International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 1, Januaryr- 2013

Data Embedding in JPEG Bitstream by Code Mapping

G.Narahari, *M. Tech Student*, and Mr. D. Sharath Babu Rao, *Faculty*
Department of Electronics & Communication Engineering
Jawaharlal Nehru Technological University
Anantapur, India

Abstract

We propose an algorithm to embed data directly in the bitstream of JPEG imagery. The motivation for this aproach is that images are seldom available in uncompressed form. Algorithms that operate in spatial domain, or even in coefficient domain, require full (or at best) partial decompression. Our approach exploits the fact that only a fraction of JPEG code space is actually used by available encoders. Data embedding is performed by mapping a used variable length code (VLC) to an unused VLC. However, standard viewers unaware of the change will not properly display the image.We address this problem by a novel error concealment technique. Concealment works by remapping run/size values of marked VLCs so that standard viewers do not lose synchronization and displays the image with minimum loss of quality. It is possible for the embedded image to be visually identical to the original even though the two files are bitwise different. The algorithm is fast and transparent and embedding is reversible and file-size preserving. Under certain circumstances, file size may actually decrease despite carrying a payload.

***Index Terms*—Code mapping, data embedding, JPEG, VLC,**
watermarking.

I. INTRODUCTION

Data embedding describes a general framework where a payload is embedded within a cover image for authentication, fingerprinting or ownership verification (watermarking), covert communications (steganography) or simply as a vehicle to attach metadata to a cover image (data embedding). All three attempt to exploit an unused communication channel in the cover image but, depending on the application, there are different approaches and requirements. In digital watermarking, the payload does not have to be large but embedding must be robust, or semi-fragile, and secure. In steganography, the payload can be substantial while the cover image is secondary. Robustness and security are still important but transparency is critical to hide the presence of a covert channel. Digital watermarking of images has traditionally been implemented either in spatial domain [1] or transform domain [2]. However, multimedia signals are seldom available in uncompressed form because compressed media is often the first generation signalavailable from digital cameras. It is, therefore, highly desirable to develop watermarking algorithms that work entirely in compressed domain. There has been substantial work in embedding watermarks in JPEG compressed imagery. Examples include classical transform domain watermarking algorithms where the watermark is embedded in appropriately selected transform coefficients. However, we do not consider these algorithms strictly “compressed domain” watermarking because partial decompression is required to gain access to transfrom coefficients. Examples are JSTEG [3], F5 [4], OutGuess [5], and J-Mark [6]. The term JPEG-to-JPEG watermarking has also been used [7], although the proposed algorithm is still not truly compressed domain watermarking. What is desirable is the ability to embed the watermark directly in the bitstream of compressed media with no transcoding or decompression. Bitstreamwatermarking is nowa recognized research subarea in watermarking. Compressed domain is a particularly challenging environment for data embedding. The reason is that embedding relies on redundancy in the cover media. Compressed domain, by definition, has little redundancy. One proposed approach is a form of LSB watermarking whereby levels representing a VLC carry the watermark. The algorithm has been applied to MPEG-2 streams [8]. Another similar method proposed for compressed domain is done by modulating DCT coefficient levels [9]. The drawback of both of these algorithms is that they are lossy. A lossless JPEG watermarking scheme appears in [10]. This algorithm is still not strictly “compressed domain”, as it needs partial decompression of DCT coefficients. In [11], a bitstream-level MPEG-2 watermarking is proposed. The approach is in fact implemented in the bitstream and embeds data in the LSB of the VLCs representing the DC differential of a block. Both approaches can be grouped under code mapping. Watermark embedding and recovery has also been formulated as problem in channel coding and error control [12]. More recently, attempts have been made to embed a watermark in the bitstream of H.264/AVC stream by intraprediction mode modification [13]. Embedding in the bitstream of open compression standards carries two limitations: fragility and weak security. For example, re-encoding at a different compression ratio may erase the watermark by replacing the code space. LSBs of VLCs in [8] and [11] are vulnerable to such attacks. This vulnerability is specifically mentioned in [6]. Embedding in an open standard such as JPEG or MPEG creates security holes. First, the bitstream must remain syntax compliant to remain viewable by standard viewers. Second, because the embedding rule is public, the attacker can erase the watermark and replace it using the same rule. Although the watermark is often encrypted, we believe bitstream embedding is not strictly appropriate as a digital watermarking tool. As a result we position our algorithm as a data embedding tool, and not necessarily a watermarking tool, that is best used in environments where "attacks" in the conventional sense are not relevant or likely. Such applications desire to exploit the unused capacity of the bitstream to embed metadata to save bandwidth and keep the metadata attached to the cover image. For example, the medical community has recently recognized the possibilities of watermarking medical imagery [14]. For example, embedding Electronic Patient Records(EPR), saves storage in Hospital Information Systems, enhances confidentiality and saves transmission bandwidth [15]. In addition, it is envisioned that doctors' notes can be embedded in EPR and be available for browsing by medical professionals. Since bitstream embedding keeps the stream syntax-compliant, embedded imagery will conform to the DICOM standard [16].

In this paper, we expand upon a recently published approach to compressed domain watermarking [17] and apply it to specific bitstream as defined by the JPEG standard [18]. Results reported here expand upon the work in [19], as well. The additional work includes a more systematic approach to code mapping in JPEG bitstream, substantially more experimental verification, application to color and satellite imagery, comparison with other compressed domain watermarking and discussions about robustness, transparency, and security. In the process, we have achieved six, at times conflicting, goals: 1) compressed domain implementation, 2) reversibility, 3) syntax-compliance, 4) file-size preservation, 5) no visual impact, and 6) blind decoding. The rest of the paper is organized as follows. Section II introduces data embedding through code mapping. Section III introduces run/size remapping of VLCs to hide or minimize the impact of embedding. Section IV presents arguments on transparency, robustness, and security as they relate specifically to bitstream embedding. Section V describes watermark recovery and Section VI presents experimental results.

