An efficient motion vector composition algorithm for fast-forward playback in a video streaming system

Chang-Hong Fu, Tsz-Kwan Lee, Yui-Lam Chan*, Wan-Chi Siu

Centre for Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

A R T I C L E   I N F O

Article history:
Received 24 March 2010
Accepted 1 September 2010
Available online 9 September 2010

Keywords:
Digital video browsing
Digital video cassette recording
Fast-forward playback
Frame-skipping video transcoding
Forward dominant vector selection
Motion re-estimation
Digital video coding
Motion vector composition

A B S T R A C T

Fast-forward playback enables viewers to scan through the video scene of interest efficiently. One approach to realize fast-forward playback is to employ a frame-skipping transcoder which transcodes only the frames required for playback at the desired fast speed. Various motion vector composition algorithms are used to compose the new motion vectors with reduced complexity. These algorithms do not work well for dropping a large number of frames, which is very common in fast-forward playback. In this paper, a new multiple-candidate vector selection algorithm (MCVS) is proposed to select a composed motion vector from a set of candidate motion vectors, which utilizes relevant areas in the target macroblock to ensure a reliable tracking process for motion vector composition. Experimental results show that the proposed MCVS can provide fast-forward playback through video transcoding with significant gain, in terms of rate-distortion performance, especially when a large speed-up factor is required.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

Fast-forward and fast-backward operations are the key functions that enable quick browsing of video in a video streaming system. However, motion-compensated prediction adopted in MPEG-1/2/4 [1–3] and H.26× series [4,5] is mainly designed and best suited for normal playback in which frames are decoded and played back in a pre-determined frame order. The frame dependency severely complicates the fast-forward/backward operations since a predicted frame can only be decoded when all its reference frames have already been decoded and it requires the network to send all these related frames in addition to the actually requested frame [6–16]. Consequently, a much higher rate, which can be many times of that required by the normal playback, is necessary for fast video playback. Besides, it also requires high computational burden in a client decoder to decode all these extra frames. One simple approach to implement the fast-forward/backward playback is just to send and decode I-frames. However, if a video application browses an encoded bitstream with a very large GOP size, or needs high-precision in video-frame access, sending only I-frames may not be acceptable.

Some approaches have been proposed in the literature for the realization of fast playback [6–13]. In [6], fast-forward/backward operations were implemented at the client side by using pre-fetched video frames. However, this approach requires a huge buffer in the client machine, which is not desirable. Omoigui et al. [7] suggested a number of client-server time-compression solutions for a fast-forward playback and video browsing. By using time-compression, a server stores multiple pre-encoded video bitstreams with various temporal resolutions and transmits a video bitstream with appropriate temporal resolution according to the speed-up factor required by users. This approach does not induce excessive buffers in the client but the speed-up granularity is limited by the number of pre-stored video bitstreams. In [8–10], various dual-bitstream schemes were proposed to store a forward-encoded bitstream as well as a backward-encoded bitstream in the server. The generation of the backward-encoded bitstream can simply be done by encoding the video sequence in reverse order [8]. Fast-forward or fast-backward playback of the video is then achieved by selecting the required frames from these two encoded bitstreams with the minimum number of frames to be decoded or transmitted. Nevertheless, this approach approximately doubles the storage requirement of the server, and more than necessary frames are to be transmitted over the network and processed by the decoder. In [11], an approach to rearrange an ordinary GOP structure into a hierarchical GOP structure was further proposed. As shown in Fig. 1, an I-frame is located in the centre of the hierarchical GOP structure, and this new GOP structure is very effective for fast-forward and fast-backward operations. Nevertheless, the prediction distance between the P-frames and the I-frame inside the GOP increases. As a result, this hierarchical GOP structure...
sacrifices the performance of normal playback in order to enhance the browsing operations. To avoid the additional storage requirement in the server and computational burden in the client decoder, a transcoding approach for fast-forward playback was proposed in [12, 13]. This transcoding approach fully decodes the video in the server, selects the frames required for fast playback at a desired speed, re-encodes these selected frames, and sends the transcoded bitstream to the client decoder for decoding and display. The re-encoding process introduces a high computational complexity due to motion vector composition. Hence, the authors in [12, 13] also utilized some motion vector composition methods in frame-skipping transcoding [17–22] to expedite the re-encoding process. Among various motion vector composition algorithms, the forward dominant vector selection (FDVS) [19] and its enhanced version (E-FDVS) [22] are suggested to be the best in fast-forward transcoding [12]. However, our work shows that the FDVS/E-FDVS does not work well for a large speed-up factor of the fast video playback since the composition of new motion vectors may not represent the current macroblock (MB) anymore. In this case, the performance of the transcoded video deteriorates.

