Selection of Long-Term Reference Frames in Dual-Frame Video Coding Using Simulated Annealing

Mayank Tiwari and Pamela C. Cosman

Abstract—**In dual-frame video coding, both encoder and decoder store a short-term reference (STR) and a long-term reference (LTR) frame for motion compensation. In past work, LTR frames at regular intervals were assigned higher quality than the other frames to improve overall video quality. In this letter, we present a method of LTR frame selection using simulated annealing, and we show that PSNR is improved compared to the case of evenly spaced LTR frames. To reduce delay and computational complexity, we consider a constraint on the size of the look-ahead window.**

Index Terms—**Dual-frame video coding, long-term reference frame, simulated annealing, video compression.**

I. Introduction

MOTION-COMPENSATED prediction is widely used for *inter-frame* video coding to remove temporal redundancy. Each block in the current frame to be encoded is typically predicted from a block in the immediate past frame (known as the reference frame) by searching for the best match block for it. In dual-frame video coding [1]–[4], one short-term reference (STR) frame and one long-term reference (LTR) frame are available for motion compensation. The LTR and STR frames are stored in both encoder and decoder. For encoding frame n, the STR is frame $n-1$ and the LTR frame is frame $n-k$, for some $k>1$. The LTR frame can be chosen by jump updating [2], in which, for example, the LTR frame remains the same for encoding N frames, then jumps forward by N frames and again remains the same for encoding the next N frames. In such an approach, every frame serves as an STR, but only every Nth frame serves as an LTR; this allows the use of high-quality LTRs which are allocated more bits than regular frames. This enhanced the quality of the entire sequence [5], [6]. In [5], the assumption was that certain frames could be starved of bits so as to generate high-quality LTR frames at regular intervals, provided that a long-term average bit rate constraint was met. This method improved the average video quality by 0.6 dB over a regular dual-frame encoder in which all frames were given equal importance. In [6], dual-frame coding was considered in a cognitive radio scenario.

Manuscript received May 22, 2007; revised October 24, 2007. This work was supported in part by the Center for Wireless Communications at UCSD and in part by the UC Discovery Grant Program of the State of California. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Philippe Salembier.

The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407 USA (e-mail: mayank@ece.ucsd.edu; pcosman@ece.ucsd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LSP.2007.914928

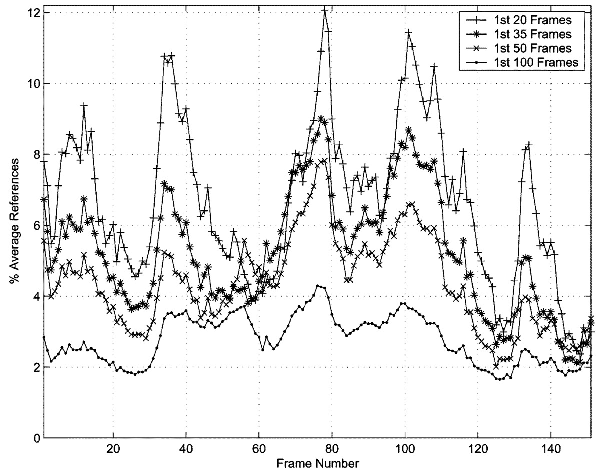

Fig. 1. Percentage average references to a frame when it is treated as a high-quality LTR frame. "1st 20 Frames" shows the effect of an LTR frame on the first 20 frames using it.

In both cases, the high-quality LTR frames were not chosen based on video content. It is possible that the frame chosen as an LTR may not serve as a good reference for future frames. The variable usefulness of LTR frames is shown in Fig. 1 for the Mother-Daughter video sequence. To generate this figure, the video was encoded repeatedly; each time, only one frame is a high-quality LTR frame. On the x-axis is the frame number of the frame that is available as a high-quality LTR frame. On the y-axis is the percentage of macroblocks (MB) of the following k frames (for $k = 20$, 35, 50, and 100) which choose to reference that LTR frame rather than use the immediate past (STR) frame for reference. The frames where we see the peaks (for example, frames 34, 35, 36, and 78) are more useful as LTRs than the frames in the valleys (for example, frames 24, 25, 26, 60, and 61). For example, the plot shows that when frame 78 is chosen to be an LTR, over 12% of the MBs of the next 20 frames prefer to reference it rather than the STR. In contrast, when frame 127 is the LTR, only 3% of MBs in the next 20 frames choose to reference it, which means 97% of the MBs find a better match in the STR. The curve for "1st 20 frames" is almost always above the curve for "1st 35 frames" which shows that the effect of the LTR frame fades as we move away from the LTR frame.

