Memory Efficient Hierarchical Lookup Tables for Mass Arbitrary-Side Growing Huffman Trees Decoding

Sung-Wen Wang, *Member, IEEE*, Ja-Ling Wu, *Fellow, IEEE*, Shang-Chih Chuang, Chih-Chieh Hsiao, and Yi-Shin Tung

***Abstract*—This paper addresses the optimization problem of minimizing the number of memory access subject to a rate constraint for any Huffman decoding of various standard codecs. We propose a Lagrangian multiplier based penalty-resource metric to be the targeting cost function. To the best of our knowledge, there is few related discussion, in the literature, on providing a criterion to judge the approaches of entropy decoding under resource constraint. The existing approaches which dealt with the decoding of the single-side growing Huffman tree may not be memory-efficient for arbitrary-side growing Huffman trees adopted in current codecs. By grouping the common prefix part of a Huffman tree, in stead of the commonly used single-side growing Huffman tree, we provide a memory efficient hierarchical lookup table to speed up the Huffman decoding. Simulation results show that the proposed hierarchical table outperforms previous methods. A Viterbi-like algorithm is also proposed to efficiently find the optimal hierarchical table. More importantly, the Viterbi-like algorithm obtains the same results as that of the brute-force search algorithm.**

***Index Terms*—Audio/video decoding, Huffman decoding, tree data structure.**

I. Introduction

TRADITIONAL entropy coding in multimedia compression applications, such as JPEG, MPEG-1/2/4, AAC/AAC+, H.264/AVC and Windows Media Video version 9 (WMV9), heavily relies on Huffman-tree based construction codes. Results of [1], [2] shown that even after optimizing the decoding module, entropy decoding still occupies a major portion of decoder's timing profiles. Due to its sequential nature, speeding up entropy decoding, however, is not as straight forward as to speed up the other modules of a decoding process by issuing several independent instructions simultaneously. Therefore, Huffman decoding is one of the important factors that influencing on the overall complexity of a decoding module.

Manuscript received December 02, 2006; revised October 18, 2007. First published March 21, 2008; current version published October 24, 2008. This work was supported in part by the Excellent Research Projects of National Taiwan University, under Grant 95R0062-AE00-02. This paper was recommended by Associate Editor C. N. Taylor.

S.-W. Wang, J.-L. Wu, S-C. Chuang, and C.-C. Hsiao are with the National Taiwan University, Taipei, Taiwan 106 (e-mail: song@cmlab.csie.ntu.edu.tw).

Y.-S. Tung is with Hon-Hai Precision Ind. Company, Ltd., Taipei, Taiwan, and also with National Taiwan University, Taipei, Taiwan 106.

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/TCSVT.2008.920968

One way to speed Huffman decoding up is to find a tree-based representation structure of the given Huffman code and its efficient traversing algorithm. Chung *et al.* [3] proposed an memory-efficient array to present the Huffman-tree (HT) in breadth-first search manner. From left to right, the nodes of the HT are stored in an array of consecutive memory locations, of which the entries are either the return symbols of leaf nodes or the offset addresses of internal nodes to the branch down children nodes. The data structure designed for MPEG-2 AAC, proposed by Lee *et al.* [4], is similar to that of [3]. Chen *et al.* [5] assigned the weight 2^{h-l} to level l leaf nodes of a height h HT and then performed the Huffman decoding by searching the corresponding weight of the given binary codeword. Lin *et al.* [6] transformed the original HT to a single-side growing Huffman-tree (SGH-tree) and presented a memory-efficient data structure for representing and traversing the SGH-tree. Lin *et al.* assigned the logical address, $c_i - \varphi_{l_i} + \sum_{k=2}^{l_i} f_{k-1}$, to each symbol s_i, where c_i is the codeword of symbol s_i, l_i is the level of s_i, φ_{l_i} is the number of internal nodes in level l_i and f_k is the number of leaf nodes of level k. The Huffman decoding of [6] is conducted by searching the logical address of the given binary codeword. This approach might be infeasible for existent HTs because the process of transforming HT to SGH-tree is required. The memory usage of tree-based methods [3]–[6] is very efficient; however, one does not know how many bits should be inspected during decoding. That is, all possible lengths of codewords are required to be inspected during decoding, in the worst case.

Another way to speed Huffman decoding up is to construct a look-up table (LUT) and inspect several input bits, at a time, as the address of the table. This multiple-bit-inspecting nature implies that the LUT-based method needs shorter memory access time than that of the tree-based apporach, in general. Hashemian, in [7], partitioned an HT by clustering several bits together into subtrees and constructed a memory efficient LUT for each subtree; Aggarwal *et al.* [8] partitioned the code symbols into clusters and constructed one LUT for each cluster; Choi *et al.* [9] and Jiang *et al.* [10] partitioned an SGH-tree on the basis of pattern-matching and constructed memory-efficient LUT-based Huffman decoders. In general, when an *HT* is irregularly sparse, the approaches of [7]–[10] will face the memory waste problem because there are many redundant nodes (i.e., redundant table entries). Hashemian also proposed a condensed Huffman table (CHT) for SGH-tree in [11] which partitions

codewords into several sets based on the codeword lengths. The Huffman decoding of [11] is conducted by performing a sequential search on the CHT. The problem CHT approach faced similar to that of [6], that is, when decoding actual data the process of transforming exist HTs to SGH-trees is required. A binary search based method was proposed by Wang *et al.* in [12], where codewords were extended to a unique length (i.e., the depth of the HT). These expanded codewords partition the contiguous table address into disjoint intervals. Wang *et al.* recorded only the starting addresses of the disjoint intervals and performed the Huffman decoding through binary search upon these starting addresses. The problem of binary-search-based method is that reaching higher probability codewords needs more memory access time than that of lower probability ones. The aforementioned approaches, in short, can customize a single SGH-tree based Huffman decoder well in terms of memory efficiency and required execution speed.

Nevertheless, most designs of nowadays video/audio codecs have masses of HTs, e.g., AAC has 12 HTs, AAC+ has 12 HTs for spectral band replication (SBR), MPEG-2 video has 14 HTs, MPEG-4 has 24 HTs, H.264/AVC has 29 HTs and WMV-9 has 86 HTs. More importantly, the structures of these HTs are sometimes arbitrary-side-growing instead of single-side-growing. By taking all HTs of a specific codec into account under certain resources constraints, such as memory space or power consumption, direct SGH-tree based partitioning becomes infeasible because: 1) the independency of each direct SGH-based partitioning may result in a memory requirement larger than the total memory constraint, and 2) the corresponding HTs are often complicated and arbitrary-side growing, the approaches which are designed on the basis of SGH-tree may not work efficiently enough. That is, the execution speed may not meet the application need. Therefore, to deal with the first issue, an automatic join partitioning approach for all given HTs (under certain resource constraint) is investigated. We first propose a Lagrangian multiplier based penalty-resource metric in Section II for compromising the processing efficiency and the memory usage. With the proposed metric, any optimal solution algorithm for finding a Lagrangian multiplier can be adopted. As to the second issue, [13] provided two hierarchical approaches for partitioning any arbitrary-side growing HT effectively in memory usage. Since optimizing a hierarchical partition plays a crucial role in partitioning an arbitrary-side-growing Huffman tree (AGH-tree), in Section III, we develop a Viterbi-like algorithm for searching optimal hierarchical structures in AGH-trees. The processing efficiency and memory usage of our approach are then experimentally compared with those of the conventional approaches in Section IV. Finally, concluding remarks are made in Section V.