II. Data Embedding by Code Mapping

A. Analysis of the Code Space

The proposed data embedding in compressed bitstream is accomplished by mapping eligible codes in the entropy portion of the bitstream to appropriate regions of the code space. To make this idea work, the JPEG code space needs to be explored further. The entropy-coded portion of a compressed bitstream consists of individual VLCs that can be classified into four categories: 1) valid, 2) invalid, 3) used, and 4) unused but valid. A *valid* code word is a VLC that is generated by the specific source coding algorithm adopted by the compression algorithm. In JPEG, Tables K. 5 and K. 6 list all valid luminance and chrominance VLCs [18]. An *invalid* code word is a code thatis either outside the valid code space or violates the prefix condition. Then there is the case of *used* versus *unused* code words. The code space of an optimally coded bitstream is full. This means that any bit flip at any location of any VLC will generate a code word that violates the prefix condition or simply

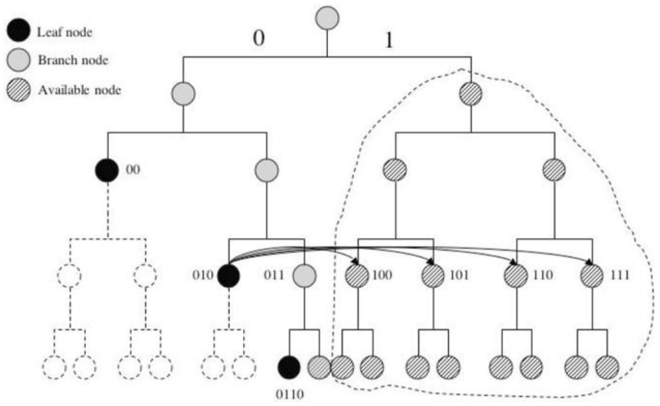

Fig. 1. Code tree for variable length codes. Leaf nodes represent used VLCs.
Branch nodes are transition nodes and cannot be valid VLCs themselves. Available nodes define legal code words and are candidates for mapping. The entire right half of the tree is available for code mapping. For example, 010 can be mapped to any of the four nodes but not to 011 because it will violate the prefix condition. Since codes are mapped to nodes at the same level, there is no change in code length or file size.

becomes another code. This situation may cause synchronization failure or simply lead to undetectable errors. Majority of JPEG encoders, however, do not fully utilize the code space. What this means is that a large segment of valid VLCs simply are not needed and, hence, go unused. More importantly, except for custom coded JPEGs, the header of most images mistakenly communicate to the decoder that the code space is in fact fully occupied. This results in a situation where there is a considerable void between the valid code space and the used code space. The reason for this is that JPEG encoders, in general, are not optimized for any specific image. It is simply assumed that the entire code space will be used. The core idea introduced in [17] and illustrated in [19] embeds data by pushing valid VLCs to unused portions of the code space. See Fig. 1. If the entire code space were occupied, code mappingwould not be possible. The decoderwould have noway of telling a mapped VLC from the original. To identify the available code space for embedding, the used portion of code space is mapped to a binary tree whose leaf nodes define the VLCs. Mapping of a VLC is accomplished by flipping one (or possibly more) bit(s) at appropriate locations. Flipping a bit is equivalent to mapping the VLC to another node at the same level (since the length does not change). However, not all nodes are available for this mapping. Only those mappings that do not cause *collisions* are eligible. Collision occurs when the mapped VLC coincides with a used VLC, prefixes a used VLC or another used VLC prefixes it. We will now demonstrate how code mapping is applicable to JPEG code space.

B. JPEG Code Space

The entropy-coded portion of JPEG can either be Huffman or arithmetic coded. Because most images use Huffman encoding, that is the focus of our discussion. The JPEG standard does not require that a specific Huffman table be used for every image.
Instead, to maximize compression, the standard encourages customizing the Huffman table based on the specific run/size occurrences for each individual image. However, most commercial JPEG encoders bypass this customization step and instead use example tables provided in the standard itself. These standard tables include VLC assignments for all 162 possible run/size combinations. The VLC assignment to each of the 162 possible run/sizes was determined based on a large library of images, approximately 100 000 according to the standard. However, for a specific image, the default code space is not optimized leaving a large portion of the code space unused. For a code word to be successfully mapped, at least one bit must be changed without causing a collision. For the default Huffman table, it is not possible to flip a bit without violating the prefix property of the code table. However, it may be possible to map a VLC to the outside of the used code space. It is not unusual to find a natural image that uses only 40 to 60 of the possible 162 run/size combinations. The mapped VLC will still belong to the default Huffman table but is not used in the image. To maintain lossless embedding, once a VLC is mapped to an unused VLC, the unused VLC must be considered "used." Essentially, this keeps two used VLCs from being mapped to the same unused VLC (it would be impossible to know which was the original in this case). For security purposes, the hash of the message is then embedded in the header of the JPEG file. This step is further explained in Section IV. Embedding rule is summarized below.

Embedding

1. Parse the bitstream, extract VLCs.
2. Build the code tree, identify used and unused code space.
3. Identify qualified VLC pairs for mapping.
4. To embed a 1, map one used VLC to its unused counterpart.
5. Remap run/size of the unused VLC to minimize or eliminate visual impact.
6. To embed a 0, do not map a qualified VLC.
7. Embed a hash of the message in the header.

