H.264/MPEG-4 AVC Encoder Parameter Selection Algorithms for Complexity Distortion Tradeoff *

Rahul Vanam† Eve A. Riskin†
Richard E. Ladner§
† Department of Electrical Engineering, Box 352500,
University of Washington, Seattle, WA 98195-2500.
§ Department of Computer Science and Engineering, Box 352350,
University of Washington, Seattle, WA 98195-2350.
Email: {rahulv,riskin}@ee.washington.edu, ladner@cs.washington.edu

Abstract

The H.264 encoder has input parameters that determine the bit rate and distortion of the compressed video and the encoding complexity. A set of encoder parameters is referred to as a *parameter setting*. We previously proposed two offline algorithms for choosing H.264 encoder parameter settings that have distortion-complexity performance close to the parameter settings obtained from an exhaustive search, but take significantly fewer encodings. However they generate only a few parameter settings. If there is no available parameter settings for a given encode time, the encoder will need to use a lower complexity parameter setting resulting in a decrease in peak-signal-to-noise-ratio (PSNR). In this paper, we propose two algorithms for finding additional parameter settings over our previous algorithm and show that they improve the PSNR by up to 0.71 dB and 0.43 dB, respectively. We test both our algorithms on Linux and PocketPC platforms.

1 Introduction

The joint effort by the ITU-T's Video Coding Experts Group and the ISO/IEC's Moving Pictures Experts Group resulted in the standardization of H.264/MPEG-4 AVC [1]. This state-of-the-art standard yields a bit rate savings of about 50% over MPEG-2 for the same PSNR [2]. The compression efficiency of H.264 encoder makes it suitable for low bandwidth cellular networks, but comes with an increased cost in complexity.

Some applications that use video transmission involve video-conferencing, downloading pre-encoded videos and video-streaming. In downloading a pre-encoded video file, the encoder can be optimized to achieve the best compression and quality without considering the

*This work is supported by the National Science Foundation under grant number CCF-0514353.

encoding delay. In streaming applications, such as broadcast of a live event, a short delay in transmission is permissible. Hence, the encoder can use complex algorithms for higher compression.

Conversational services like video-conferencing require the video encoder and decoder to run in real-time. Conversational services using video cell phones are already in use in countries like Sweden and Japan, where high wireless network bandwidths are available, and this enables the Deaf in these countries to communicate in Sign Language. One of the largest challenges in this application is running the video encoder in real-time on cell phones, because they have low processing power. Choosing the right encoder parameters is important in such applications to ensure good quality while satisfying bit rate and complexity constraints. In this paper, we consider distortion-complexity optimization algorithms, where the encoder parameter settings are chosen offline using training data and stored in a table for later use.

This paper is organized as follows. In Section 2, we define our problem. We describe our algorithms in Section 3. In Section 4, we present our experiments and results. We conclude in Section 5.

2 The Problem Definition

In this paper, we use the x264 encoder, an open source implementation of the H.264 standard [3]. The x264 encoder was compared with different commercial H.264 encoders and found to provide the best quality in terms of PSNR, DCT-based video quality metric, and structural similarity metric [4]. In [5], the x264 encoder was compared with the JM reference encoder (ver 10.2) [6] and was shown to be 50 times faster while providing bit rates within 5% for the same PSNR.

We vary the following x264 parameters in our experiments: sub-pixel motion estimation (`subme`), number of reference frames (`ref`), partition size (`part`) and quantization method (`trellis`). The `subme` has 7 different options, $(1, \ldots, 7)$, that specify the number of iterations of half pixel and quarter-pixel diamond search that has to be performed on all candidate block types. Up to 16 reference frames (`ref`) can be used. The `part` has 10 different options, $(1, \ldots, 10)$, that specify the partition size allowed for intra (I), predictive (P) and bi-predictive (B) macroblocks. The `trellis` has three options, $(0, 1, 2)$, where the first option uses uniform deadzone quantizer and the other two options use a trellis for quantization. We reindex these options as $(1, 2, 3)$. For each parameter, a lower option number corresponds to a lower complexity and lower PSNR.

At a fixed bit rate, we measure a PSNR and encoding time when we encode a video with a given parameter setting. For example, each point in Figure 1 corresponds to the PSNR of the luma component (PSNR-Y) and the average encode time per frame corresponding to a parameter setting. The optimal parameter settings having the least encode time for a given PSNR can be found offline by encoding the video over all possible parameter settings and obtaining the parameter settings on the distortion-complexity convex hull. However, an exhaustive search is time consuming.