II. Penalty-Resource Metric

Intuitionally, for any specific codec, a Huffman decoding module (HDM: a Huffman decoder which has to deal with more than one HTs) usually possesses more resource than a simple Huffman decoder (which deals with only one HT) and better coding performance (lower penalty) is also expected. For example, if we construct a LUT based on the longest bit length of an HT, $l_{\max}$, the memory access time is only one unit but the corresponding memory size becomes exponentially huge, i.e., $2^{l_{\max}}$. Obviously, there is a tradeoff between the resource and the penalty. A meaningful metric for designing good HDM should consider the penalty and the resource constraint at the same time.

A. Penalty-Resource Modeling

An appropriate modeling of an HDM should honestly reflect the relationship between the penalty (e.g., the number of memory access or decoding speed) and the resources requirement (e.g., the amount of table size) for fulfilling the application needs. However, the realization of an application is both algorithm and technology dependent, which means a precise relationship between penalty and resource requirement is hard to be defined.

First, from the algorithmic point of view, the logarithmic function seems reasonable to reflect the relationship between the penalty and the resource requirement. This is because both LUT-based and binary-search-based algorithms conducting multiple bits inspection at one time, and therefore, the required memory size increases exponentially with respect to the number of inspection bits. Second, from the realization technology point of view, the radical squared expression seems useful for reflecting the prescribed relationship. This comes from the fact that the capacity of a memory, for an embedded system, can be mapped to the width of a chip area which decides the latency of a random access. In general, for applications without resource constraints, the equivalent relation between penalty and resource requirement changes with applications and may not be linear.

Fortunately, for applications with resources constraints, the Lagrangian Multiplier technique can naturally capture the equivalent relation between the penalty and the resources requirement. This is because for ensuring the formulation of Lagrangian multiplier based cost function meaningful, both the penalty and the resource constraint should be monotonic functions of the non-negative parameter λ [14]. Therefore, mathematically, the specific relationship between penalty and resource constraint can be represented as an appropriate function of λ. This explains why, as represented in usual Lagrangian formulations, there is no scaling and weighting criteria involved in our Penalty-Resource modeling for HDM, as illustrated in the following discussions.

B. Penalty-Resource Modeling for HDM

Suppose an HDM has n HTs and there are in total I possible fulfilled partitions. Let f_i denote one of the possible fulfilled partitions composed by σ_{ij}, $j = 1 \ldots n$, that is

$$f_i \triangleq \overline{\sigma_i} = (\sigma_{i1} \circ \sigma_{i2} \circ \cdots \circ \sigma_{\text{in}}) \tag{1}$$

where $\overline{\sigma}_i$ denotes the overall partition for the ith fulfilled partition, σ_{ij} denotes the jth component (partition method) of $\overline{\sigma}_i$,

$i = 1, 2, \ldots, I$. Consequently, we define the penalty and the resource of a fulfilled partition for an HDM, f_i, respectively, as

$$P(f_i) = 1 + \sum_{j=1}^{n} p(\sigma_{ij}) \tag{2}$$

and

$$R(f_i) = n \times entry\_size + \sum_{j=1}^{n} r(\sigma_{ij}). \tag{3}$$

To meet the memory space constraint, the resource requirement here is defined linearly proportional to the memory size. $p(\sigma_{ij})$ in (2) denotes the required number of memory access for the partition σ_{ij} and is measured by counting the memory access times $p_{\sigma_{ij}}$ weighted by a factor u_i, that is

$$p(\sigma_{ij}) = u_i p_{\sigma_{ij}}. \tag{4}$$

Similarly, $r(\sigma_{ij})$ in (3) denotes the required memory size $r_{\sigma_{ij}}$ and is measured by counting the memory usage of the partition σ_{ij} weighted by a factor v_i, that is

$$r(\sigma_{ij}) = v_i r_{\sigma_{ij}}. \tag{5}$$

To locate each σ_{ij} onto the final memory bank, we need one extra table, with size of $n \times entry\_size$ to memorize the root addresses of each σ_{ij} in the memory bank. The cost of this extra table access results in the term '1' in (2).

C. Lagrangian Multiplier Optimization

The Lagrangian technique could be applied to discrete samples without loss of optimality if a solution exists in the convex hull that meets the resource constraint [15], [16]. The first issue mentioned in Section I can now be modeled as: given a resource constraint R_c, find the most appropriate partition f^* such that

$$R(f^*) \leq R_c \tag{6}$$

and

$$P(f^*) \tag{7}$$

is minimized.

Lagrangian multiplier techniques can naturally be applied to solve this kind of problems. Thus, we define the penalty-resource (P-R) cost of f_i as

$$C(f_i) \triangleq P(f_i) + \lambda(R(f_i)) = \sum_{j=1}^{n} p(\sigma_{ij}) + \lambda \left(\sum_{j=1}^{n} r(\sigma_{ij}) \right). \tag{8}$$

The problem becomes, for a given $\lambda \geq 0$, finding the solution f^* which minimizes $C(f_i)$. The theorem given in [16] illustrates that the above constrained problem (cf. (6) and (7)) can be solved as a unconstrained problem with a certain nonnegative λ (cf. (8)). Let $P^*(\lambda)$ and $R^*(\lambda)$, respectively, be the penalty and the resource constraint corresponding to f^* under the given λ. To obtain a correct λ under the given resource constraint, several approaches have been provided [16], [17]. A simple strategy to find λ is using iterative bisection search algorithm[16] which makes $R^*(\lambda)$ toward R_c. Furthermore, since each σ_{ij} is independent of f_i, to solve

$$\min \left(\sum_{j=1}^{n} p(\sigma_{ij}) + \lambda \sum_{j=1}^{n} r(\sigma_{ij}) \right) \tag{9}$$

is equivalent to solve

$$\sum_{j=1}^{n} \min(p(\sigma_{ij}) + \lambda r(\sigma_{ij})). \tag{10}$$

That is, we can separately find σ_{ij} which minimizes

$$p(\sigma_{ij}) + \lambda r(\sigma_{ij}) \tag{11}$$

under the given λ.

With the aid of the P-R cost function, the join penalty-resource optimal solution for an HDM with multiple HTs can be found systematically. However, for discrete samples, the drawback of the above approach is that the optimal f^* may not be reachable if Lagrangian techniques are used directly. Fortunately, if the convex hull of the P-R cost function is dense enough, the gap between the best achievable and the optimal solutions would be small. In the next section, we will develop a novel partitioning method which is believed to reach a tiny neighborhood of the optimal solution.

III. Hierarchical Arbitrary-Side Growing Table (HASGT)

In this section, we break an HDM into discrete samples (i.e., Huffman trees, T_i) and focus on optimizing the hierarchical partition of each sample (i.e., finding each σ_{ij}). We first introduce two partition methods to cluster an HT into several LUTs: the modified Hashemian cut (MHC) and the bits-pattern-Xor (Bpx). These two methods are motivated by the Hashemian's method [7] and the bits-pattern matching scheme [9], [10], respectively. Hahemian's method [7] uses fixed length clustering and a supper tree to memories the location of individual subtree. MHC provides more flexible ways for constructing LUTs, which clusters the common length symbols together [cf. Fig. 1(b)]. MHC behaves very similar to Hahemian's method except the length of clusters is varying, and also incorporates a supper tree into LUTs. The scheme given in [9], [10] simply counts the leading 1's of symbols, so it is inefficient for an AGH-tree; therefore, a better memory usage can be expected. Bpx is a LUT that indexes each symbol by the number of leading common bits with a certain bits-pattern (cf. Fig. 1(c)).

