Efficient Rate Control for JPEG-2000

Wei Yu, *Member, IEEE*, Fangting Sun, *Student Member, IEEE*, and Jason E. Fritts, *Member, IEEE*

***Abstract*—In this paper, we present two new methods for efficient rate control and entropy coding in lossy image compression using JPEG-2000. These two methods enable significant improvements in computation complexity and power consumption over the traditional JPEG-2000 algorithms. First, we propose a greedy heap-based rate-control algorithm (GHRaC), which achieves efficient postcompression rate control by implementing a greedy marginal analysis method using the heap sort algorithm. Second, we propose an integrated rate-control and entropy-coding (IREC) algorithm that reduces the computation complexity of entropy coding by selectively entropy coding only the image data that is likely to be included in the final bitstream, as opposed to entropy coding all image data. Together, these two methods enable significant savings in computation time and power consumption. For example, the GHRaC method demonstrates 16× speedup for rate control when encoding the *Lena* color image using a target compression ratio of 128:1, one quality layer, and code blocks of 32×32 pixels. The IREC method expands upon GHRaC to perform entropy coding in conjunction with rate control. Using an enhanced version of IREC, these two methods jointly achieve a speedup in execution time of $14 \times$ over traditional rate control and entropy coding, which first entropy codes all image coefficients and then separately performs postcompression rate control using the generalized Lagrange multiplier method to select which data are included in the final bitstream. Both theoretical analysis and empirical results are presented in validating the advantages of the proposed methods.**

***Index Terms*—Integrated rate control and entropy coding (IREC), JPEG-2000, rate-distortion (R-D) optimization.**

I. Motivation and Introduction

JPEG-2000 is the latest international standard for still image compression [1]. It has been shown to be superior in many aspects to the existing standards [2]: JPEG, MPEG-4 Visual Texture Coding, JPEG-LS, and Portable Network Graphics (PNG). Besides providing state-of-the-art image compression, it offers a number of features to address the requirements of emerging applications, such as progressive compression and transmission, region of interest coding (ROC), and robustness to bit errors, among others [3]. These features are of importance to many high-end and emerging applications, including the Internet, remote sensing, and mobile computing.

Although JPEG-2000 has many advantages over other image coding standards, the high computation complexity currently prevents its practical use in many applications, especially applications with real-time or low-power constraints. In this paper, we address the computation complexity issue of some core components of JPEG-2000. In particular, we present two new methods for faster rate control and entropy coding. First, we propose the greedy heap-based rate-control (GHRaC) algorithm,[1] which is an efficient postcompression rate-control algorithm that implements a greedy marginal analysis method [4] using the heap sort algorithm [5]. Compared with the widely used generalized Lagrange multiplier method [6], the proposed scheme offers an order of magnitude speedup for typical lossy compression. For example, when compressing the *Lena* color image using a target compression ratio of 64:1, one quality layer, and code blocks of 32×32 pixels, the GHRaC method achieved a speedup of nearly $14 \times$ over the generalized Lagrange multiplier method for postcompression rate control. Overall, when encoding with compression ratios of 64:1 and one quality layer, speedups with rate control of $10 \times$ to $18 \times$ were obtained for code blocks of 32×32 pixels, and speedups of $4 \times$ to $9 \times$ were achieved with code blocks of 64×64 pixels. Even greater speedups can be achieved with higher compression ratios or multiple progressive quality layers.

However, improving the speed of only postcompression rate control does not greatly reduce the overall execution time for lossy compression, since the rate-control procedure typically accounts for only a moderate fraction of the overall execution time. To achieve more significant execution time speedups, we also target the entropy-coding procedure, which in JPEG-2000 has been found to dominate execution time and power consumption [7]. With the exception of lossless or very high-quality compression (i.e., very high bit rates), a large fraction of the data generated from entropy coding is never used in the final bitstream, resulting in an enormous waste of computing resources. To address this inefficiency, we propose the integrated rate-control and entropy-coding (IREC) scheme to jointly reduce the computation complexity of rate-control and entropy-coding procedures. The IREC scheme tries to perform entropy coding for only those parts of the image data that are likely to be included in the final bitstream, based on the target compression ratio and characteristics of the image. We propose two versions of IREC. The initial version, called the *IREC local-K* scheme, reduces the cost of entropy coding by only entropy coding the next three unselected coding passes in each code block. The second version, called the *enhanced IREC local-K*, incorporates coefficient weighting models to further reduce the number of unselected coding passes that need to be entropy coded over all code blocks.

Compared with schemes that perform rate control and entropy coding separately, the IREC method can dramatically reduce the overall computation complexity with a negligible loss in performance. Empirical results demonstrate the efficacy of the IREC method. For example, when encoding the Lena color

Manuscript received November 14, 2003; revised September 22, 2005. This paper was recommended by Associate Editor H. Watanabe.