III. Error Concealment in Code Mapping

The concept of error concealment is generally associated with MPEG video. The idea is to replace corrupted macroblocks with some form of replenishment, through a variety of ways including inter or intra prediction or simple repetition of the corresponding data from previous frames. In this section we point out that error concealment is also relevant in the context of JPEG when embedding is treated as forced bit errors. One of the key objectives in this work is that embedded JPEG stream must be viewable by any JPEG viewer. Therefore, we define two classes of viewers, 1) *embedding-aware* viewer, meaning that the viewer is also an authorized decoder and 2) *standard* viewer, meaning all other publicly available JPEG viewers. By mapping a used VLC to an unused VLC, not only the VLC will change, so will its run/size assignment. This remapping of the run/size would most likely result in catastrophic loss of synchronization in a standard viewer. The reason standard viewers are handicapped is because every VLC is a legal VLC. Therefore, a mapped VLC will be displayed with a run/size different from the original. Whereas, for embedding-aware viewers, mapped VLCs are "illegal" VLCs, illegal for *that* image that is, and are recognized as such. Note that the way JPEG decoders actually produce a visual image is unrelated to the VLCs themselves. The VLCs are simply pointers to run/sizes. It is the run/size pair that recreates the quantized

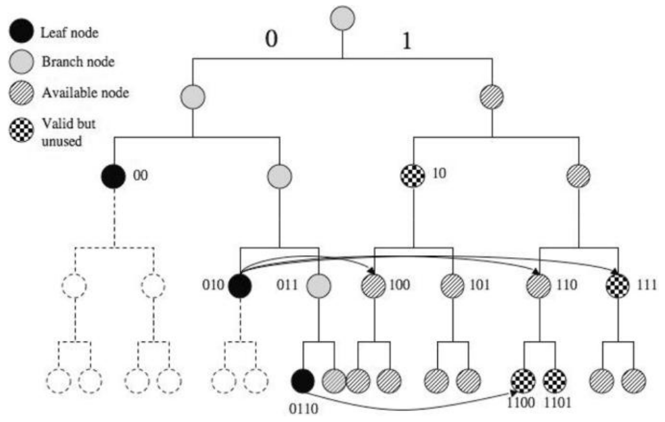

Fig. 2. In many JPEG images, a large number of VLCs are defined but never used in the image. This diagram illustrates the difficulties a standard viewer might have with a watermarked image. Watermarked VLCs are incorrectly parsed because watermarking may violate the prefix rule. The solution is to redefine the original Huffman table.

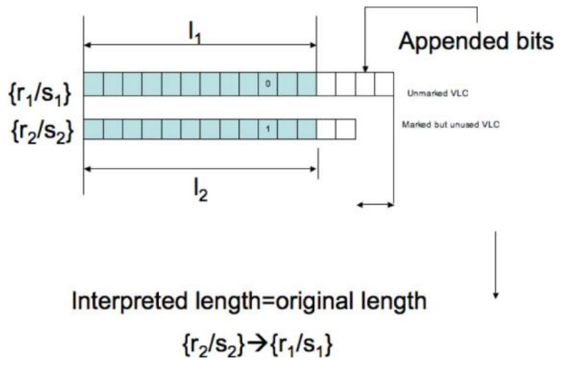

Fig. 3. Mapped VLC appears as another VLC of the same length. The new size S_2 must be remapped to S_1 to maintain synchronization. Also, $r_2 = r_1$.

DCT coefficients. Therefore, if a VLC is mapped to a new runs/size, then the value of run/size affects only the manner in which standard decoders *interpret* the mapped VLCs. What needs to be done at this point is a clever way to redefine the run/size values of the mapped VLCs to make standard viewers interpret the embedded VLC as close to its original as possible.

A. Error Concealment by Run/Size Remapping

Since mapped VLCs are only mapped to unused VLCs, modifying their runs/sizes will only affect how the image is displayed. No other part of the image is affected by this modification. Ideally, run/size should be changed to match the original run/size. However, this cannot be done in all cases. The interpreted length of the mapped VLC may be equal to, shorter, or longer than the length of the original VLC. Fig. 3 shows the first case. A mapped VLC with a run/size r_1/s_1 appears as anotherVLC with run/size r_2/s_2 but the same length. The viewer then expects the beginning of the next VLC to be S_2 bits later. In reality, the next VLC begins S_1 bits later. If nothing is done, synchronization will be lost. The solution is to redefine the run/size values of the mapped VLC to match the original one. This can be done directly in the Huffman table definition of the JPEG file. Two other cases are shown in Figs. 4 and 5. In both cases,the run of the interpreted VLC must

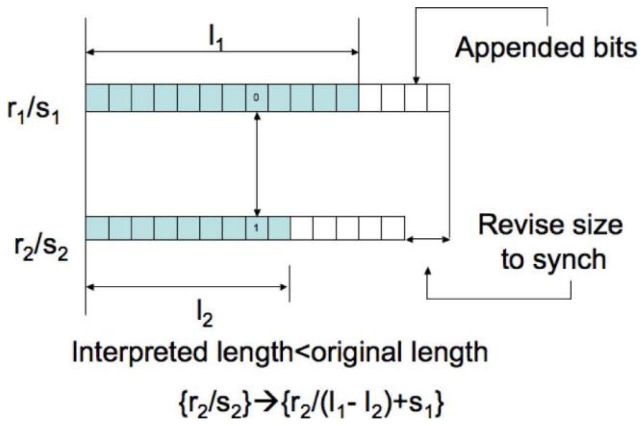
l_1
Appended bits
r_1/s_1
r_2/s_2
Revise size to synch
l_2
Interpreted length<original length
$\{r_2/s_2\} \rightarrow \{r_2/(l_1 - l_2)+s_1\}$

Fig. 4. Valid, but unused, VLC appears as a prefix in a mapped VLC. S_2 must
be redefined as shown to maintain synchronization.

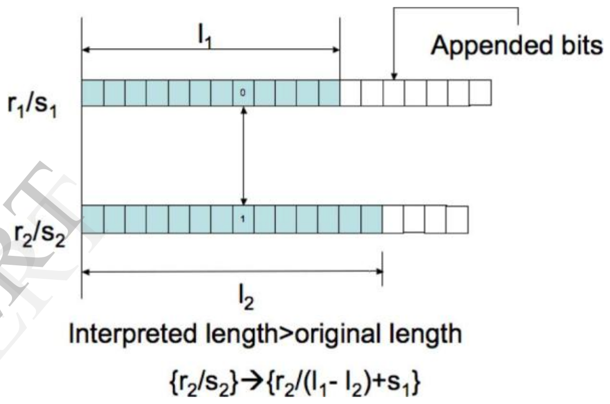
l_1
Appended bits
r_1/s_1
r_2/s_2
l_2
Interpreted length>original length
$\{r_2/s_2\} \rightarrow \{r_2/(l_1 - l_2)+s_1\}$

Fig. 5. Interpreted length of the mapped VLC is now a longer VLC. must
be redefined as shown to maintain synchronization.