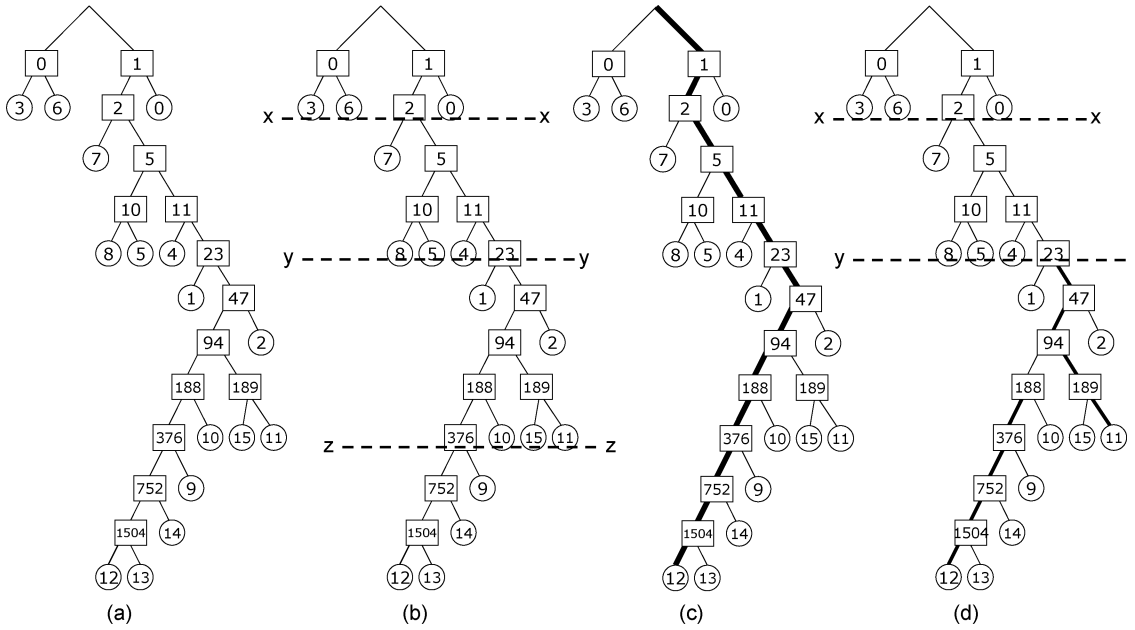

Fig. 1. Rectangular boxes represent the codeword values and the circle nodes represent the values of symbols. (a) Huffman tree with 16 symbols. (b) Example of applying the MHC method. (c) Example of applying the Bpx method in which the boldface line represents one of the bits-patterns. (d) Example of HASGT clustering.

More specifically, as shown in Fig. 1(a), we take a Huffman tree [18],[1] T, as an example to illustrate the MHC and the Bpx schemes. MHC partitions a tree by cut-lines. As shown in Fig. 1(b), T is partitioned by three cut-lines: x-x, y-y, and z-z. There is a LUT for each cluster which memorizes every value of symbols and every code length of symbols within the cluster. MHC introduces memory waste because it duplicates the symbols that are not of the same length as the cluster size. For example, symbol 7 in Fig. 1(b) will be duplicated four times in the second cluster. For a Bpx, we index symbols by counting the largest number of common leading bits of a given bits-pattern. As an example, for the bits-patterns $(1001)_b$ and $(1011)_b$ the largest number of common bits (LCB) is two. In practice, the above method can be realized by counting the number of leading zeros after bitwise XORing with a given bits-pattern. Consider the example in Fig. 1(a), given the bits-pattern "$(101111000000)_b$" [as shown in the boldface line in Fig. 1(c)], if we cluster all symbols of T by Bpx, the LCB of {3,6}, 0, 7, {8,5}, 4, 1, 2, {15,11}, 10, 9, 14, and {12,13} are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11, respectively. Likewise, we need small LUTs to distinguish the symbols that are of the same number of LCB, such as {3,6}, {8,5}, {15,11}, and {12,13}. Due to the hardware consideration, the instruction for counting the leading zeros and bitwise-XORing are limited in length, we can not apply any length Bpx onto a Huffman tree haphazardly. Meanwhile, if the symbols are equal probable, the codelengths of them are the same after Huffman code construction. MHC performs better than Bpx does, in this case. In the next section, we will integrate the two partition methods, MHC and Bpx, together to provide an even better Huffman tree partitioning.

[1]This Huffman tree associating with a Huffman table is used in Windows Media Video 9, which estimates the probabilities of occurrence of different transform types at high bit-rate.

A. Structure of an HASGT

Due to the difference of tree growing tendencies (single-side v.s. arbitrary-side), the preference of MHC and Bpx is tree structure dependent. Basically, MHC is suitable for near-full trees while Bpx behaves better for more-sparse trees. As an example, T in Fig. 1(d) is first cut by an MHC with length-2, because there is a length-2 full subtree at the top of the Huffman tree. The remainder of T beyond the cut-line, $x - x$, forms another subtree t_2. As for t_2, however, there is no apparent partitioning method. Because, if t_2 is again partitioned by an MHC with a longer or shorter length, the memory waste problem or the memory access time increasing problem will be introduced, respectively. Conversely, if t_2 is partitioned by a Bpx with bits-pattern $(1111)_b$, the memory waste problem is solved but there still needs an extra memory access for t_{10}. In brief, the selection of partitioning method for this example [cf. Fig. 1(d)] is only heuristics-based and, more importantly, the total memory access or memory size usage does not guarantee reaching to the optimal solution. A compact data structure to represent HASGT and the corresponding discussions can be found in [13].

As a prescribed, the heuristics-based approaches can not give apparent criteria to construct the optimum HASGT; moreover, manually partition HT is not applicable for mass HTs and may lead to erroneous judgment. Therefore, we propose an optimal HASGT construction method in Section III-C based on the cost function associated with an HASGT, presented in Section III-B. Before defining the cost function, we first express several terminologies for simplifying the following discussions.

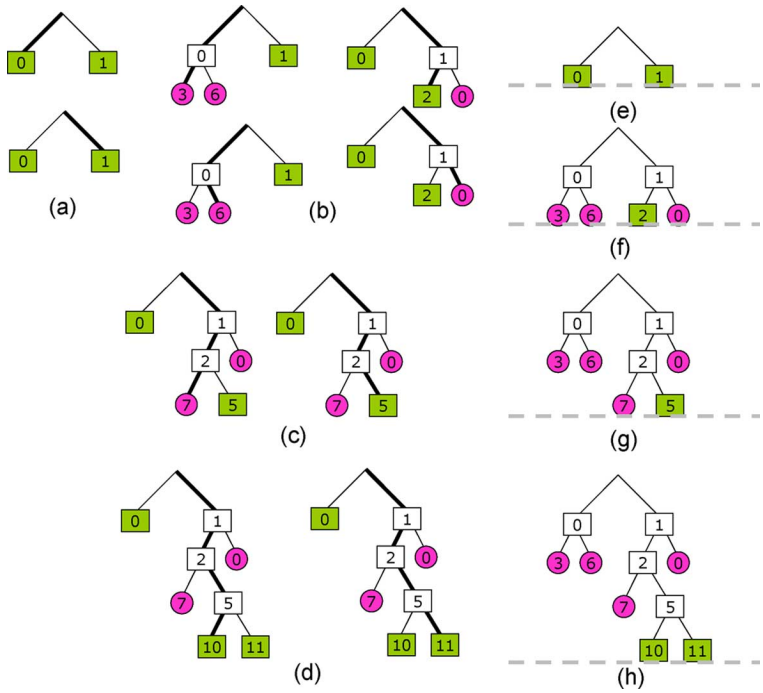

Fig. 2. Referring to Fig. 1(a), there are totally fourteen possible ways to partition the tree from the root under the condition that the length of instruction is 4 bits. The bold lines indicate the paths corresponding to various bits-patterns. The dotted lines represent cut-lines. The green boxes are roots of subtrees in which the partitioning methods are switched. The pink circles are symbols in the current cluster. (a)–(d) There are 2, 4, 2, and 2 possible ways for conducting Bpx partitions in which the bits lengths are one, two, three and four, respectively. (e)–(h) Clusters corresponding to one, two, three and four bits lengths and represent MHC-1, MHC-2, MHC-3 and MHC-4, respectively.

