hemami, "Quantifying the effect of disruptions to temporal coherence on the intelligibility of compressed american sign language video," in *Proc. SPIE, HVEI '09*, vol. 7240, 2009.

[84] D. Agrafiotis, N. Canagarajah, D. R. Bull, J. Kyle, H. Seers, and M. Dye, "A perceptually optimised video coding system for sign language communication at low bit rates," in *Signal Processing: Image Commun.*, no. 21, 2006, pp. 531–549.

[85] K. Nakazono, Y. Nagashima, and A. Ichikawa, "Digital encoding applied to sign language video," in *IEICE Trans. Inf. & Sys.*, vol. E89-D, no. 6, June 2006.

[86] F. M. Ciaramello and S. S. Hemami, "Complexity constrained rate-distortion optimization of sign language video using an objective intelligibility metric," in *Proc. SPIE, VCIP '08*, vol. 6822, Jan. 2008.

[87] ——, "An objective intelligibility measure for assessment and compression of american sign language video," p. in preparation.

[88] A. Ortega and K. Ramchandran, "Forward-adaptive quantization with optimal overhead cost for image and video coding with applications to MPEG video coders," in *Proc. of IS&T/SPIE Digital Video Compression '95*, February 1995.

[89] J. Harkins, A. Wolff, E. Korres, R. Foulds, and S. Galuska, "Intelligibility experiments with a feature extraction system designed to simulate a low-bandwidth video telephone for deaf people," vol. 14, 1991, pp. 38–40.

[90] N. Cherniavsky, "Activity analysis of sign language video for mobile telecommunication," Ph.D. dissertation, Computer Science and Engineering, University of Washington, 2009.

[91] J. Tran, "Power saving strategies for two-way, real-time enabled cellular phones," Master's thesis, Electrical Engineering, University of Washington, 2010.

[92] N. Cherniavsky, A. Cavender, R. Ladner, and E. Riskin, "Variable frame rate for low power mobile sign language communication," in *Assets '07: Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, 2007, pp. 163–170.

[93] H. Shim, Y. Cho, and N. Chang, "Power saving in hand-held multimedia systems using mpeg-21 digital item adaptation," in *In Proceedings of IEEE Workshop on Embedded Systems for Real-Time Multimedia*, 2004, pp. 13 – 18.

[94] N. Chang, I. Choi, and H. Shim, "DLS: dynamic backlight luminance scaling of liquid crystal display," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 8, pp. 837–846, 2004.

[95] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan, "Energy-adaptive display system designs for future mobile environments," in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, 2003, pp. 245–258.

[96] F. D. Vito, T. Ozcelebi, M. R. Civanlar, A. M. Tekalp, and J. C. D. Martin, "Per-gop bitrate adaptation for h.264 compressed video sequences," in *VLBV05*, 2005, pp. 198–206.

[97] T. Schierl and T. Wiegand, "H.264/avc rate adaptation for internet streaming," in *Packet Video Workshop*, December 2004.

[98] L. De Cicco, S. Mascolo, and V. Palmisano, "Skype video responsiveness to bandwidth variations," in *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2008, pp. 81–86.

[99] G. J. Sullivan, T. Wiegand, and K.-P. Lim, "Joint model reference encoding methods and decoding concealment methods," *JVT-N046*, Jan 2005.

[100] H.-M. Nam, K.-S. Shin, J.-Y. Jeong, H.-S. Kim, and S.-J. Ko, "Bandwidth estimation based video streaming for mobile devices over hsdpa network," *Computers in Education, International Conference on*, pp. 1–2, 2009.

[101] V. Singh, J. Ott, and I. D. D. Curcio, "Rate adaptation for conversational 3g video," in *INFOCOM'09: Proceedings of the 28th IEEE international conference on Computer Communications Workshops*, 2009, pp. 205–211.

[102] High efficiency video coding. [Online]. Available: http://www.itu.int/ITU-T/studygroups/com16/jct-vc/

[103] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *In Proceedings of Passive and Active Measurements (PAM) Workshop*, 2002, pp. 14–25.

[104] C. Dovrolis and R. Prasad. Pathrate: A measurement tool for the capacity of network paths. [Online]. Available: http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate.html

[105] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient available bandwidth estimation for network paths," in *In Passive and Active Measurement Workshop*, 2003.

[106] L. Merritt, "Speed control for the x264 encoder," 2007, private communication.

VITA

Rahul Vanam was born in Bangalore, India. He received the B.E. in electronics and communication engineering from Bangalore University, Bangalore, in 2000 and the M.S. in electrical engineering from New Mexico State University, Las Cruces, in 2005. Prior to graduate school, he worked as a Project Engineer in the DSP and Multimedia group at Wipro Technologies, Bangalore. During graduate school, he worked on several research projects in the areas of digital communication, audio and video processing and compression. He has interned at Microsoft Research, Thomson Corporate Research, and Nvidia, Inc. He was awarded the Ph.D. degree from the University of Washington in 2010.

.