A method for LTR frame selection was studied in [7] using color layout descriptors. That paper assumes a large frame buffer at the input to the encoder and the decoder to preselect the possible LTR frames. It requires either a standard incompatible bitstream if the descriptions are sent to the decoder or an increase in complexity at the decoder to generate these descriptions.

In this letter, we aim to find a set of frames in a video sequence that can serve as good LTR frames. We use simulated annealing (SA) for the LTR frame search. We consider two scenarios: 1) when the entire video sequence is available for the LTR frame search at the encoder and 2) when there is a constraint on the size of the look-ahead window. The first scenario is solely done offline for the transmission of archived video, while the second scenario can be done in real-time assuming that the encoder is computationally efficient and that a small delay can be tolerated. This letter shows that overall video quality can be improved by proper selection of high-quality LTR frames instead of just choosing them at regular intervals. SA is one method to find such good LTR frames, but other optimization methods could be applied. We note that when the LTR frames are afforded too many bits compared to the other frames, there can be an annoying pulsing of quality that is visibly perceptible. However, the slight increase in quality for LTR frames used in this work does not lead to visible quality pulsing, and it raises the perceived quality of the entire video sequence.

This letter is organized as follows: Section II describes the SA method for finding LTR frames for archived video. Section III presents the window-based SA method for finding the LTR frame positions under the constraint of look-ahead window size. Conclusions and future work are given in Section IV.

II. Simulated Annealing for LTR Frame Search

SA [8] is an optimization process derived from the physical process of cooling molten material down to the solid state. SA has been widely used for various combinatorial and other optimization problems [9]. SA starts with an initial solution that can be generated either randomly or using some known solution. A constraint-based new solution is then generated. If the new solution is better than the current solution, it is accepted unconditionally and becomes the next current solution. If, however, the new solution is worse than the current solution, it is not rejected outright, but it is accepted with a certain probability. At the beginning, to avoid a local optimum, the probability of acceptance of a worse solution is kept high. As the simulation progresses, the probability is lowered according to some predefined schedule, and after some point, a new solution is no longer accepted unless it is better than the current solution.

We use SA for LTR frame choice in a video sequence for dual-frame video coding. If we know the video characteristics as shown in Fig. 1 for the Mother-Daughter sequence, then we can pick the peaks as our initial solution. However, since generating such characteristics is computationally intensive, we instead choose our initial solution by creating high-quality LTR frames at a uniform interval of *ltr_dist*, starting from the first frame. Evenly spaced high-quality LTR frames were used in [5] which is the reference point for comparing our results. Then one of the current set of LTR frames is randomly selected and is replaced by a new frame which is also randomly selected in the range of $\pm swing\_width$ from its original position. The new arrangement of LTR frames is accepted as the new current solution if the average PSNR of the video sequence is no less than *thr_accept* below the PSNR of the current solution. Otherwise, the new solution is rejected. We then randomly choose another LTR position from among those not yet perturbed on this round and repeat the same process. After we have gone through all *total_ltr* LTR frame positions in some random order, we have completed one iteration. After completing *num_iter* such iterations, we reduce *swing_width* by one step and *thr_accept* by ϵ amount and continue with the next round of iterations. When *thr_accept* becomes zero, we stop accepting inferior solutions. The simulation stops when *swing_width* becomes zero. The position of the high-quality LTR frames at the end of the simulation is our final solution.