In this paper, we propose a more reliable algorithm to compose new motion vectors with a large speed-up factor. The motion vector composition is based on the relevant area of the current MB and can track several possible candidates related to the current MB throughout the transcoding process for fast-forward playback. Then, the best one among these multiple candidates is chosen. The organization of this paper is as follows. In Section 2, we discuss the impacts of a large speed-up factor on the performance of FDVS. Section 3 describes the proposed motion vector composition algorithm for realizing fast-forward playback of a pre-encoded video through frame-skipping transcoding. Simulation results are then presented in Section 4. Finally, some concluding remarks are provided in Section 5.

2. Fast-forward transcoding using FDVS

The video streaming system proposed in [12] transcodes only the frames required for playback at a desired fast speed, and employs a number of fast motion vector composition algorithms adopted in frame-skipping transcoders to estimate the motion vectors required for transcoding a video. Fig. 2 shows the fast-forward playback operation with a speed-up factor of 3. Since the video is to be played back at three times of the normal playback, only one out of three frames requires to be selected for display. To perform fast-forward playback via transcoding, frame 3 and frame 6 are re-encoded with the references from frame 0 and frame 3, respectively. In this situation, new motion vectors, which can be obtained by applying full-scale full-search motion estimation, are needed for the transcoded video. Instead of full-scale full-search motion estimation, good approximation can be obtained by re-using the existing motion vectors from the pre-encoded video. This can greatly reduce the computational burden. The transcoding approach for fast-forward playback in [12] implemented four different motion vector composition algorithms – “in-place”, “area-weighted average”, “median”, and “forward dominant vector selection (FDVS)”. Among these algorithms, FDVS has the best performance.

Fig. 3 shows an example illustrating how to compose the new motion vectors by FDVS. In this example, the fast-forward speed-up factor is 3 where frame \( n - 1 \) and frame \( n - 2 \) are skipped for fast playback. We assume that MB\(_n\) represents the \( k \)-th MB in frame \( n \) with the most corresponding motion vector \( m_{n,k}^{v,1} \), which uses frame \( n - 1 \) as the reference. Only four MBs are shown in each frame. When frame \( n - 1 \) and frame \( n - 2 \) are skipped for fast-forward playback, \( m_{n,k}^{v,1} \) and \( m_{n,k}^{v,2} \) become not valid since they point to the dropped frame, frame \( n - 1 \), that does not exist in the transcoded bitstream. It is necessary to find the new motion vector of MB\(_n\) with frame \( n - 3 \) as the reference frame, i.e. \( m_{n,k}^{v,3} \). For each MB, the FDVS scheme selects one dominant motion vector from the four overlapped MBs in frame \( n - 1 \). A dominant motion vector is defined as the motion vector carried by a dominant MB. The dominant MB is the MB that has the largest overlapping segment with the motion-compensated MB of MB\(_n\) in frame \( n - 1 \). Considering MB\(_n\) in the example of Fig. 3(a), the motion-compensated MB of MB\(_1\) overlaps with four MBs, MB\(_{1,2n}^{v,1}\), MB\(_{1,3n}^{v,1}\), MB\(_{1,2n}^{v,2}\), and MB\(_{1,3n}^{v,2}\), in frame \( n - 1 \). In the first step of FDVS, MB\(_{1,2n}^{v,1}\) is chosen as the dominant MB since it has the largest overlapping segment with the motion-compensated MB of MB\(_n\), while its motion vector \( m_{n,k}^{v,1} \) is selected as the dominant motion vector. This dominant motion vector selection process is repeated until the non-skipped frame is available, i.e. frame \( n - 3 \) in this example. Therefore, in the second step of FDVS, \( m_{n,k}^{v,2} \) is selected as the dominant motion vector in step 1. It is used to further compose the final motion vector. In this example, the motion-compensated MB of MB\(_{1,3n}^{v,2}\) points to \( m_{n,k}^{v,3} \) overlaps with four MBs in frame \( n - 2 \). Among them, MB\(_{1,2n}^{v,3}\) has the largest overlapping segment. It is the dominant MB and its motion vector \( m_{n,k}^{v,3} \) is chosen as the dominant motion vector in this stage. The resultant motion vector of MB\(_1\), \( m_{n,k}^{v,3} \), pointing to the MB in frame \( n - 3 \) is composed by summing up the selected dominant vectors and can be written as:

\[
m_{n,k}^{v,3} = m_{n,k}^{v,1} + m_{n,k-1}^{v,2} + m_{n,k-2}^{v,3}
\]  

The idea of FDVS is to find the most correlated MBs in the skipped frames for the current MB and then use the motion vectors of these most correlated MBs to build the linkage across the skipped frames. It can provide promising results for motion vector
composition in frame-skipping transcoding and becomes the most popular algorithm in comparison with other existing algorithms [12,19]. However, FDVS does not work well for consecutively dropping a large number of frames, which is very common in fast-forward playback. Fig. 4 shows the percentage increase in the transcoded video bitrate of using FDVS as compared with the full-search motion estimation with respect to different speed-up factors in fast-forward transcoding. The testing video stream for simulation is the ”Coastguard” sequence. This figure illustrates that the bitrate requirement of FDVS surges as the number of skipped frames increases. This phenomenon can be explained by Fig. 3(b), which is redrawn from Fig. 3(a). In the first step of FDVS, MB_{2n} is selected to be the dominant MB. Then, as shown in the second step of FDVS in Fig. 3(b), the whole MB_{2n} and the corresponding m_{2n} are used to determine the dominant MB in frame n – 2. From Fig. 3(b), it is observed that only the shaded area of MB_{2n} is actually related to the target MB. However, the non-shaded area in MB_{2n} under FDVS also contributes to the computation of the dominant MB in frame n – 2, but it is irrelevant to the target MB. This irrelevant information brings adverse influence to the subsequent process in FDVS. For instance, if frame n – 2 is also dropped and the whole MB_{2n} is used for motion vector composition in FDVS, then the relevant area for determining the dominant MB in frame n – 3 is the cross-hatch shaded area in MB_{2n}. As shown in Fig. 3(b), the cross-hatch shaded area only occupies a very minor portion of MB_{2n}. In this case, a large irrelevant area to the target MB is used to decide the dominant MB in frame n – 3, which seriously affects the accuracy of motion vector composition. It is noted that the relevant area of MB_{n} further diminishes for more skipped frames.

3. Proposed multiple-candidate vector selection algorithm

From the above observation, we set two rules for our proposed algorithm: (1) the area relevant to the target MB should be taken in account for dominant MB selection, and (2) the area relevant to the target MB should be kept as large as possible during motion vector composition. The rationales behind the two rules are explained in the following subsections.