In [7], we presented two fast offline methods, based on the generalized Breiman, Friedman, Olshen, and Stone (GBFOS) algorithm [8], called the GBFOS-basic and the GBFOS-iterative algorithms. These algorithms perform joint distortion (PSNR-Y) and complexity (encode

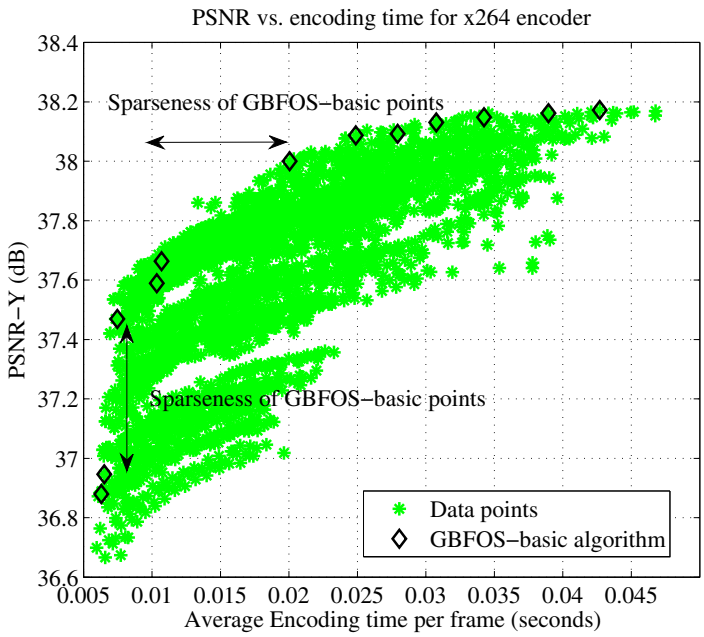


Figure 1: Distortion (PSNR) vs. complexity (average encoding time). Each data point here corresponds to a parameter setting.

time) optimization for a fixed bit rate. The algorithms chose parameter settings whose distortion-complexity (D-C) performance are close to the convex hull parameter settings. Unfortunately, both the GBFOS-basic and iterative algorithms yield only a few parameter settings. Figure 1 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 a 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 to be used for different frames. In related work to avoid time-sharing, Kiang *et al.* presented the Recursive Optimal Pruning Algorithm (ROPA) [9] for finding more tree-structured vector quantizer codebooks compared to those generated by the GBFOS algorithm [8]. 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.

3 Our Parameter Selection Algorithms

In this section, we present two methods for finding additional parameter settings for the GBFOS-basic algorithm in low and mid encode time regions, called the *controlled local search algorithm* (CLSA) and the *dominant parameter setting pruning algorithm* (DPSPA). The CLSA takes more encodings and generates more parameter settings compared to the DPSPA.

3.1 Controlled Local Search Algorithm

Our CLSA is similar to the popular local search algorithm in combinatorial optimization [10]. 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.

We explain our CLSA with an illustrative example using the x264 encoder with `subme`, `ref`, `part`, and `trellis` parameters defined in Section 2. The videos used in our experiment are described in Section 4. We first explain our notation. We use $\{1, \ldots, M\}$ to denote a set of M encoder parameters, with each parameter i having n_i options $\{1, \ldots, n_i\}$ arranged in order of decreasing mean squared error (MSE). (In our example, $M = 4$ since we consider four variable parameters. The parameters `subme`, `ref`, `part` and `trellis` have $n_i = 7$, 16, 10 and 3, respectively.) 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 $(1, 2, 1, 3)$ which specifies an H.264 encoder with the fastest sub-pixel motion estimation scheme; two reference frames; a partition size of P8 $\times$ 8; and the use of trellis algorithm for quantization and CABAC for entropy coding.

The start and end parameter settings from the GBFOS-basic algorithm are 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. 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, 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). If $\mathbf{X} = \{\mathbf{a}_1, \ldots, \mathbf{a}_i, \ldots, \mathbf{a}_K\}$ is a set of parameter settings, then a parameter setting $\mathbf{a}_i$ is dominated if there exists at least one parameter setting in $\mathbf{X}$ having lower encode time and higher PSNR. 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

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 $\mathbf{G}$

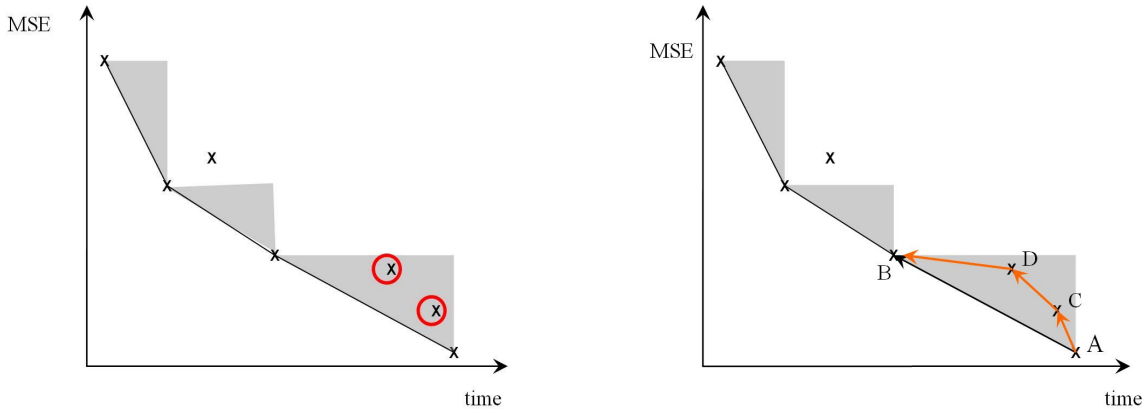
Figure 2: Controlled local search algorithm.