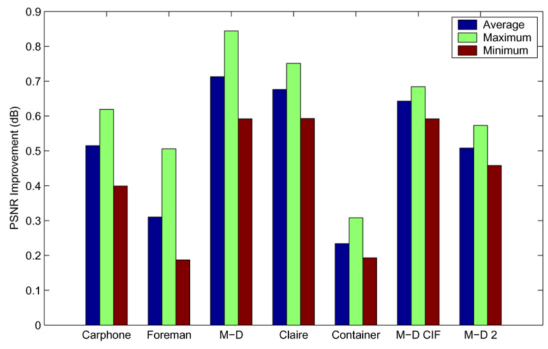

Fig. 2. Improvement over evenly spaced high-quality LTR frames using simulated annealing approach. Average PSNRs for the sequence with evenly spaced high-quality LTR frames are 33.0 dB for carphone, 29.0 dB for Foreman, 37.4 dB for M-D, 41.6 dB for Claire, 38.1 dB for Container, 33.8 dB for M-D CIF, and 40.0 dB for M-D 2 video sequence.

Results: We modified H.264/AVC [10] reference software JM 9.6, obtained from [11]. We used the 4:2:0 QCIF (176×144 pixels) video sequences Foreman, Carphone, Container, Mother-daughter (M-D), and Claire at 30 fps for our simulations. SA was performed on 200 frames with the first frame intra-coded and the remaining frames inter-coded. A lossless channel was assumed with a constant average bit rate of 58 kbps. The initial LTR frame position was chosen at a regular interval (*ltr_dist*) of 25 frames starting from the first frame. So, there were a total of 8 high-quality LTR frames ($total\_ltr$). Parameter *swing_width* was initialized to 5 and each LTR frame position was iterated $num\_iter = 4$ times for every value of *swing_width*. We initialize $thr\_accept = 0.04$, which was found empirically, as the PSNR decrease that could still be accepted. thr_accept was reduced by $\epsilon = 0.01$ whenever *swing_width* was reduced by 1.

Fig. 2 shows the results for different test video sequences. We ran six SA simulations for each video sequence. The three bars for each video sequence show the average, maximum, and minimum PSNR improvement for these six runs over the PSNR achieved by using evenly spaced LTR frames. For the Carphone video sequence, the average PSNR improvement of six SA simulations is 0.5 dB over the evenly spaced LTR frames, with the highest improvement of 0.6 dB and the lowest improvement of 0.4 dB. Best results were obtained for the M-D video sequence where the average improvement by using SA is 0.7 dB with the highest improvement of 0.8 dB and the lowest improvement of 0.6 dB. The trend of the results at CIF resolution at 58kbps for M-D video (M-D CIF) is consistent with the results for QCIF video. Similar results were also found for QCIF resolution for M-D video at 82 kbps (M-D 2). Both M-D CIF and M-D 2 are shown in Fig. 2.

As an example of the frame selection: For the Claire video sequence, evenly spaced LTR frames are 0, 25, 50, 75, 100, 125, 150, and 175. One SA run produced a PSNR gain of 0.7 dB over evenly spaced LTR frames and chose the final LTR frames 8, 32, 59, 77, 110, 122, 146, and 167. Five of the six SA runs had frames 32, 77, and 146 in their final LTR sets, suggesting that these frames are particularly useful as LTR frames. After frames 32 and 146, the video content moves very slowly. So, having these frames as high-quality LTRs improves the PSNR of subsequent frames through long-term as well as subsequent short-term references. In general, SA selects one of the first few frames of a low motion part in a video sequence and assigns it as a high-quality LTR frame. SA tries to avoid assigning a high-quality LTR in a high motion part of a video sequence because the video content changes rapidly and a high-quality LTR would not be useful for long. This is also the reason for getting higher PSNR improvement for low motion sequences such as Mother-Daughter and Claire compared to the higher motion sequences such as Foreman under the constraint of having the same number of LTR frames. In Claire, the video is constant for around 15 frames after frame 77 and then the face moves rapidly causing SA to avoid assigning new LTR frames. Therefore, the next LTR frame comes around frame 110 when the high motion part is over, resulting in a longer LTR frame distance than the average.

