X264: A HIGH PERFORMANCE H.264/AVC ENCODER

Loren Merritt et al.

1. INTRODUCTION

The joint effort by ITU-T's Video Coding Experts Group and ISO/IEC's Moving Pictures Experts Group resulted in standardization of H.264/MPEG-4 AVC in 2003 [1]. Like previous standards, H.264/AVC specifies only a decoder, therefore allowing for improvement in compression rate, quality and speed in designing the encoder. Since its standardization, there has been range of H.264 encoders developed by individuals and organizations. The encoder developed by the Joint Video Team, known as the Joint Model (JM) [3], has been used as a reference by encoder developers in enhancing existing algorithms. However, due to its speed, its use has been limited.

Another H.264 open source encoder is the x264 [4]. Its development started in 2004, and it has been used in many popular applications like ffdshow, ffmpeg and MEncoder. In a recent study, x264 showed better quality than several commercial H.264/AVC encoders [5]. In this paper, we compare the performance of the JM encoder (ver. 10.2) with x264 (ver 0.47.534) and show that x264 is about 50 times faster and provides bitrates within 5% of JM for the same PSNR.

2. X264

The high performance of x264 is attributed to its rate control, motion estimation, macroblock mode decision, quantization and frame type decision algorithms. In addition, x264 uses assembly optimized code for many of the primitive operations. In x264, there are three quantization approaches: one of them is the uniform deadzone (called Trellis-0),

and the other two use an algorithm based on H.263 trellis quantization [7], and is similar to, but developed independently from, the soft decision quantization of [8] (called Trellis-1 and Trellis-2).

2.1. Rate Control

Rate control allows selection of encoding parameters to maximize quality under the constraint imposed by specified bitrate and decoder video buffer. The rate control in H.264/AVC can be performed at three different granularities – group of pictures level, picture level and macroblocks level [a]. At each level, the rate control selects quantization parameter (QP) values, that determine the quantization of transform coefficients. Increase in QP increases quantization step size and decreases the bitrate. The rate control in x264 is based upon libavcodec's implementation [6], and is mostly empirical. There are five different rate control modes in x264 and are described below. In the hypothetical reference decoder (VBV) mode, x264 allows each macroblock to have a different QP, while in other modes QP is determined for an entire frame.

2.1.1 Two pass (2pass)

In this approach, data is obtained about each frame during a first pass, allowing x264 to allocate bits globally over the file. After application of the first pass, the 2pass approach is implemented as follows:

i. First the relative number of bits to be allocated between P-frames is selected independently of the total number of bits and it is calculated empirically by

$$bits = complexity^{0.6} \quad (1)$$

where *complexity* is the predicted bit size of a given frame at some constant QP. I- and B-frames are not analyzed separately, but rather use the QP from nearby P-frames.

ii. The results of (i) are scaled to fill the requested file-size.

iii. Encoding is performed in this step. After encoding each frame, the future QPs are updated to account for mispredictions in size (this is referred to as long-term compensation). If the second pass is consistently off from the predicted size, then

$$offset = 2^{(real\ filesize/predicted\ filesize)/6} \quad (2)$$

is added to all future QPs. Apart from long-term compensation, there is short-term compensation (using *real filesize - predicted filesize* instead of the ratio) to prevent x264 from deviating too far from the desired file size near the beginning (when there is less data for long-term compensation) and near the end (when long-term compensation does not have time to react). Short-term compensation is based on the absolute difference between encoded size and target size, rather than the ratio.

2.1.2 Average bitrate (ABR)

This is a one-pass scheme where the rate control must be done without knowledge of future frames. Therefore, ABR can not exactly achieve the target file size. The steps given below are numbered to match the corresponding steps in 2pass.

i. This is the same as in 2pass, except that instead of estimating complexity from a previous encode, we run a fast motion estimation algorithm over a half-resolution version of the frame, and use the Sum of Absolute Hadamard Transformed Differences (SATD) of the residuals as the complexity. Also, the complexity of the following GOP is unknown, so I-frame QPs are based on the past.

ii. We do not know the complexities of future frames, so we can only scale based on the past. The scaling factor is chosen to be the one that would have resulted in the desired bitrate if it had been applied to all frames so far.

iii. Long and short term compensation is the same as in 2pass. By tuning the strength of compensation, it is possible to obtain quality ranging from close to 2pass (but with file size error of ±10%) to lower quality with reasonably strict file size.