Match the original and the size is redefined to make the overall length (VLC length plus appended bits) the same as the original VLC. It is illustrative to establish the application of this concept to actual JPEG code space.

B. Run/Size Remapping for JPEG VLCs

Case 1: Interpreted VLC Length is Equal to the Mapped VLC Length: Ideally, the lengths of the original VLC and the mapped version are the same, but this is not guaranteed. If they are the same, then this is the most favorable case. In this case, it is always possible to replace the run/size of the interpreted, but unused VLC with the run/size of the original VLC. The run of the interpreted VLC can always be modified because it never appears in the image. This makes mapping of this particular group of VLCs completely invisible since nothing is changed for display purposes. The VLC to be mapped is number 43 in Table K. 5 [18] and is represented by run/size/length/total length of 4/2/10/12 and shown in Fig. 6. Flipping the LSB maps this VLC to number 112 represented by 11/1/10/11 for a total length of 11 bits. This VLC does not appear anywhere in the image.If the size of the interpreted VLC is not changed, the viewer will parse a total of 11 bits, causing synchronization failure. Instead, we change the run/size from 11/1 to 4/2 in the appropriate Huffman table. Total length of the interpreted VLC will then be 12 and the run remains unchanged. Therefore, embedding will have zero visual impact.

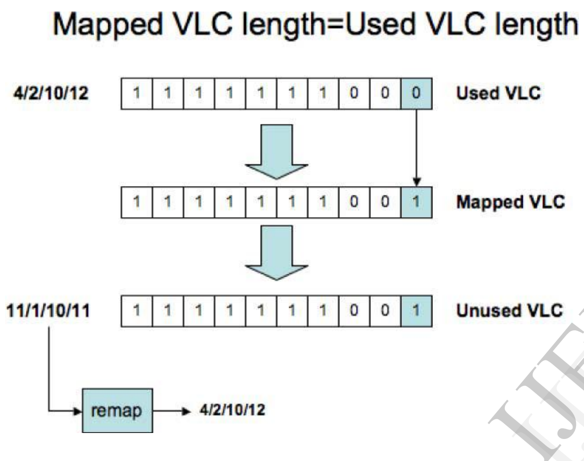

Fig. 6. Embedding-unaware decoder mistakenly decodes a marked VLC as an
unmarked, valid but unused VLC of the same length. Run/size of the interpreted VLC must be changed to maintain synchronization.

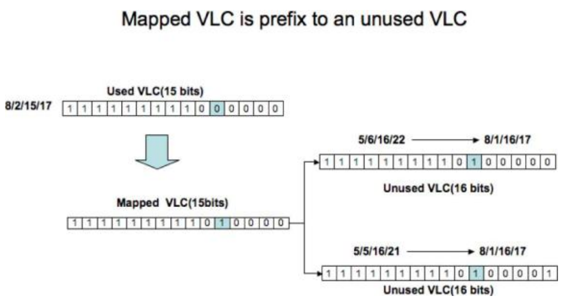

Fig. 7. Mapped VLC may become a prefix to several unused VLCs. Embedding- unaware decoder mistakenly decodes a marked VLC as an unmarked valid but unused VLC of the same length. Run/size of the interpreted VLC must be changed to maintain synchronization.

Case 2: Interpreted VLC Length Longer Than Mapped VLC

Length: In this case, there is the possibility that a single VLC when mapped becomes a prefix to two or more VLCs. See Fig. 7. The run can always be made identical but the size will have to be changed to keep synchronization. This means that if the length difference is one, then the size must be modified to be one less than the original. Since the lengths of the mapped versions are longer than the original VLC, what will determine how the mapped versions are displayed will be the appended bits. Also, since the decoder could interpret the mapped version as multiple different VLCs, each of the possible interpreted VLCs run/size combinations must be altered. They will all be modified to the same new run/size. The difference between this case and Case 1 is that changing the size of the VLC will also change the number of appended bits and how the VLC is displayed. If this change is deemed too much, that particular VLC should not be mapped.

Case 3: Interpreted VLC Length is Shorter Than Mapped

VLC Length: The final, and most problematic, possibility is that the interpreted VLC will be shorter than the actual length, as shown in Fig. 8. In this case, it is possible that two or more VLCs when mapped will be interpreted as the same VLC by an embedding-unaware decoder. It is important to note that the mapped VLCs are distinct; however, the first N bits of each are the same and belong to an unused VLC. Therefore, an embedding -unaware decoder will interpret the first N bits of any of these mapped VLCs as the same valid VLC. Therefore, it would not be possible to mask the effect of mapping since it is possible to modify one run/size only. The solution then is to select only

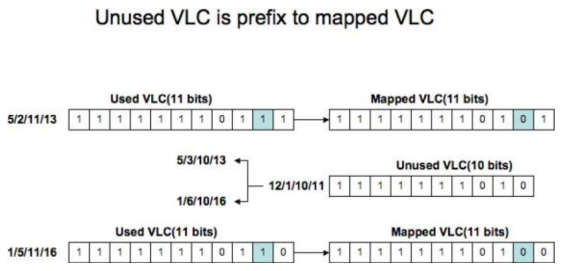

Fig. 8. Two different VLCs, when mapped, may be interpreted as another VLC of shorter length. To keep synchronization, run/size of the interpreted VLC must be changed to match either the first or the second mapped VLC. In this case, only one of the VLC will then be selected for embedding. If capacity is important, then the VLC that occurs more often is the one that should be mapped.

one of the two VLCs. Consider two mapped VLCs are defined by 5/2/11/13 and 1/5/11/16. The interpreted VLC is 12/1/10/11. This run/size must be redefined to either 5/3/10/13 to match the first VLC or 1/16/10/16 to match the second. Notice in both cases the total length, hence synchronization, is maintained. The choice of which VLC to map depends on a number of factors. If embedding rate is important, then the VLC with a higher occurrence rate must be selected. Note that the remapping of run/size value will have a visual impact. So between the two choices, the one with the smallest deviation in size must be chosen. In the example of Fig. 8, both mappings are equivalent because they change the size by one relative to original VLCs.