The container video sequence has the largest gains for evenly spaced high-quality dual-frame coding over regular quality dual-frame coding among all the video sequences tested [5]. Because of the rather constant motion between the ship and the camera, evenly spaced LTR frames do well. For an LTR spacing of 25, it showed about 0.8 dB PSNR improvement over regular quality evenly spaced LTR frames. As we can see in Fig. 2, this video sequence gives the least improvement using SA over evenly spaced LTR frames. Since there is no significant change in motion, the importance of all the frames is almost the same. When we make a plot similar to Fig. 1 for this video sequence, it produces an almost flat number of references to any frame in the video sequence. Therefore, while the evenly spaced high-quality LTR frames produce a big PSNR gain compared to evenly spaced regular-quality LTR frames, further change in LTR position will give just a small additional PSNR improvement. Conversely, while only 0.3 dB improvement was achieved for the Claire video sequence in [5] for evenly spaced high quality LTR frames with spacing of 25 compared to regular-quality dual-frame coding, we were able to achieve a further 0.7 dB of PSNR improvement on top of the evenly spaced LTR frames. In general, more than 1.0 dB PSNR improvement is achieved over evenly spaced regular-quality LTR frames in dual-frame video coding by using both high-quality LTRs [5] and uneven spacing of LTRs as discussed in this letter. The PSNR improvement is achieved with a high computational complexity which is on the order of $swing\_width \times num\_iter \times num\_ltr \times$ sequence length.

III. Window-Based Approach for LTR Frame Search

The PSNR improvement achieved in Section II assumes that the encoder has access to all 200 frames of the video sequence in advance. It is good for broadcast video where long encoding delay is possible but is not suitable for real-time or near real-time applications. For long video sequences, it requires huge memory to store the input video and also a large amount of computational resources. To overcome this problem, we propose a window-based heuristic approach to find LTR frame positions. This approach can be used for real-time video transmission with a small encoding delay.

Fig. 3 shows the average percentage of a frame that references the LTR frame (y-axis) as a function of the distance back to the LTR frame position (x-axis) for all the five video sequences. Each of the first 150 frames of each video was sequentially selected as a high-quality LTR frame and the number of references made to this LTR frame was observed over the next 100 frames (and averaged over the 150 frames), except for the frame next to the LTR frame. From this figure, we can clearly see that the

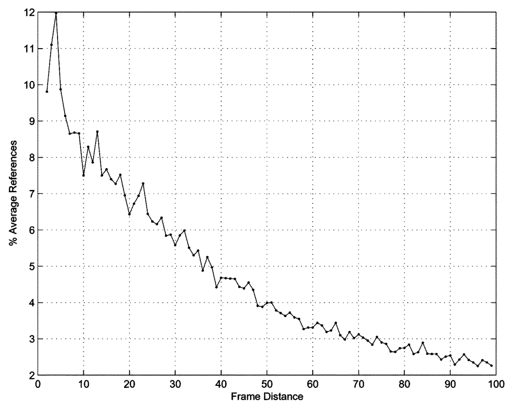

Fig. 3. Average percentage references to a high-quality LTR frame as a function of frame distance.

importance of LTR frames decreases with increasing frame distance. One approach would therefore be to assign frequent LTR frames. However, we must limit the number of LTR frames (because each one requires more bits than a typical frame).

We define the average LTR distance (*avg_ltr_dist*) at any given time to be the ratio of the number of frames remaining to be encoded to the number of LTR frames remaining to be created. Initially, *avg_ltr_dist* is 25 (200 frames to be encoded using 8 LTR frames). Experimentally, for *avg_ltr_dist* of 25 frames, we found that keeping an LTR frame for at least 15 frames provides a good quality. We denote this distance as *min_ltr_dist*. We keep an LTR frame for a minimum of *min_ltr_dist* frames, and after that, we begin to look for the next LTR frame position. Therefore, the lower boundary for the current LTR frame search (frame number $ltrL_B$) is *min_ltr_dist* from the previous LTR frame position. We recalculate *avg_ltr_dist* after choosing each LTR frame position. In general, we set *min_ltr_dist* = max(*avg_ltr_dist* −10, 0), so that *min_ltr_dist* increases if *avg_ltr_dist* increases (that means we are getting frequent LTR frames) and vice versa. This reduces the chances of getting LTRs too close to each other. The next LTR frame position is initialized at *avg_ltr_dist* from the previous LTR frame position and its frame number is *init_ltr_loc*. We search for the LTR frame position starting from frame $ltrL_B$), keeping frame *init_ltr_loc* in the middle of the search range by extending the search range to the same number of frames ($X = init\_ltr\_loc - ltrL_B$) on the other side of frame *init_ltr_loc*. We denote the upper boundary of the search range as $ltrU'_B = init\_ltr\_loc + (init\_ltr\_loc - ltrL_B)$.