W. Yu and F. Sun are with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA (e-mail: weiyu@glue.umd.edu; ftsun@glue.umd.edu).

J. E. Fritts is with the Department of Mathematics and Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: jfritts@slu.edu).

Digital Object Identifier 10.1109/TCSVT.2006.873161

[1]GHRaC is pronounced like "grace," and IREC is pronounced like "Iraq."

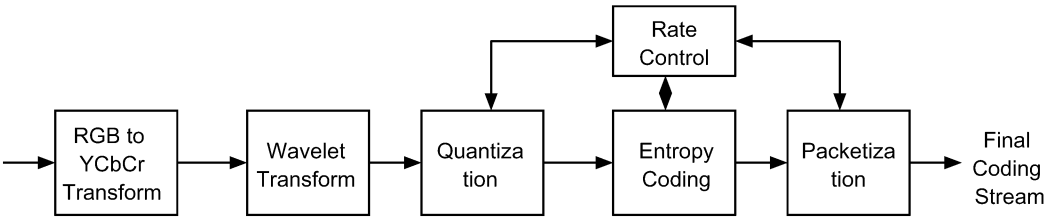


Fig. 1. Structure of the JPEG-2000 encoder.

image with a target compression ratio of 128:1, one quality layer, and code blocks of 32×32 pixels, the enhanced IREC local-3 method achieved over $14\times$ speedup over separate rate control and entropy coding. Overall, speedups in entropy coding and rate control of $2\times$ to $3.5\times$ were achieved with the base IREC local-K scheme (with $K = 3$), and speedups of $5\times$ to $9\times$ were achieved with the E-IREC local-K scheme (with $K = 3$) when encoding with compression ratios of 64:1, one quality layer, and code blocks of 32×32 pixels.

The remainder of this paper is organized as follows. Section II gives a general introduction to JPEG-2000 and the current postcompression rate-control methods. Section III presents the GHRaC optimization, which applies the heap sort to a greedy rate-control algorithm to achieve significant speedups in rate control. Section IV presents the IREC scheme to jointly perform rate control and entropy coding. Section V presents the experimental results, demonstrating the efficacy of the two new optimizations. Finally, Sections VI concludes this paper.

II. Rate Control in JPEG-2000

A. Introduction to JPEG-2000

JPEG-2000 is a wavelet-based image compression standard that employs the Embedded Block Coding with Optimized Truncation (EBCOT) algorithm for entropy coding and rate control [8]–[10]. The general encoding structure is illustrated in Fig. 1. Following RGB to YCrCb color space conversion, the wavelet transform is applied to transform each color component into a hierarchy of subbands using the dyadic decomposition attributed to Mallat [11]. As illustrated in Fig. 2, in the transform domain, each component is represented as a collection of resolution levels, and each level contains three subbands, with the exception of the coarsest resolution level $(LL0)$, which has only one subband. Each subband is then quantized and coded separately. To provide flexibility and error resilience, each subband is often further partitioned into relatively small code blocks (usually 32×32 or 64×64 samples per code block), and entropy coding is independently performed on each code block.

The fundamental method of entropy coding in JPEG-2000 is based on the MQ coder, which is essentially a bit-plane coder employing classical context-adaptive arithmetic coding to efficiently represent a collection of binary symbols [12]. Starting from the most significant bitplane in a code block, three passes are made over each bitplane in each code block. These passes are referred to as coding passes.[2] Each coding pass provides a variable quality contribution to the reconstructed image. After all of the coding passes have been generated, a postcompression rate-control procedure is applied to determine which coding passes should be included in the final bitstream according to some user-specified optimization criteria.

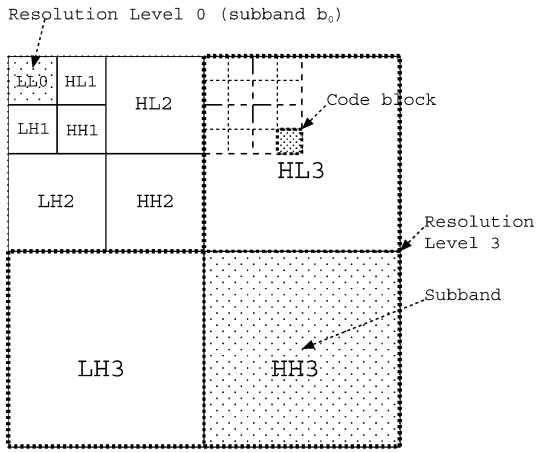


Fig. 2. Wavelet decomposition of an image.