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 with start and end parameter settings interchanged and modifying the expansion(.) subroutine to decrease each parameter option. In our experiments, we find that the approach in Figure 2 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.

3.2 Dominant Parameter Setting Pruning Algorithm

Our DPSPA is similar to ROPA [9]. It generates parameter settings that include the GBFOS-basic algorithm parameter settings and it requires the same number of encodings as the GBFOS-basic algorithm. The DPSPA is also similar to the GBFOS-basic algorithm, except for the slope pruning stage. In the GBFOS-basic algorithm, parameter settings that are not on the convex hull of a D-C plot are not considered during the pruning stage. However, the DPSPA considers the dominant non-convex hull parameter settings, as shown by the circled



(a) (b)

Figure 3: Dominant Parameter Setting Pruning Algorithm (DPSPA). (a) The dominant non-convex hull points are indicated by the red circles. (b) Path traversed by the DPSPA (indicated by red arrows) and the GBFOS-basic algorithm (indicated by the black arrow).

points in the shaded region of Figure 3(a).

We explain our algorithm using the four variable parameters of the x264 encoder described in Section 2, and the notation of Section 3.1. We consider the MSE of the luma component as distortion $d(.)$ and average encode time as complexity $c(.)$. We denote the parameter setting generated during each iteration by $\mathbf{p}$. The algorithm is:

1. The MSE vs. average encoding time per frame plot is generated for each variable parameter by setting the other variable parameters to their highest complexity option. For example, for `subme`, we set `ref` = 16, `part` = 10, and `trellis` = 3 and vary `subme` from 1 to 7, to obtain 7 distortion-complexity points. Figures 3(a) and 3(b) illustrate these points.

2. For each parameter, find the parameter settings on the D-C convex hull and its corresponding slope. If $\mathbf{a}$ and $\mathbf{b}$ are adjacent convex hull parameter settings, then we define the slope as:

$$S(\mathbf{a}, \mathbf{b}) = abs\left(\frac{d(\mathbf{a}) - d(\mathbf{b})}{c(\mathbf{a}) - c(\mathbf{b})}\right), \tag{1}$$

   where $abs(.)$ gives the absolute value.

3. Generate the first parameter setting by selecting the lowest D-C convex hull point corresponding to each parameter. For example, if `subme` =7, `ref` = 12, `part` = 9 and `trellis` = 3 correspond to the lowest D-C convex hull points, then the first parameter setting $\mathbf{p} = (7, 12, 9, 3)$.

4. For each pair of convex hull points, find the non-convex hull points that are dominant. For example, in Figure 3(a), the circled points are dominant. In Figure 3(b), we label the convex hull points as A and B and their dominant points as C and D. The points C, D, and B correspond to `subme` = 6, 5 and 4, respectively.

5. The new parameter settings are generated in the steps below by updating the previous parameter setting $\mathbf{p}$ with the pruned parameter.

6. 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 3 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 DPSPA, we instead check if the slope contains dominant non-convex hull points. If they exist, we generate new parameter settings by pruning each point. For example in Figure 3(b), we prune AC, CD, and DB, and update the `subme` in $\mathbf{p}$ to result in three new parameter settings: $(6, 12, 9, 3)$ and $(5, 12, 9, 3)$ and $(4, 12, 9, 3)$.