3.1. Only utilizing the relevant area from the target MB

By taking the first rule into consideration, Fig. 5 shows an improvement mechanism for FDVS. When MB_{2n} is chosen as the dominant MB in the first step of FDVS, only the shaded area with
slash lines, which is the relevant region to the target MB (MB\(^1\)), instead of the whole MB\(^2\) is used to decide to the next dominant MB in frame \(n - 2\). Notice that MB\(^2\) is selected contrasts with the selection of the original FDVS as shown in Fig. 3(a) where MB\(^2\) is picked. Again, when frame \(n - 2\) is also dropped, only the cross-hatch shaded area in frame \(n - 2\) of Fig. 5 is used to determine the next dominant area in frame \(n - 3\). This mechanism can ensure only relevant area of MB\(^1\) is employed in motion vector composition. From Fig. 5, the resultant motion vector, \(m v\)\(^1\) \(n\) \(-3\) is different from the result obtained by using FDVS in (1), and can be written as

\[
m v\)\(^1\) \(n\) \(-3\) = m v\)\(^1\) \(n\) \(-2\) + m v\)\(^1\) \(n\) \(-1\) \(\times 2\) + m v\)\(^1\) \(n\) \(-1\) \(\times 2\) \(\times 3\) \(\times 4\)
\]

(2)

3.2. Maximizing the relevant area

The objective of the second rule is to maximize the relevant area for dominant MB selection. Other non-dominant areas in the skipped frames, but relevant to MB\(^1\), can also be utilized to enhance the usage of relevant area in MB\(^1\). Considering the case that the largest overlapping segment could not be dominant enough, as shown in Fig. 6(a), the largest overlapping segment with the motion-compensated MB of MB\(^1\) is very close to the second largest one. Since only the relevant region to MB\(^1\) is employed in motion vector composition, the relevant area may diminish after each skipped frame. In the example shown in Fig. 6(a), the cross-hatch shaded area in frame \(n - 2\) for selecting the next dominant MB becomes very small. As a result, it decreases the reliability of the resultant motion vector.

3.2.1. Merging process within homogeneous area

To fully utilize the relevant area in MB\(^1\), the proposed algorithm also considers the homogeneity of motion vectors, which is essential to enlarge the relevant area for motion vector composition. We again use the example in Fig. 6(a), but \(m v\)\(^2\) \(n\) \(-1\) \(-2\) is now equal to \(m v\)\(^2\) \(n\) \(-1\) \(-2\) as shown in Fig. 6(b). In this case, the shaded area overlapped with MB\(^1\) and MB\(^2\) could be combined together, and this merging area is used for deciding the next dominant MB in frame \(n - 2\). From Fig. 6(b), the selected MB in frame \(n - 2\) is MB\(^4\) where the area relevant to MB\(^1\) is larger as compared with the case of Fig. 6(a). This larger area is more reliable to determine the dominant MB in frame \(n - 3\). This merging process is appropriate for areas with homogeneous motion and it is particularly true for macroblocks in the background and inside the moving objects.

3.2.2. Multiple candidates across object boundaries

At the object boundary of a video object, no homogeneous motion field exists. We suggest using more than one candidate MBs in order to expand the area relevant to the target MB in motion vector composition. In the following, we propose to use multiple-candidate MBs for each skipped frame. Assume that \(C\)\(^n\) is the \(i\)th candidate in frame \(n - 1\) sorted by the area of the overlapping segment. In Fig. 6(c), two-candidate MBs are used to compose the motion vector for each step. In frame \(n - 1\), \(C\)\(^n\) and \(C\)\(^n\) are the largest and second largest overlapping segments with the motion-compensated MB of MB\(^1\), respectively. Therefore, both MB\(^1\) and MB\(^4\) are used to determine the next dominant MBs in frame \(n - 2\). It is noted that both of the shaded areas in MB\(^2\) and MB\(^1\) are relevant to MB\(^1\) and taken into consideration for determining the next dominant MB. From the top diagram of Fig. 6(c), four candidates (\(C\)\(^n\), \(C\)\(^n\), \(C\)\(^n\), and \(C\)\(^n\)) due to the motion-compensated segment of \(C\)\(^n\) are considered for the next step. In addition, one candidate (\(C\)\(^n\)) contributed from the motion-compensated segment of \(C\)\(^n\) is regarded as one of the possible candidates in the next step, as depicted in the bottom diagram of Fig. 6(c). Since two candidates are used for each step, from Fig. 6(c), \(C\)\(^n\) and \(C\)\(^n\) are chosen as the largest and second largest overlapping segments, and their corresponding MBs are MB\(^1\) and MB\(^2\), respectively. In Fig. 6(c), the top diagram shows the same procedure of motion vector composition as illustrated in Fig. 6(a). On the other hand, the bottom diagram gives an alternative path to compose the new motion vector, which uses the second largest candidate MB beside the largest candidate MB in frame \(n - 1\). From Fig. 6(c), we observe that the cross-hatch shaded area in frame \(n - 2\) of the bottom diagram, which is relevant to MB\(^1\), and is used to decide the candidate MB in frame \(n - 3\), is larger than that of the top diagram. In other words, even though \(C\)\(^n\) represents the largest overlapping segment in the first skipped frame (frame \(n - 1\)), it cannot ensure that it is still the largest overlapping segment in the next skipped frame (frame \(n - 2\)). The use of multiple-candidate MBs for each skipped frame can increase the possibility of keeping the MBs with the large relevant area to the target MB during motion vector composition.