As shown in Fig. 3(a), MHC X'–X' line partitions T, which results in a subtree, t_5. MHC Y–Y line partitions t_5 introducing another subtree t_{23}. Bpx $(1011)_b$ partitions t_{23} inducing the subtree t_{188}. Finally, t_{188} is partition by Bpx $(0000)_b$ and no more subtree existed. The available *first-order* partitions of t_k are viewed as a set of operations. Referring to Fig. 2, for instance, there are 14 first-order operations, *op*'s, for the HT given in Fig. 1(a).

Since the set of *op*'s varies in t_k's, we denote $\overline{OP_k}$ as the set of *op*'s for t_k. In previous sections, the symbol σ_{ij} is needed to specify the ith sort of HASGT associated with T_j. The σ_{ij} is composed of a series of *op*'s:

$$\sigma_{ij} = (op_{ij,k}, \ldots, op_{ij,q}) \tag{12}$$

where $op_{ij,k}$ is the *op* for partitioning t_k associated with σ_{ij}. Normally, k begins from zero (i.e., the root of T).

Next, for discussing the cost of σ_{ij}, we construct a *Hierarchical – tree* (H-tree) related to each σ_{ij} by the following steps:

---

step-(1) Input: A subtree, t_k; Output: a Structure s_k

a) Select an *op* from $\overline{OP_k}$.

b) s_k = the set of

i) t_k, t_q where t_q is the root of subtrees which are out of the current *op* cluster

ii){0}, the singleton (or tree-root), if k equals to zero

step-(2) : For each t_q in s_k, Input according to t_q in step-(1)

Steps (1) and (2) are processed recursively until all symbols are included. And the resultant H-tree is the union of all s_k.

B. The Measurement of HASGT

Following the discussions given in Section II-C, we define the cost of t_k as

$$c_k = p_k + \lambda r_k. \tag{13}$$

The penalty, p_k, and resource constraint, r_k, of t_k are decided by the current choice of *op* and the cost of its associated subtrees, thus the penalty of t_k partitioned by *op* would be

$$p_{k,op} = prob(t_k) \times penalty(op) + \sum_{q \text{ in } s_k} p_q \tag{14}$$

in which $prob(t_k)$ is the probability associated with t_k, and *op* partitions t_k and induces $penalty_{k,op}$. The suffix k, op is decided by $\sigma_{ij,k}$. p_q which denotes the penalty of out-of-cluster root node t_q. The resource $r_{k,op}$ of t_k associated with *op* becomes

$$r_{k,op} = resource_{k,op} + \sum_{q \text{ in } s_k} r_q \tag{15}$$

where $resource_{k,op}$ represents the required LUT size for the partition *op* of t_k. Similarly, r_q denotes the resource requirement

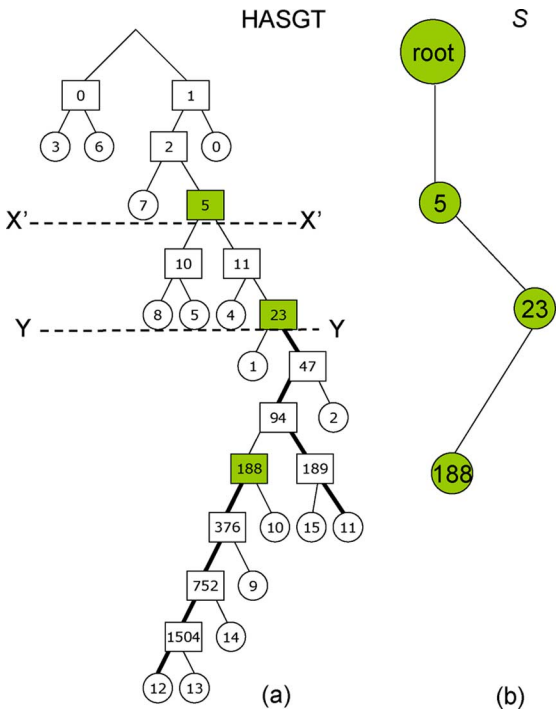

Fig. 3. (a) HASGT which is composed by t_5, t_{23} and t_{188}. (b) H-tree corresponds to Fig. 3(a).

t_q. Consequently, the P-R cost of t_k with choosing *op* from $\overline{OP_k}$ becomes:

$$\begin{aligned} c_k &= p_k + \lambda r_k \\ &= prob(t_k) \times p_{k,op} + \sum_{q \text{in } s_k} p_q + \lambda \left(r_{k,op} + \sum_{q \text{ in } s_k} r_q \right) \\ &= prob(t_k) \times p_{k,op} + \lambda r_{k,op} + \sum_{q \text{in } s_k} p_q + \sum_{q \text{ in } s_k} c_q. \end{aligned} \quad (16)$$

Notice that c_k for $k = 0$ is the same as the one described in (11). Our goal is to find an HASGT, σ_{ij}, such that the cost of hierarchical partition of T is minimal, that is

$$\min_{op} c_0. \quad (17)$$

Since manually solving hierarchical partition is tedious and time consuming; moreover, the so-obtained results may not be the best one. For automation, the brute-force top-down searching all possible partitions is, of course, an intuitive solution. Nevertheless, brute-force approach can not find the optimal solution efficiently. We will develop a fast optimizing HASGT method in the next section.

C. Viterbi-Like Optimizing HASGT Method

From (16), we observed that c_k is decided by *op*'s taken from $\overline{OP_k}$ and the precalculated cost c_q. Those c_q's which are not of minimal values can not meet to goal of minimizing c_k, that is, the corresponding set of *op*'s (denoted as $\overline{OP_q}$) could be omitted during optimization just like the tree pruning operations in the well-known Viterbi algorithms. In other words, one can calculate the initial cost (i.e., the cost of the first-order partition) and the corresponding minimal c_q, and then the problem of searching for optimal cost HASGT can be converted to subproblems of searching for minimal cost of each t_k.

According to the above inference, we propose a Viterbi-like algorithm to find the optimal HASGT. We adopt a bottom-up approach which traverses all internal subtrees, t_k. As prescribed, those *op*'s associated with the minimal P-R cost, in each $\overline{OP_k}$, will be selected as the best partitions for each t_k while those *op*'s which are not of the minimal cost will be discarded, during each searching stage. The subtree, t_k, can immediately determine a minimal *op* from $\overline{OP_k}$, once every internal subtree, t_q, behind the current t_k has found the corresponding *op*'s which are of minimal cost. Since we never know the actual probability of each t_k, we assume the $prob(t_k)$, in (16), is related to its code length. In other words, the $prob(t_k)$ is simply assigned as $(1/2)^{\text{codelength}}$. Then, the optimal structured HASGT could be constructed through a Viterbi-like algorithm with the parameters of the pre-defined $prob(t_k)$ and the λ obtained from an iterative bisection search, described in Section II-C.