2.1.3 VBV-compliant constant bitrate (CBR)

This is a one-pass mode designed for real-time streaming.

i. It uses the same complexity estimation for computing bit size as ABR.

ii. The scaling factor used for achieving the requested file-size is based on a local average (dependent on VBV buffer size) instead of all past frames.

iii. The overflow compensation is the same algorithm as in ABR, but runs after each row of macroblocks instead of per-frame.

2.1.4 Constant rate-factor

This is a one-pass mode that is optimal if the user does not desire a specific bitrate, but specifies quality instead. It is the same as ABR, except that the scaling factor is a user defined constant and no overflow compensation is done.

2.1.5 Constant quantizer

This is a one-pass mode where QPs are simply based on whether the frame is I-, P- or B-frame.

2.2 Motion Estimation

Motion estimation (ME) is the most complex and time-consuming part of the H.264/AVC encoder. It uses multiple prediction modes, multiple reference frames. There are four different integer-pixel motion estimation methods provided by x264, namely: diamond, hexagon, uneven multihexagon (UMH) and successive elimination exhaustive search. UMH provides good speed without sacrificing significant amount of PSNR, while hexagon is a good tradeoff for higher speeds. The implementation of UMH in x264 is

similar to JM's simplified UMH, except that it includes modified early termination. The following steps describe the early termination in x264:

1. We consider up to 10 candidate motion vectors (MVs) including four from neighboring locations in the current frame (A,B,C,D) and three in the previous frame (X,Y,Z). The (0,0) MV. Finally, the median MV and (in B-frames) the temporal direct MV are computed according to the H.264 standard and considered as candidate MVs. Note that the 4 candidates from the current frame take their MVs directly from the motion search that was run on those blocks, with the same reference frame as is currently being searched; This might not be the same reference frame as was finally selected to encode the neighbors. To avoid extra memory usage, the 3 candidates from the previous frame do use the reference frame finally selected, but this is compensated by scaling those MVs inversely proportional to their temporal length.

D	B	C
A	(this frame, this MB)	

X (previous frame, this MB)	Y
Z	

Fig 1 Sources of candidate MVs for motion search.

2. One step of diamond search is applied to (0,0) and the median MV.

3. If block size is 4x4 then break from UMH and apply hexagon search. Else, continue.

4. Apply one step of diamond search to the best MV. This is labeled as ‘1’ in Fig 2.

5. Early termination: If step (4) did not find a new MV, then perform the following early termination:

i. Perform diamond search of radius two. This is labeled as '2' in Fig 2.

ii. If steps (1) or (2) found a new MV, then run a symmetric cross search (radius 7) and an octagon search (radius two). This is labeled as '3' in Fig 2.

iii. If steps (5i) and (5ii) did not find a new MV, then break.

6. Adaptive radius: Select the search range for step (7), based on the best SAD so far, and on the smoothness of the motion field. Compute the average difference between the predictors used in step (1). If there were no predictors, skip step (6). The default search range is 16. This may decrease as low as 12 if SAD is small and the predictors are similar, and may increase as high as 24 if SAD is large and the predictors differ much.

7. Run 5x5 exhaustive search, uneven cross, multi-hexagon-grid, and iterative hexagon refinement as in JM.

8. Run half-pixel precision diamond search with SAD.

9. If this is not the first reference frame being evaluated for the current block, then compare SAD so far against the best SAD of any of the previously search reference frames. If the current SAD is more than 15% worse, then skip step 10.