3.3. The detailed flow of the proposed algorithm

In summary, the new motion vector is composed via the following steps. For the sake of simplicity, but without loss of generality, two candidate MBs are picked for each skipped frame and two skipped frames (frame \(n - 1\) and frame \(n - 2\)) are assumed.

1. For each MB in the current frame, say MB\(^1\), its motion-compensated MB in frame \(n - 1\) is divided into four segments which overlap four MBs (MB\(^1\), MB\(^2\), MB\(^3\), and MB\(^4\)), as represented by the four shaded segments in Fig. 7(a).

2. The segments with the same motion vectors are merged together so as to maximize the relevant areas across skipped frames, as shown in Fig. 7(b). All merged and non-merged segments are labeled as \(C\)\(^i\), where \(i = 1, 2, \ldots, k\) and \(k\) represents the total number of segments, and are sorted according to the shaded area. In Fig. 7(b), the merging results in three segments, namely \(C\)\(^n\), \(C\)\(^n\), and \(C\)\(^n\).
The largest and second largest shaded segments are selected. In other words, $C_{n-1}^1$ and $C_{n-1}^2$ are determined as the possible candidates and their corresponding motion vectors sum up with $mv_{n-1}^4$ to compose two new motion vectors between frame $n$ and frame $n-2$, $mv_{n-1}^1(C_{n-1}^1)$ and $mv_{n-1}^2(C_{n-1}^2)$. In the example shown in Fig. 7(b), the possible candidate vectors from $C_{n-1}^1$ and $C_{n-1}^2$ are $mv_{n-1}^3$ and $mv_{n-1}^4$, respectively. $mv_{n-1}^1(C_{n-1}^1)$ and $mv_{n-1}^2(C_{n-1}^2)$ are given by

![Diagram](attachment:image.png)
Fig. 7. The detailed steps of the proposed multiple-candidate algorithm in the case of two-skipped frames and two-candidate MBs picked for each skipped frame. (a) Merging process, (b) selected two-candidate MBs in frame \( n - 1 \), and (c) selected two-candidate MBs in frame \( n - 2 \).
for each skipped frame. In our simulations, C was set to 2 and 4.

4.1. Comparison of rate-distortion performance

When more frames are skipped, a larger number of possible candidates is necessary to be kept. Besides, for more skipped frames, step 4 is repeated until it goes through all skipped frames. Note that the number of candidates can be selected by the user according to the speed-up factor and the desired video quality.

4. Simulation results

Simulations have been performed to evaluate the overall efficiency of various motion vector composition algorithms in fast-forward playback. In all the simulations presented in this paper, the H.264 reference codec (JM9.2) [23] was employed to pre-encode the test sequences of CIF format (352 × 288 pixels), including “Salesman”, “Foreman”, “Mobile”, “Coastguard”, and “Tempete”, at 30 frames/s with a fixed quantization parameter. For all pre-encoded videos, the first frame was encoded as I-frame, while the remaining frames were encoded as P-frames in which a full-search motion estimation algorithm with a search window of ±16 pixels was used to determine the motion vectors in the pre-encoded videos. All the test sequences have a length of 200 frames. The pre-encoded videos were then transcoded to various fast-forward videos with speed-up factors of 3, 5, 7, and 9. All of the picture types and quantization parameter were preserved during transcoding.