D. Complexity Analysis

To show the superiority of our work, the construction complexity of the brute-force approach and that of the proposed method have been derived; however, due to the page length limit, we post the detail derivation on our web-site [19]. We summarize the derived result in the following. For a tree with depth n, let A_n be the complexity of constructing HASGTs, H_n be the complexity of constructing hierarchical MHC partitions, and let B_n be the complexity of constructing hierarchical Bpx

partitions. Then, as shown in [19], A_n can be formulated by H_n and B_n as

$$A_n = \sum_{i=1}^{n} (B_i + H_i)(A_{n-i})^{2^i}. \tag{18}$$

Basically, the concrete construction complexity depends on the structure of target HT. The constructing complexities for two extreme cases, the *perfect binary tree* (PBT) and the SGH-Tree, are derived in detail in [19].

For the case of PBT, it can be proved that

$$A_n \geq 2^{2^n-2+2^{n-1}} + 2^{2^{n-2}} \tag{19}$$

while the complexity of the proposed method, $\widehat{A_n}$, becomes

$$\widehat{A_n} = 2n \cdot 2^n - n. \tag{20}$$

Similarly, for the case of *SGH-Tree*, we have

$$A_n \geq 2 \cdot 3^{n-1} + 2^{n-1}. \tag{21}$$

and

$$\widehat{A_n} = \frac{3n(n+1)}{2}. \tag{22}$$

For both cases, we obtain that

$$\widehat{A_n} \ll A_n. \tag{23}$$

IV. Experimental Results

In this section, we focus on two subjects: 1) illustrating the benefit obtained from the prescribed metric and 2) comparing HASGT with previous well-known methods M1 [3], M2 [4], M3 [5], M4 [6], M5 [7], M6 [8], M7 [11] and M8 [12]. The first subject is investigated in Section IV-A, and the second subject is discussed from two perspectives: the complexity analysis and the performance comparison when decoding the actual data, which are presented in Sections IV-B and C, respectively.

To provide practically realistic experimental results, all experiments are conducted for various Huffman tables of video/audio codecs including: 12 HTs of AAC and 12 HTs of AAC+ SBR audio codecs, 14 HTs of MPEG-2, 24 HTs of MPEG-4, 29 HTs of H.264/AVC and 86 HTs of WMV-9 video codecs. Meanwhile, the performance evaluation is measured by the metric mentioned in Section II-B. The probability of each HT [i.e., u_i in (4)] is the average of actual appearance in the conformance sequences[20]–[22]. The number of conformance sequences for testing AAC, AAC+, MPEG-2, MPEG-4, H.264/AVC and WMV9 are 62, 47, 35, 20, 74 and 5, respectively. The resource is defined to be the actual memory size (i.e., v_i in (5) is set to be one) and the penalty is counted by the weighted numbers of memory access for all code symbols. The memory access is defined as the number of times to access the proposed data structure. The weighted factor is the actual frequency of occurrence for each symbol which is obtained directly from the conformance sequences. It is reasonable to define the number of memory access (NMA) as the penalty because NMA plays a major role in most on-chip designs. Our simulation platform is Intel Core Duo processor with clock speed of 1.8 GHz. All programs are coded in C++ language and executed under the Windows XP operating system.

A. The Optimization of Hashemian's Method

Hashemian's method proposed in [7] did not provide efficient algorithm of choosing the cluster bit lengths for mass HTs. The cluster bit-length used in [7] is a heuristic-determined value which might not be suitable for all cases. That means better performance of Hashemian's method can be expected. To show that the metric described in Section II is useful for searching optimal solution for different Huffman decoding methods, we demonstrate that the performance of [7] can be further improved by using Lagrangian multiplier technique with the aid of the proposed metric. We assume the probability of appearance of each symbol in an HT is related to its code length as before. After substituting the probability into the proposed metric, the expected optimal cluster length of each HT can be obtained through Lagrangian multiplier based searching technique. In the following, the optimized Hashemian's method is evaluated by using the conformance sequences [20]–[22] again. The conducted experiments include Hashemian's method with fixed bit-length 3 (H-3), Hashemian's method with fixed bit-length 7 (H-7) and Hashemian's method with varied bit-length (H-varied), which ranges from 3 to 7 bits. To verify that the performance of [7] is improved by using the so-obtained optimal cluster lengths, H-3 and H-7 are chosen as the baseline for comparison. Due to the page limit, only the experimental results for MPEG-2 is shown in Fig. 4. Experimental results for other codecs can be found in our web-site [23]. As prescribed, the penalty used in our experiments is the average number of memory access of the conformance sequences and the memory size is the actual memory required for all 14 MPEG-2 HTs. The curve of H-varied in Fig. 4 represents the results after optimization. The least memory size required for H-varied is smaller than that of H-3. The least memory access time of H-varied is the same as that of H-7 while the memory size of H-varied is smaller than that of H-7. For other codecs, we summarize the performance of the method that requiring the least memory size (denoted as H-$\lambda = INF$) and method that requiring the least memory access (denoted as H-$\lambda = 0$) in Table I. In general, the performances of H-$\lambda = INF$ and H-$\lambda = 0$ are better than that of H-3 and H7, respectively, in terms of the memory size (MS)

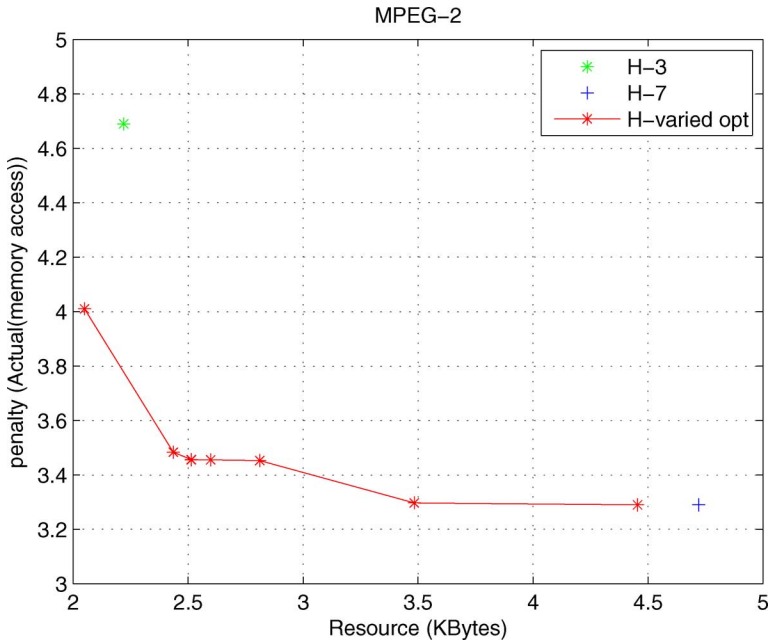

Fig. 4. For an (MPEG-2)-HDM with 14 HTs, the green star point shows that the total resource used and the average memory access required, for H-3 bits length cluster, are 2224 bytes and 4.69 units, respectively. The P-R curve is produced by combining clusters of Hashemian methods, with bit lengths 3, 4, 5, 6 and 7, under different resource constraints. The smallest resource size is 2050 bytes and the corresponding averaged memory access is 4.01 units. The least averaged memory access is 3.29 units and the corresponding memory size is 4410 bytes. The averaged memory access of H-7 cluster is the same as that of the H-varied cluster but the memory requirement of H-7 is 4716 bytes which is lager than that of the H-varied one.