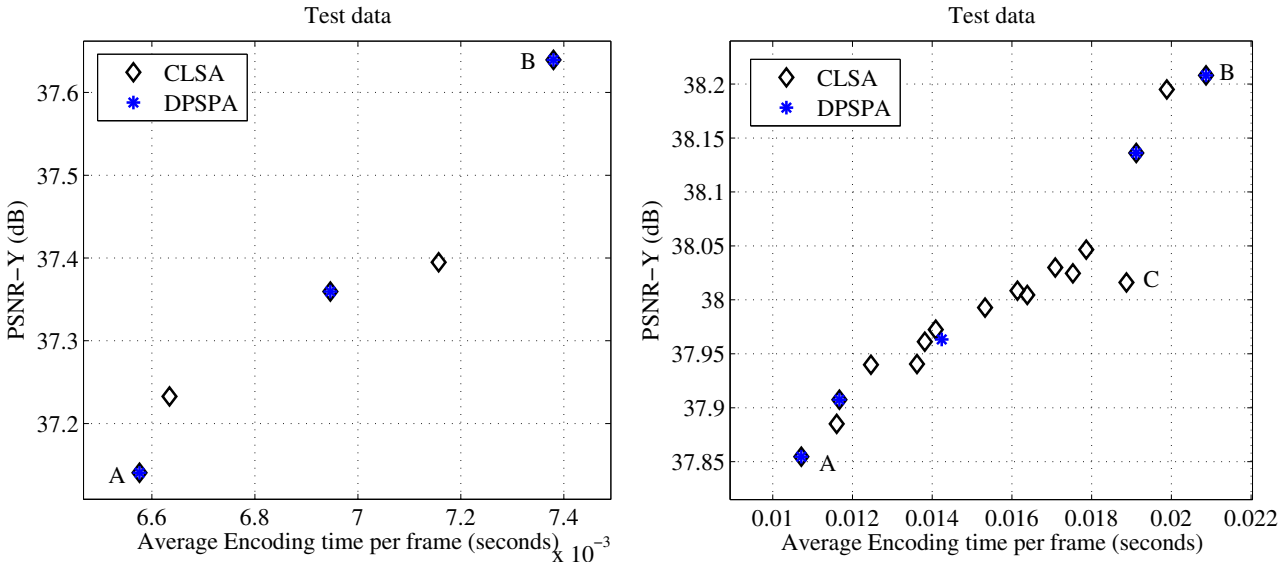
   (c) If there are no dominant non-convex hull points, we prune the convex hull corresponding to the least slope. For example in Figure 3(b), if C and D were not present then we would prune AB, resulting in $\mathbf{p} = (4, 12, 9, 3)$.

The Step 6(b) of our algorithm generates additional parameter settings over the GBFOS-basic algorithm.

4 Experiments and Results

We consider American Sign Language (ASL) videos encoded at 30 kb/s in our tests, since we are interested in lowering the complexity of the x264 encoder for encoding ASL videos for transmission over the cellular network [11]. We test our algorithms for two platforms: 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 QCIF (176×144) and 10 QVGA (320×240) ASL videos. These videos were recorded using a camera at 30 frames per second (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 consider the four variable parameters described in Section 2 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. We apply the GBFOS-basic algorithm on the 14 ASL training videos and plot the results in Figure 1. The GBFOS-basic algorithm takes only 33 encodings per video vs. 3360 encodings per video for an exhaustive search [7]. We apply both our algorithms described in Section 3 to obtain extra parameter settings in the low and mid encode time regions indicated in Figure 4. In Figure 4(b), point C performs poorly due to the mismatch between training and test data.



(a) (b)

Figure 4: 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.

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 parameter settings generated by our algorithms on the test data.

Table 1 gives results for both our algorithms for the x264 encoder on the Linux platform. Our algorithms result in a slight increase in the maximum PSNR gain on the Linux platform. For the CLSA, we use the following start and end parameter settings for low encode time region {(1,1,3,2),(3,1,3,2)}, and for mid encode time region we use {(3,5,4,2),(7,5,4,2)}, respectively. The DPSPA takes the same number of encodings as the GBFOS-basic algorithm while the CLSA takes additional encodings. Although CLSA takes more encodings, it yields more additional parameter settings.

For our tests on the cell phone, we use 10 QCIF ASL videos. Five 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 limited options. We use the `subme` options (0,1,2,3), where the option zero corresponds to integer pixel motion estimation. We use `part` options

Table 1: 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 2: 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}

(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 {(0,1,1,1),(1,1,1,1)} and {(1,1,2,1),(3,1,2,1)} as the start and end parameter setting pairs, respectively. The results for low and mid encode time regions are tabulated in Table 2. 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 few parameters having few options.

We also formulated the distortion-complexity optimization problem as a Multiple Choice Knapsack Problem [12], and tried solving it using dynamic programming. However, this approach did not yield many parameter settings.

5 Conclusion

Our previous GBFOS-basic algorithm generate few parameter settings having good distortion-complexity performance. In this paper, we propose two algorithms for finding additional parameter settings for the GBFOS-basic algorithm. The two algorithms have a tradeoff between the number of encodings and the number of additional parameter settings generated. The additional parameter settings in the low-encode time region are useful for the H.264 encoder on a cell phone, since it allows the encoder to choose a parameter setting that yields higher PSNR while satisfying the encoding speed constraint. We tested our algorithms for ASL videos on both PocketPC and Linux platforms. The additional parameter settings generated by CLSA and DPSPA improve the PSNR by up to 0.71 dB and 0.43 dB, respectively.

Acknowledgement

We thank Jaehong Chon for helping us record ASL videos on the cell phone.

References

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

[2] 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.

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

[4] *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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

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

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