IV. ROBUSTNESS TRANSPARENCY, AND SECURITY

Transparency refers to the impact of embedding on visual and statistical properties of the cover image.We have identified three code-mapping cases. Of the three, Case 1 produces zero visual impact on the cover image. If the embedding rule is limited to Case 1, the displayed original and embedded image will remain visually identical. This property is unique among watermarking algorithms in that two image files that differ in binary content produce identical images. The reason for this property can be traced back to the JPEG standard. Two different code words project to the same color if their run/size designations are the same. Cases 2 and 3 do cause visual degradation but the impact can be kept to a minimum if the size of the new VLC is kept close to the original.We have not reassigned sizes by more than one. The resulting peak signal-to-noise ratio (PSNR) (Table III) is 50 dB or higher; more than needed for transparency. PSNR is computed by the mean square difference of the embedded image and the original after they are decompressed. Therefore, any difference is due to embedding, not compression. Robustness refers to the ability of the detector to recover embedded data when the cover media is subjected to malicious attacks as well as ordinary signal processing operations. Strict compressed domain embedding is vulnerable to recompression and transcoding. Recompression is achieved by scaling the quantization tables. This scaling may push quantized coefficients to different quantization bins, overwrite the mapped VLC and, thus, alter code mapping carefully orchestrated at the embedder. However, depending on the scaling parameter, quantized DCT coefficients may not change and, thus, the corresponding VLCs will not change either. For example, for *lena* 30% of nonzero DCT coefficients remain unchanged using a quantization scale factor of 1.5. This number is reduced to 18% for scale factor of 2. So it is possible to choose VLCs that could survive moderate recompression although larger scale factors will likely cause full erasure. Vulnerability to recompression is not unique to code mapping. Label-carrying VLCs [8], and coefficient embedding such as F5 and J-Mark are equally vulnerable to erasure by recompression. Another issue related to robustness is the occurrence of bit errors. However, bit errors adversely affect JPEG decoding as a whole and not just the data recovery portion. Other than placement of restart markers, the JPEG standard has little inherent protection against bit errors. Therefore, it is expected that bit errors are handled at a higher protocol level. It is in fact rare to encounter JPEG files today that have been corrupted by bit errors. Security is declared perfect if repeated observations of the image does not cause leakage of the secret key [20]. Secure embedding in compressed domain of an open standard is a difficult problem because of the imposed framework. Embedding must take place in a JPEG bitstream and the stream must be viewable by standard viewers. The problem is that every piece of information for decoding a JPEG stream is in the header and cannot be concealed or encrypted. Data hiding is often tied to some form of secret key that is exchanged with the decoder. In our approach, the key is the pair of (used VLC, unused embedded VLC). This association, however, cannot be kept secret if the bitstream is to remain syntax-compliant and viewable by standard viewers. To remain viewable, the modified Huffman table must be stored in the header portion and remain in the clear. If it is in the clear, then modified run/size values and the knowledge of the algorithm can be used to detect the watermark, much like what the authorized decoder does. Any attempt to conceal or encrypt the modified Huffman table will make the image unviewable by standard viewers. The most serious concern, for example in medical applications, is privacy and forgery. Attackers may erase the embedded data by re-encoding and replacing it with someone else's. To identify this attack, we rely on computing MIC(Message Integrity Code) and storing it in the header section of the image. MIC is computed by hashing, using MD2 for example, the concatenation of the embedded data with a secret key, MD2(*key*,). The secret key can be unique to the image and/or owner. MD2 is a 128-bit hash and is considered secure.

Most images contain user fields that can be used to store the 16 byte hash. The attack scenario is as follows. The attacker identifies mapped VLCs and either strips the embedded data and/or inserts his own. The owner cannot identify tampering because VLCs are legally marked according to the algorithm. However, an MIC check can resolve this matter. Authorized decoder recovers the forged data , appends the secret key and computes its MIC, MD2(*key*,). This MIC will not match what is stored in the header and, thus, signals authentication failure.

V. Data Recovery

The most important piece of information for the decoder is the list of mapped VLCs. This list is easily generated at the encoder side by parsing the image. However, the decoder cannot inspect the image in the same way as the embedder because VLCs have already been changed. The easy way would have been to include this table as an additional Huffman table in the header. Even though this addition would have minimally increased the file size, we chose not to do so. Instead, we have developed a procedure by which the decoder, in an indirect way, builds the VLC pairs (original and corresponding mapped VLCs) from existing Huffman tables that are always included in the image. Here we examine how the decoder handles the three cases covered in the previous section. First, the decoder merges the publicly available standard Huffman table with the Huffman table extracted from the embedded image. The merged table is then sorted by run/size. An examination of this table reveals the following. For Case 1, the decoder will see duplicate run/sizes for pairs of VLCs of the same length. This would not happen in an unmarked image. Since we know what the true run/size is for any VLC, the VLC with the modified run/size must have been a mapped one. Similar logic holds for Case 2 and 3. The decoder identifies pairs of VLCs with the same run/size. In this case, they will be of different length but that does not matter. Of the two VLCs the one with modified run/size is the marked one. By comparing the original and modified run/size and knowing what the encoder had to do to maintain synchronization, it is possible to arrive at the run/size of the original VLC and, hence, the VLC itself, prior to embedding. The decoder has now established one VLC pair. This knowledge can be used to completely reverse the embedding after extraction. Once the VLCs have been linked, the decoder then parses through the image data looking for VLC pairs. Once a VLC is located, the decoder checks to see if it is a marked VLC, or a VLC that could have been mapped. If the VLC could have been mapped but is not, then it must be carrying a data bit 0. Otherwise it carries a 1.