10. Run quarter-pixel precision with SATD0, defined as

$$SATD0 = SATD + \lambda * bits \qquad (3)$$

where SATD is the sum of absolute Hadamard-transformed difference, λ is the Lagrange multiplier, and *bits* is the number of bits needed to encode the motion vectors using Context-based Adaptive Variable Length Coding (CAVLC).

11. After mode decision, the MV is further refined with rate-distortion scores.

The above algorithm differs from JM in many ways. JM does not do steps 5, 9, or 11, and has fewer candidates in step 1. Between each candidate in step 1, JM applies a termination threshold based on the candidate's SAD, and decides whether to skip directly to diamond or hexagon search. In x264, we have not found it useful to stop before checking all the candidates.

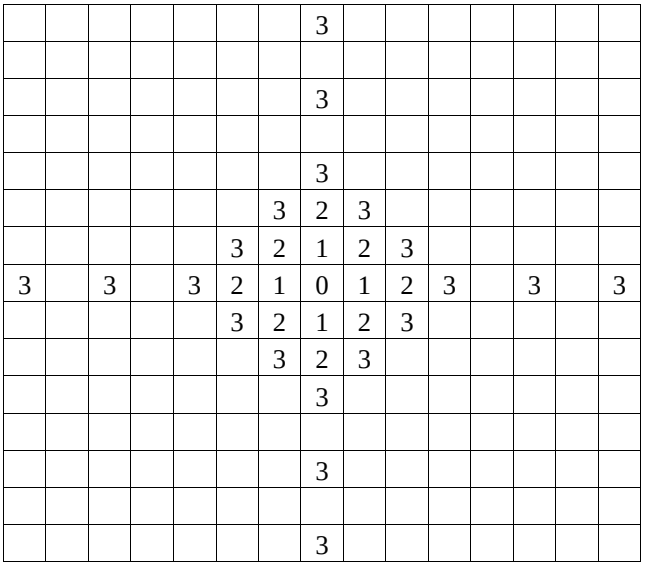

Fig 2. Search approach for early termination decision in uneven multi-hexagon search. Diamond search of radius one is labeled as '1', diamond search of radius two is labeled as '2', and symmetric cross search of radius 7 and octagon search of radius two is labeled as '3'.

2.3Mode Decision

The H.264/AVC standard allows a wide range of macroblock partitions. For inter-prediction, it allows a 16x16 macroblock to be partitioned into blocks of 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 or 4x4 pixels. For intra-predictions, the allowed partition sizes are 16x16, 8x8 and 4x4. In x264, the mode decision to choose macroblock partition is a hybrid of SATD0 and rate-distortion optimization. Two early termination conditions are used by x264 to speed up mode decision, and one for the final motion refinement (see 2.2.11).

1. In this step, we run motion estimation and compare the SATD0 scores of various macroblock modes. Based on the scores of some modes, other modes may be discarded

even without running their motion search. First we search the 16x16 partition for the immediate previous frame (P16x16ref0). If P16x16ref0's MV is equal to P-Skip's MV, and the distortion caused by leaving it with no residual is sufficiently small (see JVT-??? for the threshold), then this macroblock is assigned to be P-Skip and any further analysis is avoided. Else, we evaluate P16x16 and P8x8 with all reference frames. Then the cost of P16x8 and P8x16 are estimated as being equal to (SATD of P8x8) + 0.5*(bit cost of P8x8). If this estimate is worse than the known SATD0 cost of P16x16, then skip P16x8 and P8x16. Otherwise search them using only the reference frames chosen for the two co-located 8x8 partitions. Among P16x16, P16x8, P8x16 and P8x8, the mode having the least SATD0 cost is chosen. If P8x8 is chosen, then we search for P4x4. The decision of whether to search for P8x4 and P4x8 is made using the same approach as for P16x8 and P8x16.