TABLE I
PERFORMANCE IMPROVEMENT OF HASHEMIAN'S METHOD [7]

	AAC		AAC+(SBR)		MPEG-2	
Method	MS (KB)	MA	MS (KB)	MA	MS (KB)	MA
H-3bits	6.7	5.12	3.83	3.12	2.22	4.69
H-$\lambda = INF$	5.97	3.91	3.61	3.08	2.05	4.01
H-7bits	9.24	3.41	9.56	3.02	4.72	3.29
H-$\lambda = 0$	9.24	3.41	6.94	3.02	4.41	3.29

	MPEG-4		H.264/AVC		WMV9	
Method	MS (KB)	MA	MS (KB)	MA	MS (KB)	MA
H-3bits	4.62	5.66	1.82	3.50	26.22	4.37
H-$\lambda = INF$	4.02	4.64	1.71	3.33	25.28	3.82
H-7bits	12.48	3.59	3.79	3.04	47.37	3.16
H-$\lambda = 0$	10.4	3.59	3.79	3.04	46.54	3.16

and the number of memory access (MA). In short, from Fig. 4 and Table I, the performance of Hashemian's method, indeed, can be further improved by using the proposed P-R metric.

B. Complexity Analysis of Different Methods

The objective of this section is to compare the complexity of the previous methods [3]–[8], [11], [12] and that of HASGT in terms of MA and MS. Moreover, to depict a clear picture, the cost of structure construction (CT) is also included. Basically, the ideal scheme should be the one with little average memory access, small memory space and short construction time. We show the results of comparison for the different methods in Table II. Since one does not know how many bits should be inspected during decoding when tree-based approaches: M1 [3], M2 [4], M3 [5] and M4 [6] are used, the worst MA's will be linearly proportion to the longest codelength of the HT, N. Therefore, the corresponding MA complexity of tree-based methods M1, M2, M4 is $O(N)$. M3 needs another $O(\log n)$ process for each given codeword, so its MA complexity is $O(N \log n)$. On the other hand, tree-based methods store nearly all leaf nodes and internal nodes of an HT, so the complexity of memory usage is $O(n)$, where n is the number of symbol in an HT. The construction of a tree-based representation needs to traverse the given HT, so the CT complexity is $O(n)$.

TABLE II
COMPLEXITY COMPARISON AND ACTUAL CONSTRUCTION TIME

Scheme	Worst MA	MS	CT	WMV9 sec.
M1 [3]	$O(N)$	$O(n)$	$O(n)$	0.13
M2 [4]	$O(N)$	$O(n)$	$O(n)$	0.28
M3 [5]	$O(N \log n)$	$O(n)$	$O(n)$	0.19
M4 [6]	$O(N)$	$O(n)$	$O(n)$	0.09
M5 [7]	$O(N/c)$	$O(2^c N/c)$	$O(2^c N/c)$	0.31
M6 [8]	2	$O(2^N)$	$O(2^N)$	0.14
M7 [11]	$O(N)$	$O(n)$	$O(n)$	0.03
M8 [12]	O(log2N)	$O(n)$	$O(n)$	0.17
HASGT	O(N/c)	$O(2^c N/c)$	$O(2N2^N i)$	57.6

The LUT-based methods, M5 [7] and M6 [8] inspect several input bits at a time, so the symbol searching time is proportional to the number of required inspections. If M5 inspects c bits at a time, then N/c inspections are needed in the worst case. M6, first, inspects N bits as an index and then performs table lookup for locating the correct symbol according to the so-obtained index. The MA complexity of M6 is, thus, only 2. M5

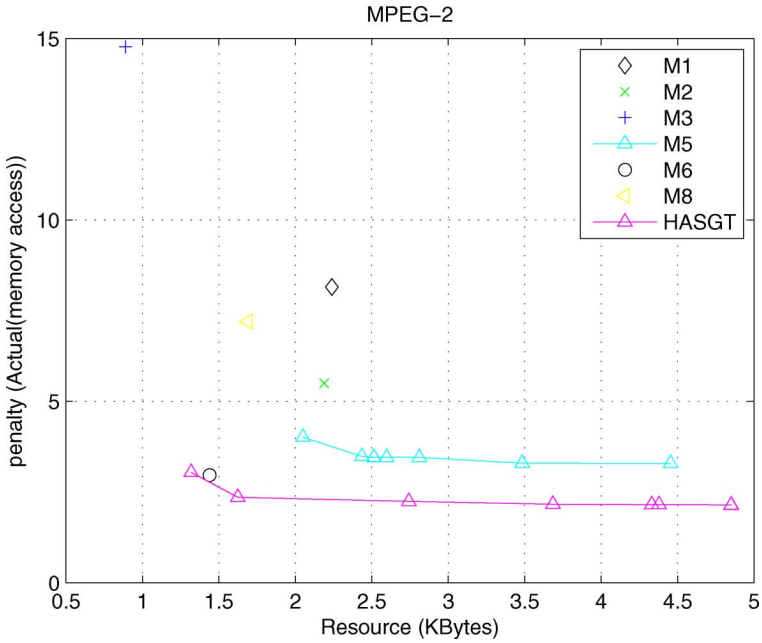

Fig. 5. For MPEG-2 with 14 HTs, the memory sizes of M1, M2, M3, M6 and M8, respectively, are 2.24, 2.19, 0.89, 1.44 and 1.69 KB and the corresponding memory access times are, respectively, 8.15, 5.50, 14.77, 2.97 and 7.20 units. The smallest memory size and the corresponding memory access time of M5 are 2.05 KB and 4.01 units, respectively. The least memory access time and the corresponding memory size of M5 are 3.29 units and 4.41 KB. The smallest memory size of HASGT is 1.32 KB and the corresponding memory access time is 3.04 units. The least memory access time of HASGT is 2.14 units and the corresponding memory size is 4.85 KB.

needs to construct 2^c entries for every cluster and M6 needs to construct 2^N entries in the worst case. Thus, CT complexities of M5 and M6 are $O(2^c N/c)$ and 2^N, respectively. The MS complexity of M5 increases exponentially with c, there are totally N/c clusters and each has 2^c entries. Hence, the MS complexity of M5 is $O(2^c N/c)$. Similarly, M6 needs 2^N entries in the worst case, thus, its MS complexity is $O(2^N)$. M7 [11] performs a sequential search on CHT. The number of entries in CHT is linearly proportional to the number of various codelengths, so the MA complexity of M7 is $O(N)$. The memory usage of M7 depends directly on the numbers of symbols and different codelengths, so its MS complexity is $O(n+N)$. The construction of M7 is proportional to the number of entries in CHT, so its CT complexity is $O(N)$. M8 [12] records the starting addresses of symbols, thus both of its MS and CT complexities are $O(n)$. M8 performs binary search on the proposed structure, so its MA complexity is $O(\log n)$.

Both MA and MS complexities of HASGT are similar to those of the LUT-based method M5. However, the c used in M5 is a heuristic-determined value which could affect both the performances of MA and MS. We have proposed an algorithm, described in Section III-C, to find out the appropriate c under a given resource constraint. The construction complexity of HASGT depends on the required number of iterations, i, for searching an appropriate Lagrangian Multiplier and the complexity of the proposed Viterbi-like algorithm. Thus, the CT complexity of HASGT is $O(2N2^N i)$. The CT complexity of HASGT is much higher than that of the other methods. However, the constructed data structure can be reused all the time because the given HTs should not changed over time. Thus, it is reasonable to spend more time for construction when better performance can be provided. Of course, for applicability, the actual construction time should be within an acceptable range. The actual time required for constructing the 86 WMV9 HTs for each of prescribed methods is listed in Table II. We apply cluster length 4 for M5 as used in [7]. The constrained memory size of HASGT for WMV9 is targeted at 32 KB, and in total 11 iterations are required for finding the optimal partition, in our experiment. The time required for the process of transforming HT to SGH-tree, as prescribed, is not considered in our experiments. From Table II, the actual construction time for HASGT is acceptable. In addition to the prescribed complexity comparison, the performance measurement for decoding actual data set is of equal or even higher importance. In the next section, we will focus on the performance of decoding an actual data set.