A. Data Extraction Summary

1. Parse the image.
2. Extract used VLCs.
3. Compare run/size of used VLCs with run/size of the standard Huffman table.
4. Work backward to identify corresponding {original,mapped} VLC.
5. Build{origina,mapped} VLC pairs list.
6. Extract a 1 if eligible VLC pair appears in the image.
7. Extract a 0 if eligible VLC pair appears unchanged.
8. Using secret key, compute MIC of extracted data and compare with the stored copy.
9. If two MICs match authentication succeeds.

VI. Results

The algorithm was tested across ten images obtained from the online image database of the University of Southern California [21]. Selected images are in 512 X 512 TIFF format and were converted to grayscale and compressed by varying JPEG Q-factors. Testing involved embedding each image with the maximum allowable capacity. PSNR was used as a measure of quality by computing the mean square difference between the embedded image and original after they are decompressed. Therefore, any difference is due to embedding, not compression.

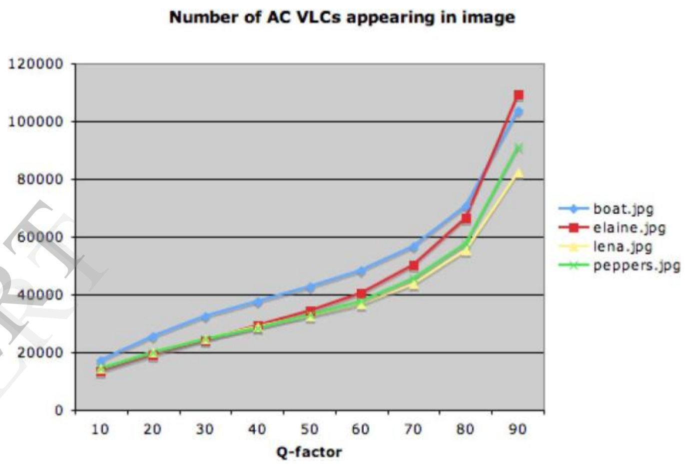

Fig. 9. Number of VLCs versus JPEG Q-factor. As Q decreases (compression
increases), the number of VLCs also decrease.

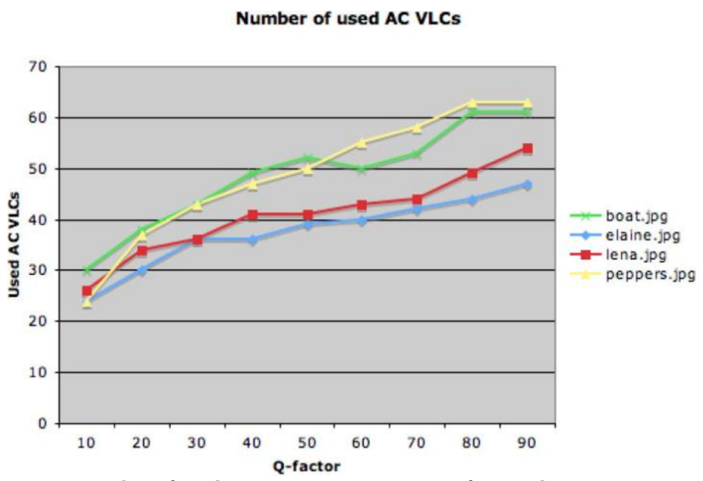

Fig. 10. Number of used AC VLCs versus JPEG Q-factor. These numbers are each out of the default 162 defined in the JPEG standard. Since these numbers are far less than nominal, there exist redundancies in the code space which will be used for embedding.

A. Pre-Embedding Statistics

Prior to embedding several statistics were recorded which aided in identifying key issues regarding embedding capacity and PSNR. Fig. 9 shows the number of VLCs that actually occur in each image versus Q-factor. This verifies that as Q increases so does the total number of VLCs. However, the number of unique AC VLCs was found to be far less than the maximum 162. This supports the claim that there is indeed unused code space within the JPEG images. As shown in Fig. 10, the number of AC VLCs used in each image never exceeds 70, corresponding to code space occupancy of 70/162 = 43.2%. The final pre-embedding statistic is the number of qualifying VLCs, i.e., VLCs that can be mapped to an unused VLC, versus Q-factor. Results are shown in Table I. From the data collected, it is clear that increasing compression will reduce the number of VLCs used within any given image. The number of unique AC VLCs, however, is image dependent since different images will use different VLCs. With this in mind, it can be concluded that the number of qualifying VLCs is more influenced by image content than the Q-factor.

TABLE I
NUMBER OF QUALIFYING VLCS VERSUS JPEG Q-FACTOR

	10	20	30	40	50	60	70	80	90
Baboon	12	12	16	15	17	19	19	19	16
Boat	10	10	13	17	20	18	19	25	25
Bridge	11	11	12	15	19	17	22	22	24
Elaine	7	10	8	6	7	6	8	8	11
F16	8	8	9	17	18	20	23	27	32
Gray21	6	11	15	15	16	14	17	22	29
Lena	10	6	8	9	9	9	10	18	18
Peppers	9	11	13	17	18	23	24	27	26
Splash	9	11	10	13	15	20	23	25	31
Tiffany	12	10	13	16	21	22	28	32	33

TABLE II
EMBEDDING CAPACITY(BITS) VERSUS JPEG Q-FACTOR

	10	20	30	40	50	60	70	80	90
Baboon	792	2035	1071	1342	820	887	953	317	195
Boat	476	350	457	355	439	514	231	270	528
Bridge	719	554	469	583	683	244	283	161	188
Elaine	111	130	139	200	249	79	92	146	339
F16	545	400	310	367	401	298	259	210	222
Gray21	152	261	488	292	333	310	752	934	1152
Lena	263	197	235	259	317	94	99	86	152
Peppers	350	247	338	242	278	357	164	144	378
Splash	284	369	261	296	378	547	383	359	296
Tiffany	226	348	222	310	332	391	534	332	279