4.1. Comparison of rate-distortion performance

For comparison, the full-search motion estimation algorithm (FS), the forward dominant vector selection algorithm (FDVS) [19], the enhanced version of FDVS (E-FDVS) [22], and the proposed multiple-candidate vector selection algorithm (MCVS-C) were used to obtain the motion vectors of the transcoded videos. In MCVS-C, C represents the number of candidate MBs selected for each skipped frame. In our simulations, C was set to 2 and 4 represented by MCVS-2 and MCVS-4, respectively. Table 1 shows the total bits generated (in kbits) and average PSNR obtained (in dB) by different motion vector composition algorithms for transcoding the “Coastguard” sequence with speed-up factors of 3, 5, 7, and 9. The PSNR result for each transcoded frame is computed by different motion vector composition algorithms with speed-up factors ranging from 4 to 9 are also plotted in Fig. 8. These rate-distortion plots indicate that the gaps between FS and MCVS-4 become narrower remarkably, which demonstrate the benefit gained from MCVS-4. This figure also includes the results when a search window size of ±1 pixel is adopted for a motion vector refinement (dotted lines in Fig. 8). It is expected that all tested algorithms can achieve better performances. However, our proposed MCVS-4 still has the best performance and the use of the refinement further minimizes the gap between MCVS-4 and FS.

Table 2 lists the simulation results for performing fast-forward playback with a speed-up factor of 5 on other test sequences. It can be easily seen that the bits to be generated for the proposed algorithm are much fewer than that of FDVS, especially for the case of using MCVS-4. For the sequences with very fast motion or scene variations such as “Foreman”, the range of motion vectors increases significantly, especially when consecutive frames are dropped in fast-forward playback. Therefore, it is difficult for FDVS to obtain the accurate composed motion vectors in this situation, as shown in Table 2. Table 2 further demonstrates the improvement of MCVS-2 and MCVS-4 compared with FDVS in “Foreman” by showing different speed-up factors. Since our proposed algorithm can prevent the irrelevant area from the motion vector tracking process, the composed vectors are relatively accurate compared with those from FDVS and E-FDVS. From Table 3, it can be found that MCVS-2 and MCVS-4 can achieve more gain as the speed-up factor increases for “Foreman”. From these statistics, we can conclude that the proposed algorithm can provide excellent results.

4.2. Comparison of complexity

Regarding the computational complexity of the proposed MCVS, the only major overhead compared to FDVS and E-FDVS is the SAD calculation required for the final selection of different resulted motion vectors from different candidates in the last step. To compare the computational complexity for different schemes, the average transcoding time per frame for various speed-up factors are measured and tabulated in Table 4. The experiments were performed on an Intel Core™2 CPU 6600 at 2.40 GHz PC with 2 GB memory. It can be easily seen that all algorithms can substantially reduce the computational complexity of the full conventional process by nearly 80%. Although the transcoding time of MCVS increases with the number of multiple candidates, the influence is very slight. The transcoding time of MCVS-4 is still quite similar with that of FDVS and E-FDVS.

It is well-known that FDVS and E-FDVS require extra MV buffers to store the motion vectors of the skipped frames that will be used for motion vector composition. For FDVS, it can be processed in the
forward order [19] when multiple frames are dropped. In the example shown in Fig. 3, the motion vectors of frame \( n \) are stored in the MV buffer when frame \( n-2 \) is skipped. If frame \( n-1 \) is also skipped, the incoming motion vectors from frame \( n-1 \) will be added with the motion vector of the corresponding dominant block in the MV buffer. The MV buffer is then updated with the new composed values. Therefore, only one MV buffer is required for FDVS. However, for consecutively dropped frames in