C. The Performance Comparison With Actual Data Set

The objective of this subsection is to compare the performance of HASGT, upon actual data set, with that of previous works: M1 [3], M2 [4], M3 [5], M5 [7], M6 [8], and M8 [12]. For M4 and M7, the process of transforming given HTs to SGH-trees needs to change the processing stages within both encoder and decoder sites which is infeasible for conformance sequences, so M4 and M7 are not included in our experiments. For HASGT, we limit the instructions for counting LCB (i.e., for Bpx) and pre-fetching (i.e., for MHC) to 7 bits in length. The Lagrangian multiplier λ is set to an arbitrary large value so that the least memory size of specific codec, required by HASGT, is achieved. When λ is set to zero, the least number of memory access is also achieved.

TABLE III
PERFORMANCE COMPARISON UNDER ACTUAL DATA SET

	AAC		AAC+ (SBR)		MPEG-2	
Method	MS (KB)	MA	MS (KB)	MA	MS (KB)	MA
M1 [3]	5.59	7.27	2.54	3.79	2.24	8.15
M2 [4]	5.4	5.83	2.38	2.88	2.19	5.5
M3 [5]	2.49	17.26	1.23	7.17	0.89	14.77
M5 $\lambda = INF$ [7]	5.97	3.91	3.61	3.08	2.05	4.01
M5 $\lambda = IMM$ [7]	6.51	3.63	4.54	3.02	4.22	3.29
M5 $\lambda = 0$ [7]	9.24	3.41	6.94	3.02	4.41	3.29
M6 [8]	3.77	3	2.34	3	1.44	3
M8 [12]	4.77	7.93	2.34	6.74	1.69	7.2
HASGT $\lambda = INF$	4.41	3.08	2.02	2.08	1.32	3.04
HASGT $\lambda = IMM$	4.72	2.35	2.27	2.01	1.62	2.36
HASGT $\lambda = 0$	7.88	2.2	6.45	2.01	4.85	2.14

	MPEG-4		H.264/AVC		WMV9	
Method	MS (KB)	MA	MS (KB)	MA	MS (KB)	MA
M1 [3]	4.26	11.08	1.68	4.68	26.46	6.87
M2 [4]	4.13	6.9	1.75	3.62	25.88	5.13
M3 [5]	1.71	18.19	0.8	6.21	12.41	12.76
M 5 $\lambda = INF$ [7]	4.02	4.64	1.71	3.33	25.28	3.82
M 5 $\lambda = IMM$ [7]	8.58	3.59	3.46	3.05	27.17	3.16
M5 $\lambda = 0$ [7]	10.4	3.59	3.79	3.04	46.54	3.16
M6 [8]	55.52	3	1.51	3	24431.68	3
M8 [12]	3.29	7.31	1.5	4.45	24.08	6.9
HASGT $\lambda = INF$	3.03	3.51	1.18	2.3	14.57	3.02
HASGT $\lambda = IMM$	3.76	2.54	1.37	2.12	18.56	2.17
HASGT $\lambda = 0$	12.48	2.29	4.58	2.02	59.85	2.08

Referring to Fig. 5, for 14 MPEG-2 HTs, the least memory size of HASGT is smaller than that of M1, M2 and M8 while the actual number of memory access is much less than that of M1, M2 and M8. For the least memory size, HASGT needs 48% more memory size than that of M3 but gains 4.86 times in execution speed as compared with M3. M6 partitions HT through the positions of first bits change (i.e., the first positions from 0 to 1 and 1 to 0). M6 can be analyzed from the number of zeros before one and/or ones before zero (i.e., 0_x1 and/or 1_x0) which are subsets of Bpx. When $\lambda = INF$, HASGT prefers to select Bpx partition for saving memory space. Therefore, the performance of M6 is nearly the same as that of HASGT for MPEG-2 codec. However, in the worst case, the memory requirement of M6 could be very large, as shown for the cases of MPEG-4 and WMV9 in Table III. Under different resource constraints (i.e., different Lagrangian multipliers), the performances of the optimal M5 shown in Section IV-A are compared with those of HASGT. For all cases with resource constraints, HASGT based approach outperforms Hashemian method significantly. Notice that the convex P-R curves of HASGT converge to two memory accesses and this phenomenon can be explained as: one of the access is for accessing the address of each LUT and another is for locating the symbol within the LUT. This is also the reason why the number of memory access of M6 is 3.

Due to the page limit, we only illustrate the performance for MPEG-2 codec in Fig. 5. Similar experimental results for other codecs can be found in our web-site [23]. For other codecs, we summarize the performance comparisons among various methods in Table III. Since we can not list all results under different resource constraints, only the least memory size (denoted as $\lambda = INF$), the least number of memory access (denoted as $\lambda = 0$) and the tradeoff result (denoted as $\lambda = IMM$) are included in Table III. The tradeoff result is obtained from the outcome of the first iteration when bisection Lagrangian Multiplier search is used. In general, when $\lambda = IMM$, the performances of HASGT are better than those of M1, M2, M5 and M8 in terms of memory size and the number of memory access. As compared with M3, in average, HASGT needs 55% more memory size to gain 4.33 times in execution speed. The performances of M6 are similar to those of HASGT in the least memory sizes for AAC, AAC+ (SBR), MPEG-2 and H.264/AVC. However, from Table III, the memory size of M6 is huge for MPEG-4 and WMV9.

V. CONCLUSION

The Huffman-based entropy decoding is investigated in detail in this paper. We propose a penalty-resource metric to distinguish the best solution among different entropy decoding methods under the resource (or penalty) constraints. We also propose a hierarchical partitioning method, HASGT, which provides more flexible partitioning methods with higher resource usage efficiency for dealing with AGH-tree. By the designed metric, any Lagrangian multiplier based techniques for searching optimal solutions can be applied directly. An optimal Lagrangian multiplier is associated with a total resource constraint; therefore, given a fixed multiplier, the constraint problem can be solved as an unconstraint problem. Once the optimal Lagrangian multiplier is given, the problem of searching an optimal HDM can be broken into finding the best HASGT structure for each individual HT due to the independency of HTs. Therefore, under a given Lagrangian multiplier, the problem of considering overall resource constraint can be released to individually find the best HASGT for each HT. This approach for searching optimal solution can also be applied to improve the performance of Hashemian methods.

Since HASGT provides more flexible ways to partition an AGH-tree, the number of possible HASGT structures is tediously large. Searching the optimal solution in brute-force way becomes infeasible, so we develop a Viterbi-like algorithm to efficiently find the optimal HASGT structure. After theoretical analysis, the complexity of the proposed Viterbi-like algorithm is substantially less than that of the brute-force method. Consequently, the performances of HASGT and previous existing methods are evaluated for applying to considerable quantities of HTs which are adopted in modern multimedia codecs such as AAC, AAC+(SBR), MPEG-2, MPEG-4, H.264/AVC and WMV-9. Under actual data set, the experimental results show that HASGT based approach performs better than previous approaches [3], [4], [7], [8], [12] in terms of memory size and memory access. For the method of [5], HASGT needs extra 55% memory size but gain 4.33 times in execution time.