Once a decision is made on the inter mode partition size, search is performed on the intra modes, with I16x16 first. The selection of intra directions is decided by SATD0. (Except that I8x8 is evaluated with the 8x8 Hadamard Transform since I8x8 uses the 8x8 DCT, whereas I16x16, I4x4, and all other invocations of SATD0 use the 4x4 Hadamard because it's faster and they mostly use the 4x4 DCT.) If none of the neighboring macroblocks used intra, and I16x16 is much worse than the best inter mode, then search for other intra modes is skipped. Otherwise search is performed for I8x8 and I4x4.

2. Now we refine the mode decision using the rate distortion (RD) cost function. We discard any mode whose SATD0 is more than a threshold above the best mode's cost computed in the previous step. This threshold is unlimited in I-frames, 25% in P-frames, and 6% in B-frames. The remaining modes are then compared using exact RD scores.

3. Now we refine the motion vector(s) or intra direction(s) of the final selected mode, using RD. In inter blocks, subpel diamond search is performed using partition size

decided in step 2, and using RD cost instead of SATD0. As with the RD mode decision, a threshold (6%) is applied to the SATD0 scores of all candidate MVs before computing the RD score for each MV. In intra blocks, the thresholds are 12% for I16x16, 37% for I8x8, and unlimited for I4x4 (because it costs more time to keep track of 4x4 SATD scores than we save in reduced RD calls).

2.4Quantization

After selecting a macroblock type and motion vector (and during each RD evaluation), the residual is computed, which is the DCT of difference between the input frame and the intra- or inter-predicted macroblock. Integer values for representing the DCT coefficients are selected and this is known as quantization. In this section, we describe quantization approach used both in JM and x264.

2.4.1 Quantization Methods

The conventional quantization approach used in most codecs is the *uniform deadzone*. This works by simple division, biased towards zero. The bias accounts for the fact that smaller coefficients take fewer bits. This is RD-optimal assuming the coefficients are independent and follow a Laplacian distribution (exponential probabilities imply that the difference between the cost of quantized DCT coefficient (*N*) and cost of *N+1* is constant).

JM includes an *adaptive deadzone,* which allows the magnitude of the Laplacian to vary over time, and vary for different frequencies (rounding being different for each position of DCT coefficient). It still assumes independence between coefficients. In practice, this differs from uniform deadzone only at very high bitrates. (At low rates, the vast majority of coefficients are zeros or ones, so the dependence between coefficients matters more than the distribution of one coefficient in isolation).

Trellis (a.k.a. *Viterbi*) quantization in x264 relaxes the assumptions that DCT coefficients are independent and have a Laplacian distribution. It optimizes for the real context-based adaptive binary arithmetic coding (CABAC) costs, including effects of other coefficients in the DCT block. That is, the CABAC context under which each coefficient is coded depends on its position in DCT, how many previous DCT coefficients are ones, and how many are greater than one. The probability distributions are taken directly from the CABAC algorithm, rather than being estimated.

There are three quantization approaches that a user can choose in x264 based upon rate-complexity tradeoffs. One of them is uniform deadzone (called Trellis-0) and the other two are based upon trellis (called Trellis-1 and Trellis-2). There is an increase in both complexity and bitrate savings, by using higher order implementations of trellis. In Trellis-1, trellis is applied only in the final encode of a macroblock, while uniform deadzone is used for the RD computations in Section 2.3. Trellis-2 uses trellis for all refinement steps in mode decision.

2.4.2 Trellis in x264

In this algorithm, quantization is performed for each coefficient of a DCT block such that the total rate-distortion cost of the block is minimized. This is similar to the algorithm discussed in [8]. The trellis can be described by a directed graph given in Fig 3. A 4x4 luma block will consists of 16 DCT coefficients, each represented by a column in Fig 3. (An 8x8 block works the same way, just with a bigger graph.) Each coefficient can be coded with one of 8 possible CABAC contexts. The contexts are determined by the number of coefficients greater than one (*NumLg1*) and equal to one (*NumEq1*). The nodes of the graphs are labeled as *NumEq1_NumLg1* [8]. *level* is the value of the quantized coefficient. Trellis is implemented as a dynamic program, where any two

states of the same (*dct_index, cabac_context*) are considered the same, and only the one with the best score is kept. The algorithm has four parts to it:

1. Comparing states based on rate-distortion cost and choosing the state with the least cost. Since DCT is a unitary transform, the RD can be computed incrementally [10].
2. Transition between states. This involves encoding a new coefficient onto one of the partial encodings. First we calculate the rate (using CABAC) and distortion of the chosen quantized level for the new coefficient. Then we update the CABAC context depending on the level coded, as specified in the H.264/AVC standard. The RD cost corresponding to the new coefficient is added to the total cost of the current state. If we end up in a state that has already been reached through a different path in the trellis, then we compare the RD costs and keep the path with the least cost.
3. The states to try. We use Dijkstra's shortest path algorithm [9].
4. Optimizations. Some coefficient values are very unlikely to occur so we can omit them from the search without measurable quality loss. We first consider the coefficient quantized with no deadzone (i.e. rounded to the nearest integer). If the quantized coefficient (*level*) equals zero, then the coefficient is chosen to be zero, and its column can be omitted from the graph in Fig 3. Otherwise, we try step 2 for both *level* and *(level-1)*. As an example, let the unquantized coefficient be 1.8. This is quantized to a *level* = 2. We try step 2, for both *level* = 2 and *(level-1)* = 1. In this case the horizontal transition (corresponding to *level* = 0) is omitted.

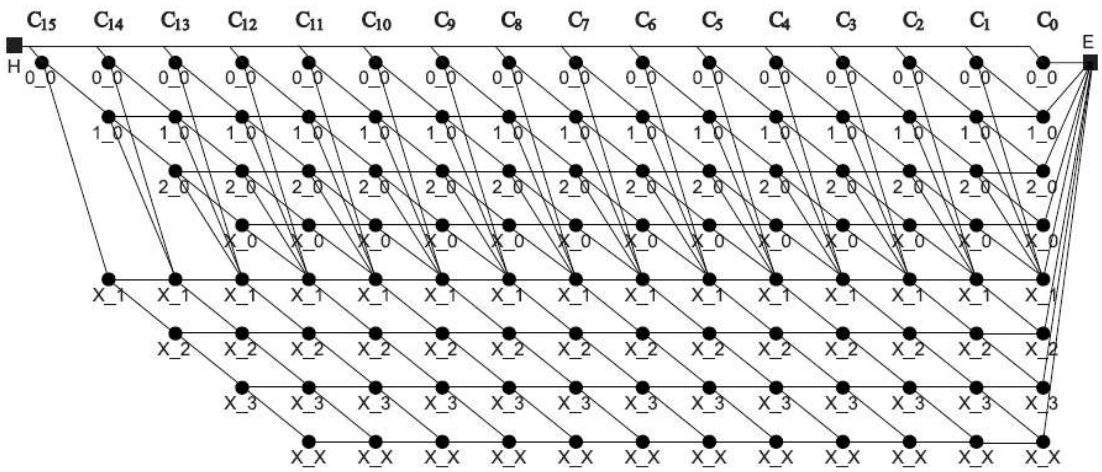

Fig 3. Graph structure for trellis quantization in x264 [8]

2.5 Frame Type Decision

We heuristically decide how many consecutive B-frames to use and where to place the I-frames, in order to maximize compression while obeying a maximum keyframe interval. A keyframe is a frame where decoding can be started without any reference to previous frames. We want to limit that interval so as to allow error resilience and efficient seeking, but try not to use too many keyframes because they take more bits than inter-frames.

To decide between zero and one B-frames, we run a fast motion estimation pass over the next two frames, once with both as P-frames and once as BP. Use whichever mode has lower total SATD0 cost, biased towards BP to account for the uneven quantization that is possible with B-frames.