<table>
<thead>
<tr>
<th>Speed up factor</th>
<th>Full search</th>
<th>FDVS</th>
<th>E-FDVS</th>
<th>MCVS-2</th>
<th>MCVS-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>PSNR</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
</tr>
<tr>
<td>3</td>
<td>2780</td>
<td>34.02</td>
<td>2808</td>
<td>33.87</td>
<td>2808</td>
</tr>
<tr>
<td>5</td>
<td>2019</td>
<td>33.87</td>
<td>2223</td>
<td>33.61</td>
<td>2211</td>
</tr>
<tr>
<td>7</td>
<td>1635</td>
<td>33.84</td>
<td>1913</td>
<td>33.51</td>
<td>1871</td>
</tr>
<tr>
<td>9</td>
<td>1387</td>
<td>33.87</td>
<td>1703</td>
<td>33.51</td>
<td>1607</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison using different algorithms with various speed-up factors for transcoding the “Coastguard” sequence (PSNR: dB and bits: kbits).

![Fig. 8. Rate-distortion plots of different schemes for the “Coastguard” sequence, indicating speed-up factors from 4 to 9 at each rate-distortion point.](image-url)

<table>
<thead>
<tr>
<th>Sequences</th>
<th>Full search</th>
<th>FDVS</th>
<th>E-FDVS</th>
<th>MCVS-2</th>
<th>MCVS-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>PSNR</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
<td>Bits (PSNR)</td>
</tr>
<tr>
<td>Coastguard</td>
<td>2019</td>
<td>33.87</td>
<td>2253</td>
<td>33.61</td>
<td>2231</td>
</tr>
<tr>
<td>Foreman</td>
<td>1234</td>
<td>36.52</td>
<td>1525</td>
<td>35.80</td>
<td>1330</td>
</tr>
<tr>
<td>Mobile</td>
<td>2465</td>
<td>32.48</td>
<td>2635</td>
<td>32.35</td>
<td>2808</td>
</tr>
<tr>
<td>Salesman</td>
<td>468</td>
<td>38.30</td>
<td>501</td>
<td>38.01</td>
<td>496</td>
</tr>
<tr>
<td>Tempete</td>
<td>2207</td>
<td>33.49</td>
<td>2472</td>
<td>33.28</td>
<td>2411</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison using different algorithms at a speed-up factor of 5 for various sequences (PSNR: dB and bits: kbits).

<table>
<thead>
<tr>
<th>Speed-up factor</th>
<th>FDVS</th>
<th>E-FDVS</th>
<th>MCVS-2</th>
<th>MCVS-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>PSNR</td>
<td>Bits (%)</td>
<td>PSNR</td>
<td>Bits (%)</td>
</tr>
<tr>
<td>3</td>
<td>1714</td>
<td>36.24</td>
<td>4.9</td>
<td>-0.43</td>
</tr>
<tr>
<td>5</td>
<td>1525</td>
<td>35.80</td>
<td>-12.8</td>
<td>-0.61</td>
</tr>
<tr>
<td>7</td>
<td>1394</td>
<td>35.63</td>
<td>-16.1</td>
<td>-0.52</td>
</tr>
<tr>
<td>9</td>
<td>1283</td>
<td>35.60</td>
<td>-17.8</td>
<td>-0.54</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison of different algorithms with FDVS for transcoding the “Foreman” sequence.
other motion vector composition algorithms such as E-FDVS, they have to be processed in the backward order starting from the last skipped frame to the first skipped frame. This backward processing requires all motion vectors of the skipped frames to be stored. As a result, the number of the MV buffers required by E-FDVS depends on the speed-up factor. For instance, when the speed-up factor is SF, (SF–1) frames are skipped for every SF frames. It implies that (SF–1) MV buffers are required for E-FDVS. Interested readers are encouraged to read [19,22] about the MV buffer requirements of FDVS and E-FDVS. The MV buffer arrangement of the proposed MCVS-C is very similar to E-FDVS, except that one additional MV buffer is required for each candidate. For the example shown in Fig. 7(b) where C is equal to 2, like E-FDVS, two MV buffers are required to pre-store the motion vectors of frame n–1 and frame n–2. In the composition process, the MV buffer of frame n–1 is updated with the best composed motion vector, \( \mathbf{m}_{n-1} \), when frame n–1 is skipped (step 3 of Section 3.3), as E-FDVS does. Besides, one more intermediate MV buffer is required to store the second best candidate \( \min_{\mathbf{m}_{n-1}} \). In the next step when frame n–2 is skipped as shown in Fig. 7(c), the MV buffer of frame n–2 is updated with the best composed one, \( \mathbf{m}_{n-2} \), and the intermediate MV buffer is updated with \( \mathbf{m}_{n-2} \). Table 5 then summarizes the detailed comparison of the required MV buffers for various algorithms. Although the proposed algorithm needs extra memory, it achieves a significant gain in terms of rate-distortion performance. It is also noted that, for CIF sequences, the size of each MV buffer is only 792 bytes (22 \( \times \) 18 \( \times \) 2 bytes), which is always negligible in the server with transcoding capability.