REFERENCES

[1] X. Zhou, E. Q. Li, and Y. K. Chen, "Implementation of H. 264 decoder on general-purpose processors with media instructions," in *Proc. SPIE Conf. Image Video Commun. Process.*, 2003, vol. 5022.

[2] S. W. Wang, Y. T. Yang, C. Y. Li, Y. S. Tung, and J. L. Wu, "The optimization of H. 264/AVC baseline decoder on low-cost TriMedia DSP processor," *Proc. SPIE*, vol. 5558, p. 524, 2004.

[3] K. L. Chung and J. G. Wu, "Level-compressed Huffman decoding," *IEEE Trans. Commun.*, vol. 47, no. 10, pp. 1455–1457, Oct. 1999.

[4] J. S. Lee, J. H. Jeong, and T. G. Chang, "An efficient method of Huffman decoding for MPEG-2 AAC and its performance analysis," *IEEE Trans. Speech Audio Process.*, vol. 13, no. 6, pp. 1206–1209, Jun. 2005.

[5] H. C. Chen, Y. L. Wang, and Y. F. Lan, "A memory-efficient and fast Huffman decoding algorithm," *Inf. Process. Lett.*, vol. 69, no. 3, pp. 119–122, Mar. 1999.

[6] Y. K. Lin and K. L. Chung, "A space-efficient Huffman decoding algorithm and its parallelism," *Theoret. Comp. Sci.*, vol. 246, no. 1, pp. 227–238, 2000.

[7] R. Hashemian, "Memory efficient and high-speed search Huffman coding," *IEEE Trans. Commun.*, vol. 43, no. 10, pp. 2576–2581, Oct. 1995.

[8] M. Aggarwal and A. Narayan, "Efficient Huffman decoding," in *Proc. Int. Conf. Image Process.*, 2000, vol. 1, pp. 1936–1939.

[9] S. B. Choi and M. H. Lee, "High speed pattern matching for a fast Huffman decoder," *IEEE Trans. Consumer Electron.*, vol. 41, no. 1, pp. 97–103, Feb. 1995.

[10] J. H. Jiang, C. C. Chang, and T. S. Chen, "An efficient Huffman decoding method based on pattern partition and look-up table," in *Proc APCC/OECC'99*, 1999, vol. 2, pp. 904–907.

[11] R. Hashemian, "Condensed table of Huffman coding, a new approach to efficient decoding," *IEEE Trans. Commun.*, vol. 52, no. 1, pp. 6–8, Jan. 2004.

[12] P. C. Wang, Y. R. Yang, C. L. Lee, and H. Y. Chang, "A memory-efficient Huffman decoding algorithm," in *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, 2005, vol. 2, pp. 475–479.

[13] S. W. Wang, S. C. Chuang, C. C. Hsiao, Y. S. Tung, and J. L. Wu, "An efficient memory construction scheme for an arbitrary side growing Huffman table," in *Proc. IEEE Int. Conf. Multimedia Expo.*, 2006, pp. 141–144.

[14] T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[15] H. Everett, III, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources," *Oper. Res.*, vol. 11, pp. 399–417, 1963.

[16] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 36, no. 9, pp. 1445–1453, Sep. 1988.

[17] K. Ramchandran and M. Vetterli, "Best wavelet packet bases in a rate-distortion sense," *IEEE Trans. Image Process.*, vol. 2, no. 2, pp. 160–175, Apr. 1993.

[18] S. Srinivasan, P. Hsu, T. Holcomb, K. Mukerjee, S. L. Regunathan, B. Lin, J. Liang, M. C. Lee, and J. Ribas-Corbera, "Windows media video 9: Overview and applications," *Signal Process.: Image Commun.*, vol. 19, pp. 851–875, 2004.

[19] Communication and Multimedia Laboratory, National Taiwain Univ., Taipei, Taiwan, R.O.C. [Online]. Available: http://www.cmlab.csie.ntu.edu.tw/~song/HASGT/Appendix.pdf

[20] ISO. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/

[21] JTC1/SC29 WG11, Moving Pictures Expert Group. [Online]. Available: http://mpeg.nist.gov/JVT/docs/jvt-site/

[22] Communication and Multimedia Laboratory, National Taiwain Univ., Taipei, Taiwan, R.O.C. [Online]. Available: http://www.microsoft.com/windows/windowsmedia/musicandvideo/hdvideo/contentshowcase.aspx

[23] Communication and Multimedia Laboratory, National Taiwain Univ., Taipei, Taiwan, R.O.C. [Online]. Available: http://www.cmlab.csie.ntu.edu.tw/~song/HASGT/

Sung-Wen Wang (S'05–M'08) received the B.S. degree in the Department of Computer Science, Tamkang University, Taipei, Taiwan, in 2001 and the M.S. degree from the Department of Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, in 2003, where he is currently pursuing the Ph.D. degree in the same department.

His general research interests are in the field of digital video coding, codec-processor architecture co-design and multimedia systems optimization, especially in video coding technology optimization.

Ja-Ling Wu (SM'98–F'08) received the Ph.D. degrees in electrical engineering from Tatung Institute of Technology, Taipei, Taiwan, in 1986.

From 1986 to 1987, he was an Associate Professor of the Electrical Engineering Department, Tatung Institute of Technology. In 1987, he transferred to the Department of Computer Science and Information Engineering (CSIE), National Taiwan University (NTU), Taipei, Taiwan, where he is presently a Professor. From 1996 to 1998, he was assigned to be the first Head of the CSIE Department, National Chi Nan University, Puli, Taiwan. During his sabbatical leave (from 1998 to 1999), Prof. Wu was invited to be the Chief Technology Officer of the Cyberlink Corp. In this one year term, he was involved with the developments of some well-known audio-video software, such as the PowerDVD. Since Aug. 2004, Prof. Wu has been appointed to head the Graduate Institute of Networking and Multimedia, NTU. Prof. Wu has published more than 200 technique and conference papers. His research interests include digital signal processing, image and video compression, digital content analysis, multimedia systems, digital watermarking, and digital right management systems.

Prof. Wu was the recipient of the Outstanding Young Medal of the Republic of China in 1987 and the Outstanding Research Award three times of the National Science Council, Republic of China, in 1998, 2000 and 2004, respectively. In 2001, his paper "Hidden Digital Watermark in Images" (co-authored with Prof. Chiou-Ting Hsu), published in IEEE TRANSACTIONS ON IMAGE PROCESSING, was selected to be one of the winners of the "Honoring Excellence in Taiwanese Research Award", offered by ISI Thomson Scientific. Moreover, his paper "Tiling Slideshow" (co-authored with his students) won the Best Full Technical Paper Award in ACM Multimedia 2006. He was selected to be one of the lifetime Distinguished Professors of NTU, November 2006. He was elected to be IEEE Fellow for his contributions to image and video analysis, coding, digital watermarking, and rights management.

Shang-Chih Chuang received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taiwan, in 2005 and 2007, respectively.

His current research interests include video compression, image processing, stereo imaging, multimedia application, and multimedia security.

Chih-Chieh Hsiao received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 2005 and 2007, respectively.

His research interests include digital content analysis and multimedia indexing.

Yi-Shin Tung received the B.S. and Ph.D. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1996 and 2002.

He was at postdoctoral position in the same department from 2002 to 2005. He is currently working in Hon-Hai Precision Ind. Co., Ltd. (Foxconn Corporation), Taipei, Taiwan as a Project Manager and Senior Engineer. Since 2005, he has also been an Adjunct Assistant Professor at National Taiwan University. His current research interests include image and video coding, digital video processing, multimedia systems, and software hardware co-design.