However, the upper boundary of the search range is also dictated by the size of the look-ahead window. We want to create an LTR frame five frames or more back from the end of the look-ahead window so that we have at least five frames over which to judge whether or not it is a useful LTR frame. If W is the size of the look-ahead window and it starts from frame $ltrL_B$, then the upper boundary is restricted to frame $ltrU''_B = ltrL_B + W - 5$. Therefore, the upper boundary of the LTR search range is given by $ltrU_B = min(ltrU'_B, ltrU''_B)$. Fig. 4 depicts both the cases of LTR frame search range where (a) $ltrU'_B < ltrU''_B$, which means that the search range is not restricted by the size of the look-ahead window, and (b) $ltrU'_B > ltrU''_B$ when the upper boundary of the search range is restricted by the size of the look-ahead window.

The process of searching for one LTR frame in the specified range is done using SA as described in Section II. Once

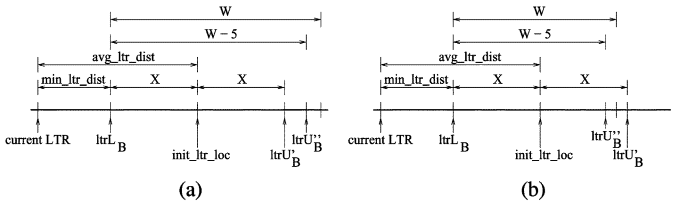

Fig. 4. LTR search range (a) $ltrU'_B < ltrU''_B$, (b) $ltrU'_B > ltrU''_B$, where $ltrU'_B = init\_ltr\_loc + (init\_ltr\_loc - ltrL_B), ltrU''_B = ltrL_B + W - 5$, and $X = init\_ltr\_loc - ltrL_B$. $ltrU_B = min(ltrU'_B, ltrU''_B)$.

an LTR frame is found, we recalculate the *avg_ltr_dist*, $ltrL_B$, and $ltrU_B$. We then move on to find the next LTR frame position using the same approach. We repeat this process until the end of the video sequence.

Results: Let W be the number of frames in the look-ahead window, meaning that these frames are assumed to be available at the encoder and are not yet encoded. After determining one LTR frame location, for computing the next LTR frame location, the look-ahead window starts at frame $ltrL_B$ and extends to frame $ltrL_B + W - 1$. All the frames before frame $ltrL_B$ are assumed to have already been encoded. Frame *init_ltr_loc* is first selected as an LTR frame to calculate the PSNR for all W frames in the look-ahead window. Then the same SA procedure is applied to select the best LTR frame in the search range, where the search range is between $ltrL_B$ and $ltrU_B$ in these W frames as described above. Once an LTR frame is selected, we calculate the new *avg_ltr_dist*, *init_ltr_loc*, $ltrL_B$, and $ltrU_B$ and repeat the procedure. Since the LTR search range is quite small compared to the range in Section II, we initialize the *swing_width* to 4 and *num_iter* to 2, thereby further reducing the complexity. Assuming the same variation of PSNR by changing an LTR frame position, we initialize $thr\_accept = \frac{8}{num\_frm}$, where *num_frm* is the number of frames in the LTR search range $(ltrU_B - ltrL_B + 1)$, and it is reduced to 0 in *swing_width* steps $(\epsilon = 0.25 \times thr\_accept)$.