B. Post-Embedding Statistics

Results for embedding capacity in bits versus JPEG Q-factor are shown in Table II.With the exception of *baboon*, the overall capacity did not vary much with changing Q-factor. Data suggests that the Q-factor weakly affects overall embedding capacity. From the data collected, PSNR is not directly related to the number of embedded bits. Instead, any change in PSNR is a result of a run/size remapping. Table III shows PSNR versus JPEG Q-factor versus capacity. The only connection Q-factor has to PSNR is that it regulates the number of VLCs which occur in the image and, hence, the resulting number of qualifying VLCs. It does not, however, influence the mapping of used VLCs to unused VLCs which ultimately will alter the visual quality of the image. After applying the algorithm to several images using the standard VLC tables, it is clear that the remapping of VLCs is not consistent from image to image. Therefore, in order to maintain sufficient PSNR, only VLC re-mappings that minimize visual distortions should be used. This approach may reduce capacity though the amount by which it is reduced is strictly image-dependent. Note that it is generally accepted that PSNR of 35 dB and above is visually indistinguishable from the original.

TABLE III
AVERAGE CAPACITY AND PSNR ACROSS ALL JPEG Q

Image	AverageCapacity(bits) Across All Q	Average PSNR(db)
Baboon	934.67	50.39
Boat	402.22	55.42
Bridge	431.56	54.41
Elaine	165.00	62.25
F16	334.67	53.46
Gray21	519.33	50.82
Lena	189.11	58.05
Peppers	277.56	57.63
Splash	352.56	57.83
Tiffany	330.44	57.55

TABLE IV
CONTRIBUTION OF CHROMINANCE (Q = 50)

Image	LumaCapacity(bits)	ChromaCapacity(bits)	Payload	Contribit of Cromi Nance
lena	314	86	400	21.5%
baboon	821	160	981	16.31%
f16	376	115	491	23.42%
peppers	282	274	556	49.28%
Earth	366	505	871	57.97%
splash	366	245	611	40.10%
sailboat	370	243	613	39.64%
louse	572	220	792	27.78%
tiffany	328	387	715	54.12%

C. Embedding in the Chrominance Band

Most watermarking algorithms stop at the luminance band.
Code mapping, however, is equally applicable to the chrominance band. JPEG standard supports up to four
Huffman AC tables; two for luminance and two for chrominance. The chrominance VLCs are defined in Table K. 6 of the standard. There are 162 chrominance VLCs, equal to the luminance component. The VLCs themselves are in fact shared between the two tables. The difference is that same VLCs are assigned to different run/sizes. Therefore, it is not possible to distinguish between chrominance and luminance VLCs based solely on the bit string. What are of most interest at this stage is howthe two code spaces compare and how much capacity increases by embedding in chrominance band, as well. Table IV illustrates the contribution of luminance to the total capacity for Q = 50 . Because JPEG color model uses YUV with 4 : 2 : 0

component sampling, adding chrominance means increasing image blocks by 50% compared to grayscale. Stated differently, chrominance blocks account for 1/3 of total blocks. Data shows that chrominance VLCs, depending on the image, account for less or more than 1/3 embedding capacity. The percentage is different across images depending on the content, population of qualifying VLCs and the frequency they occur in the image. Nevertheless, there is a substantial increase in payload by embedding in the color band, as well.

D. Computational Benchmarking

One of the promises of data embedding applied directly in the bitstream is fast execution time. To test this property, we applied the algorithm to satellite scale images of sizes close to 1000^2

Fig. 12. 3000□3000 image of San Francisco Bay collected by IKONOS. This image carries 180 000 bits of information at zero loss in quality. This is achieved by using only equal length VLC pairs and mapping their runs/sizes accordingly.

pixel shown in Fig. 12. Such image sizes have seldom been tried in watermarking literature. Our algorithm embedded 180 000 bits with zero loss in visual quality. This goal was achieved by limiting VLC mapping to Case 1 mapping. Higher embedding rates are possible by including other cases. For benchmarking we chose F5 [4]. F5 is arguably the best known JPEG watermarking algorithm that comes closest to compressed domain implementation. Data in Table V shows that our algorithm runs five to seven times faster than F5. For sizes bigger than 1552 X 1552, F5 simply hangs.

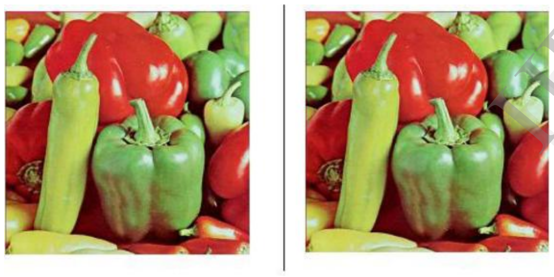
Fig. 11. Original (left) and embedded color image (right). In this case, chrominance band accounts for close to 50% of embedding capacity. Run/size mapping makes the two images visually, and numerically, identical.

VII. Conclusion

In this paper, we have advanced the state of the art in JPEG data embedding on several fronts. The data is embedded directly in the bitstream and executes considerably faster than existing techniques, which require full or partial decompression. Embedding is lossless in the sense that once the data is removed, the image can be restored to its original state with no changes. The stream, despite carrying a payload, remains syntax-compliant and, hence, viewable by standard viewers. Notably, marked images can be made mathematically and, thus, visually identical to the original image. File size increase is only due to the 16-byte hash value. In most cases there are redundant fields in the JPEG header that can be removed to offset this addition. Therefore, no file size increase is experienced. In fact, we have observed that marking a VLC may negate the need for zero pads and, thus, may actually reduce the file size.