If BP was better than PP, then try larger numbers of B-frames: Run the same fast motion estimation on the P-frame of BBP, BBBP, and so on (ignore the contents of the B-frames), and stop when the number of intra blocks in the P-frame exceeds a threshold. In

order to improve speed, the above search uses versions of the input frames that have been downscaled by a factor of 2x2. Note that this is the same metric used in ABR/CBR/CRF ratecontrol, so the scores can be reused.

3 Results

In this section, we compare the performance of x264 and JM ver 10.2 using 19 CIF sequences available in [11]. The number of reference frames (5) and QP in both the encoders are set to be equal, and the dissimilar parameters are chosen such that they result in the best possible PSNR. In x264, we choose Trellis-2 as the quantization approach since it provides higher bit savings. The encoders are tested for different QPs in the range 18 – 36 in steps of 3. Comparison is made based on average encoding time per frame and bitrate normalized for fixed PSNR. From Fig 2(a), we find that on an average, x264 performs 50x faster when compared to JM. In Fig 2(b), comparison is made between x264 and JM for relative bitrate vs. PSNR. It shows that x264 performs better than JM for PSNR greater than 38 dB and for PSNR below 38 dB there is slight increase in bitrate up to 5%. However, this loss in bit savings is small.

Comparison is also performed between different quantization methods of x264 discussed in Section 2.4. Trellis-0 is now considered the reference. The video sequences used earlier will also be used in this comparison. In Fig 2(d), we see that on an average Trellis-2 performs 0.7 % better than Trellis-1 and 5 % better than Trellis-0 in terms of bitrate. This improved performance of Trellis-2 comes with an increased cost in encoding time. On an average, Trellis-2 takes 27 % more time than Trellis-1 and 31% more time than Trellis-0.