Fig. 5 shows the average PSNR improvement for various test video sequences over evenly spaced LTR frames as a function of W which was varied from 20 to 40 in steps of five frames. The PSNR improvement was averaged over eight simulations for each W in a video sequence and compared with the results from Section II, shown here as "Full SA." As discussed in the previous section, we found that the improvement for the Container video sequence remains almost flat for various window sizes and is very close to "Full SA." Claire and Carphone have ample choices for LTR frames, and so these videos are also insensitive to the look-ahead window size. Fig. 1 for the Mother-Daughter video sequence shows narrow peaks and, for small look-ahead window sizes, sometimes we miss these peaks for LTR selection. In general, even with a small look-ahead window, we achieve significant PSNR improvement over evenly spaced LTR frames. The computational complexity is on the order of $swing\_width \times num\_iter \times num\_ltr \times (ltrU_B - ltrL_B)$. With the reduction in *swing_width*, *num_iter*, and LTR search range, the computational complexity in the window-based approach for finding LTR frames is drastically reduced compared to the full LTR search and is feasible for real-time.

IV. Conclusion

We used simulated annealing to find good locations for high-quality LTR frames. The experimental results show PSNR improvement of 0.2 to 0.7 dB for various test video sequences over evenly spaced high-quality LTR frames. On combining our results with [5], more than 1.0 dB PSNR improvement was achieved over video encoding using regular quality evenly spaced LTR frames. The process of LTR frame selection was further performed on a constrained look-ahead window size in a long video sequence for real-time video transmission which reduced delay and computational complexity. For most of the video sequences, the PSNR improvement in this case was close to the PSNR improvement when the whole video sequence was considered. In both cases, changing the parameters such as the bit rate (50 to 100 kbps), length (100 to 300 frames), resolution (QCIF and CIF), or number of LTR frames (5 to 8) produces similar results.

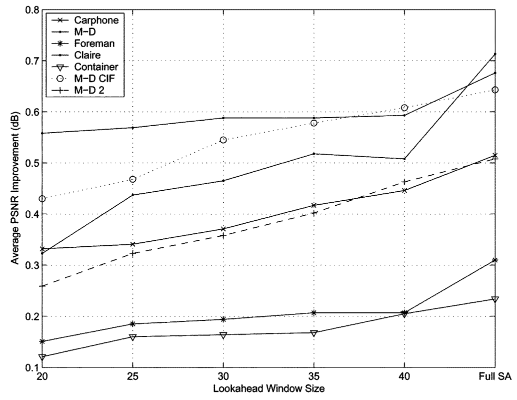

Fig. 5. Variation of PSNR improvement over evenly spaced LTR frames as a function of the look-ahead window size.

References

[1] M. Gothe and J. Vaisey, "Improving motion compensation using multiple temporal frames," in *Proc. IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, May 1993, vol. 1, pp. 157–160.

[2] T. Fukuhara, K. Asai, and T. Murakami, "Very low bit-rate video coding with block partitioning and adaptive selection of two time-differential frame memories," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 1, pp. 212–220, Feb. 1997.

[3] T. Wiegand, X. Zhang, and B. Girod, "Long-term memory motion-compensated prediction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 1, pp. 70–84, Feb. 1999.

[4] A. Leontaris and P. Cosman, "Video compression for lossy packet networks with mode switching and a dual-frame buffer," *IEEE Trans. Image Process.*, vol. 13, no. 7, pp. 885–897, Jul. 2004.

[5] V. Chellappa, P. Cosman, and G. Voelker, "Dual frame motion compensation with uneven quality assignment," in *Proc. IEEE Data Compression Conf.*, Mar. 2004, pp. 262–271.

[6] M. Tiwari and P. Cosman, "Dual frame video coding with pulsed quality and a lookahead buffer," in *Proc. IEEE Data Compression Conf.*, Mar. 2006, pp. 372–381.

[7] J. Ruiz-Hidalgo and P. Salembier, "On the use of indexing metadata to improve the efficiency of video compression," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 3, pp. 410–419, Mar. 2006.

[8] S. Kirkpatrick, C. Gelatt, and M. Vechhi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[9] P. Laarhoven and E. Aarts, "Simulated annealing: Theory and applications," in *Mathematics and its Applications*. Dordrecht, The Netherlands: Reidel, 1987.

[10] T. Wiegand, G. Sullivan, G. Bjontegaard, 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, Jul. 2003.

[11] H.264/AVC Reference Software. [Online]. Available: http://iphome.hhi.de/suehring/tml/.