REFERENCES

[1] F. Hartung and B. Girod, "Digital watermarking of uncompressed and compressed video," *Signal Process.*, vol. 66, pp. 283–301, May 1998.
[2] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon, "Secure spread spectrum watermaking for multimedia," *IEEE Trans. Image Process.*, vol. 6, no. 12, pp. 1673–1687, Dec. 1997.
[3] Steganography Software for Windows [Online]. Available: http://www.stegoarchive.com
[4] A. Westfeld and A. Pfitzmann, I. S. Moskowitz, Ed., "High capacity despite better steganalysis (F5-a steganographic algorithm)," in *Proc.*
Information Hiding 4th Int. Workshop, New York, 2001, vol. 2137, pp. 289–302, Springer-Verlag.
[5] N. Provos, "Defending against statistical steganalysis," presented at the 10th USENIX Security Symp., Washington, DC, 2001.
[6] P. H. W. Wong, O. C. Au, and J. W. C. Wong, "A data hiding technique in JPEG compressed domain," in *Proc. SPIE Security and Watermarking of Multimedia Contents III*, 2001, vol. 4314, pp. 309–320.
[7] P. H. W. Wong and O. C. Au, "A capacity estimation technique for JPEG-to-JPEG image watermarking," *IEEE Trans. Image Process.*, vol. 13, no. 8, pp. 746–752, Aug. 2003.
[8] G. C. Langelaar *et al.*, "Watermarking digital image and video data," *IEEE Signal Process. Mag.*, vol. 17, no. 5, pp. 20–46, Sep. 2000.
[9] L. Chun-Shien, J. Chen, H. Liao, and K. Fan, "Real-time mpeg-2 video watermarking in the vlc domain," in *Proc. Int. Conf. Pattern Recognition*, 2002, vol. 2, pp. 552–555.
[10] J. Fridrich *et al.*, "Lossless data embedding with file size preservation," in *Proc. EI SPIE, Security and Watermarking of Multimedia Contents VI*, San Jose, 2004, vol. 5306, pp. 354–365.
[11] H. Liu, F. Shao, and J. Huang, "A MPEG-2 video watermarking algorithm with compensation in bit stream," in *Proc. DRMTICS*, 2005, pp. 123–134.
[12] P. Loo and N. Kingsbury,"Watermark detection based on the properties of error control codes," *IEEE Proc. Vis. Image Signal Process.*, vol. 150, no. 2, pp. 115–121, Apr. 2003.
[13] Z. D. Zou and J. A. Bloom, "H.264/AVC stream replacement technique for video watermarking," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2008, pp. 1749–1752.
[14] B. M. Planitz and A. J. Maeder, "A study of block-based medical image watermarking using a perceptual similarity metric," in *Proc. Digital Image Computing: Techniques and Applications*, 2005, pp. 70–77.
[15] K. Navas, M. Sasikumar, and S. Sreevidya, "A benchmark for medical image watermarking," in *Proc. 14th Int. Workshop on Systems, Signals and Image Processing, and 6th EURASIP Conf. Focused on Speech and Image Processing,Multimedia Communications and Services*, Jun. 2007, pp. 237–240.
966 IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 19, NO. 4, APRIL 2010
[16] The DICOM Homepage [Online]. Available: http://medical.nema.org
[17] B. G. Mobasseri and R. J. Berger, II, "A foundation for watermarking in compressed domain," *IEEE Signal Process. Lett.*, vol. 12, no. 5, pp. 339–402, May 2005.
[18] Int. Telecommunication Union, CCITT Recommendation T.81, Information Technology-Digital Compression and Coding of Continuoustone Still Images-Requirements and Guidelines 1992.
[19] R. J. Berger, II and B. G. Mobasseri, E. J. Delp, III and P. W. Wong, Eds., "Watermarking in JPEG bitstream," in *Proc. Security, Steganography, andWatermarking of Multimedia Contents VII*, 2005, vol. 5681, pp. 539–548, SPIE.
[20] F. Cayre, C. Fontaine, and T. Furon, "Watermark security: Theory and practice," *IEEE Trans. Signal Process.*, vol. 53, no. 10, pp. 3976–3987, Oct. 2005.
[21] The USC-SIPI Image Database [Online]. Available: http://sipi.usc.edu/ database

Bijan G. Mobasseri (M'75–SM'01) received the B.S., M.S., and Ph.D. degrees from Purdue University, West Lafayette, IN, in 1973, 1974, and 1978, all in electrical engineering.
He is a professor of electrical and computer engineering at Villanova University, Villanova, PA. His research interests are in the areas of communications and signal processing, information embedding for authentication and steganography, digital watermarking, time-frequency analysis, image and video compression, and pattern recognition. He has been funded by the AFOSR and AFRL to develop compressed domain digital watermarking algorithms for images and video. He is currently funded by the ONR and NUWC to develop data embedding algorithms in undersea signal authentication applications.
Robert J. Berger, II, photograph and biography not available at the time of publication.
Michael P. Marcinak, photograph and biography not available at the time of publication.
Yatish J. NaikRaikar received the B.E. degree in ECE from Goa University,
India, in 2003, and the M.S. degree in EE from Villanova University, Villanova, PA, in 2006.
He is currently a Senior Engineer in the Media Processing Group at Ittiam Systems, Ltd. From 2007-2009, he was a member of the Core Technologies Group at Ingenient Technologies, Ltd. His areas of interest include video compression, video processing, watermarking, multimedia communication systems, and embedded systems.

Authors Profile

G.Narahari received B.Tech Degree in Electronics&Instrumentation Engineering from SrinivasaInstitute of Technology and Management Studies, Chittoor, India. Presently he is pursuing his M. Tech in Digital Systems & Computer Electronics specialization in the Department of Electronics & Communication Engineering from Jawaharlal Nehru Technological University, Anantapur, India. His research interests include Data Embbeding In JPEG Bitstream By Code Mapping ,

Mr. D. Sharath Babu received his B.Tech Degree in Electronics & Communication Engineering from Sri Krishna Devaraya Engineering college, gooty, India. And he received his Master of Engineering (ME) from Osmania University, Hyderabad. Presently he is pursuing his Ph.D in "VLSI Implementation of Gigabit Ethernet MAC" in the Department of ECE, from JNTU Anantapur, India. He is presently working as a Lecturer in the

Department of Electronics & Communication Engineering, JNTU University, Anantapur. His research interests include VLSI and Wireless Communications.