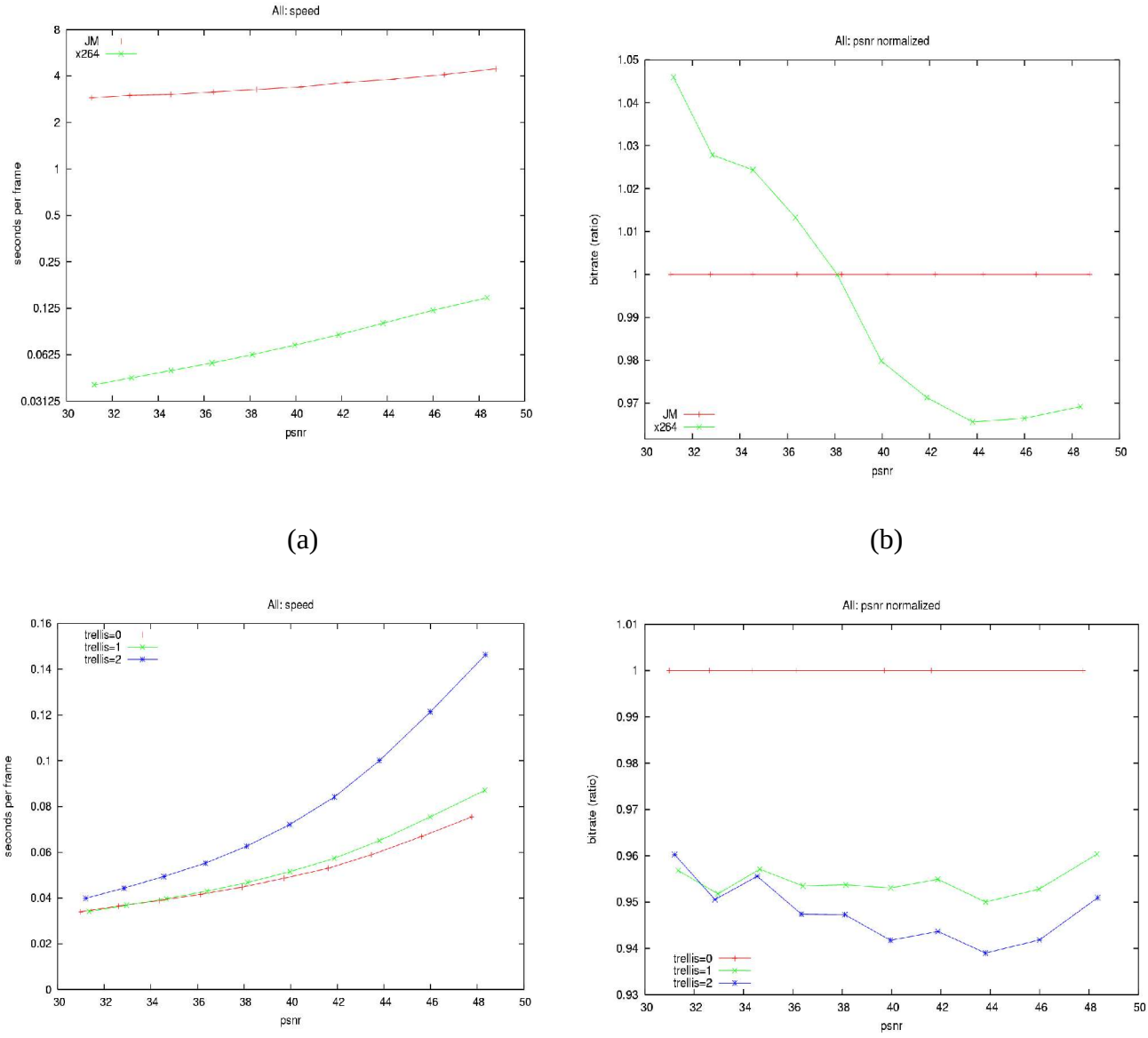

Fig 2. JM vs. x264 (a) Average encoding time per frame vs. average PSNR (dB), (b) relative bitrate vs. PSNR (dB), Comparison between trellis 0,1,and 2 of x264: (c) Average encoding time per frame vs. average PSNR (dB), (d) relative bitrate vs. PSNR (dB)

4 Conclusions

H.264/AVC is an evolving standard and there have been many new algorithms and optimization approaches being developed to improve its performance in rate, distortion and complexity. In this paper, we have presented x264, an open source H.264/AVC compliant encoder. And we have provided implementation and optimization details of some its modules such as rate control, motion estimation, mode decision, quantization and frame type decision. Comparing x264 with JM, we shown that on an average x264 is 50x faster and provides rate-distortion performance close to that of JM.

References

[1] T. Wiegand et al., "Overview of the H.264 /AVC video coding standard," *IEEE Transaction on Circuits Systems for video technology*, Vol 13, pp 560 – 576, July 2003.

[2] G. J. Sullivan, P. Topiwala, and A. Luthra "The H.264/AVC advanced video coding standard: Overview and introduction to the fidelity range extensions," *SPIE Conf. on Digital Image Processing*, Aug 2004.

[3] Joint Model reference software version 10.2, Available: http://iphome.hhi.de/suehring/tml/index.htm

[4] x264, Available: http://developers.videolan.org/x264.html.

[5] "MPEG-4 AVC/H.264 video codec comparison," CS MSU Graphics and Media Lab, December 2005

[6] ffmpeg, Available: http://ffmpeg.org/

[7] "Trellis-Based R-D Optimal Quantization in H.263+"
J.Wen, M.Luttrell, J.Villasenor, IEEE Transactions on Image Processing, Vol.9, No.8, Aug. 2000.

[8] E.-h. Yang and X. Yang, "Rate distortion optimization of H.264 with main profile compatibility," pp282 – 286, *ISIT*, July 2006.

[9] E. W. Dijkstra: *A note on two problems in connexion with graphs*. In: *Numerische Mathematik*. 1 (1959), S. 269–271.

[10] "Efficient Macroblock Coding-Mode Decision for H.264/AVC Video Coding", J.Xin, A.Vetro, H.Sun http://www.merl.com/reports/docs/TR2004-079.pdf, Dec. 2004.

[11] CIF sample videos, Available: http://www.tkn.tu-berlin.de/research/evalvid/cif.html