### 5. Conclusions

This paper presents a novel motion vector composition algorithm, which is specifically designed for realizing fast-forward playback of a pre-encoded video by generating a new video with fewer frames through frame-skip transcoding. Different from the well-known forward dominant motion vector selection algorithm (FDVS) which only selects the best intermediate motion vectors with dominant MBs, our proposed multiple-candidate vector selection (MCVS) algorithm concentrates on the segments that are really related to the target MB throughout the skipped frames, and then several possible candidates are selected in motion vector composition. This process can fully make use of the relevant area to the target MB, and it is beneficial to perform fast-forward playback with a large speed-up factor. Its performance in terms of both quality and bit rate is substantially better than that of FDVS and E-FDVS, and has been verified experimentally. Besides, the proposed MCVS is adaptive in nature, and the number of candidate MBs can be adjusted according to the speed-up factor. One of our future works could focus on determining the relationship between the speed-up factor and the number of candidates selected for each skipped frame.

### Acknowledgment

The work described in this paper is partially supported by the Centre for Signal Processing, Department of Electronic and Information Engineering, Hong Kong Polytechnic University and a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU 5120/07E). Tsze-Kwan Lee acknowledges the research studentships provided by the University.

### References


Table 4

<table>
<thead>
<tr>
<th>Sequences</th>
<th>Full search Transcoding time</th>
<th>FDVS Time (Atime)</th>
<th>E-FDVS Time (Atime)</th>
<th>MCVS-2 Time (Atime)</th>
<th>MCVS-4 Time (Atime)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coastguard</td>
<td>1.159</td>
<td>0.245</td>
<td>0.248</td>
<td>0.253</td>
<td>0.263</td>
</tr>
<tr>
<td>Foreman</td>
<td>1.136</td>
<td>(–79%)</td>
<td>(–79%)</td>
<td>(–78%)</td>
<td>(–77%)</td>
</tr>
<tr>
<td>Mobile</td>
<td>1.163</td>
<td>0.230</td>
<td>0.232</td>
<td>0.238</td>
<td>0.249</td>
</tr>
<tr>
<td>Mobile</td>
<td></td>
<td>(–80%)</td>
<td>(–80%)</td>
<td>(–79%)</td>
<td>(–78%)</td>
</tr>
<tr>
<td>Salesman</td>
<td>1.095</td>
<td>0.213</td>
<td>0.253</td>
<td>0.217</td>
<td>0.220</td>
</tr>
<tr>
<td>Tempete</td>
<td>1.158</td>
<td>0.247</td>
<td>0.297</td>
<td>0.256</td>
<td>0.266</td>
</tr>
</tbody>
</table>

Table 5

<table>
<thead>
<tr>
<th>MV buffer</th>
<th>FDVS</th>
<th>E-FDVS</th>
<th>MCVS-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF-1</td>
<td></td>
<td></td>
<td>(SF-1) + (C-1)</td>
</tr>
</tbody>
</table>