Bit rate and quality are the two primary optimization criteria for rate control. When the user specifies a desired bit rate (i.e., degree of compression), rate-distortion (R-D) optimization is performed to maximize the quality (minimize the distortion) for a target bit rate. Conversely, when the user specifies a desired quality, R-D optimization is performed to minimize the bit rate for the target quality. The JPEG-2000 postcompression rate-control procedure provides two optimization modes for these two criteria. In this paper, we focus on the first mode, where the user specifies the target bit rate, and rate distortion minimizes distortion for the specified bit rate. However, rate control for the second criteria, quality, is a complementary problem, and we shall briefly discuss in Sections III and IV how the GHRaC and IREC methods can be readily adapted to this mode, respectively.

B. R-D Optimization in JPEG-2000

Now let us introduce the R-D optimization problem in JPEG-2000. Let B_i denote the ith code block and p_i^j denote the jth coding pass of code block B_i. Each code block B_i contains at most N_i coding passes,[3] labeled p_i^0 to $p_i^{N_i-1}$. In lossy compression, the full set of coding passes in each code block is truncated to reduce the bit rate for that code block, so let n_i denote the number of coding passes selected from B_i to be included in the

[2]Hereafter, we will use pass to denote the coding procedure and coding pass to denote the embedded bitstream generated by each pass.

[3]In the EBCOT coding scheme, the number of coding passes N_i is $3b_i - 2$, where b_i is the number of bits/bitplanes for elements of code block B_i.

final bitstream. In this paper, n_i is referred to as a *truncation point* for code block B_i, which means that coding passes $p_i^{n_i}$ to $p_i^{N_i-1}$ following the truncation point n_i in code block B_i are not included in the final bitstream. Assuming an additive distortion metric and defining $D_i^{n_i}$ as the distortion associated with truncation point n_i of code block B_i, then the total distortion D from including only the first n_i coding passes of each code block B_i in the final bitstream (versus selecting all N_i coding passes) can be calculated as

$$D = \sum_i D_i^{n_i}. \tag{1}$$

Additionally, for each code block B_i, let K_i denote the set of indices for the sequence of subband samples in that code block. Let $\{\hat{s}_i^{n_i}[k]\}$ denote the sequence of subband samples associated with truncation point n_i (i.e., including only the first n_i coding passes of B_i), and let $\{s_i[k]\}$ denote the full set of subband samples, in which no sample is truncated. Then, the distortion $D_i^{n_i}$ for code block B_i may be modeled by the following equation, as indicated by Taubman [10]:

$$D_i^{n_i} = w_i^2 \sum_{k \in K_i} \left(\hat{s}_i^{n_i}[k] - s_i[k]\right)^2. \tag{2}$$

This measures distortion contributed by code block B_i due to the data loss from truncating the bitstream at point n_i. w_i is the coefficient weight of code block B_i, which is determined by the location and characteristics of B_i and the human perceptual models being used, as indicated in [12].

To determine the bit rate, let $R_i^{n_i}$ be the total number of bits in the first n_i coding passes of code block B_i. The total bit rate corresponding to the set of truncation points, $\{n_i\}$, across all code blocks in the image, then becomes

$$R = \sum_i R_i^{n_i}. \tag{3}$$

Now the rate-distortion optimization problem for the first rate-distortion optimization mode, in which the user specifies the target bit rate, can be formulated as

$$\arg\min_{\{n_i\}} D \text{ s.t. } R \le R^{\max}. \tag{4}$$

Effectively, this optimization problem is the problem of finding the optimal set of truncation points $\{n_i\}$ that minimizes the overall distortion under the specified rate constraint. Likewise, the second R-D optimization mode, minimizing the overall bit rate for a target quality, can be formulated as

$$\arg\min_{\{n_i\}} R \text{ s.t. } D \le D^{\max}. \tag{5}$$

As indicated earlier, we mainly focus on the first R-D optimization mode, but later we will briefly discuss how the second problem can be easily solved using the same method.

A variety of techniques have been proposed for solving the discrete R-D optimization problem. Some of these methods provide the optimal solution, while other methods only provide an approximation to the optimal solution. There are currently three popular approaches to this problem: dynamic programming [13], which provides an optimal solution, the generalized Lagrange multiplier (LM) method [6], and greedy marginal analysis [4], [14], of which the latter two only approximate the solution.

1) Dynamic Programming: Dynamic programming is most commonly employed when the optimal solution is desired for R-D optimization. When using dynamic programming, the algorithm starts by creating a tree that represents all possible solutions, such as that illustrated in [13]. Each stage of the tree corresponds to a code block B_i, and each node of the tree at a given stage represents a possible cumulative rate usage. Each branch has as a distortion cost corresponding to that particular truncation point, and therefore, as we traverse the tree from the root to the leaves, we can compute the accumulated distortion for each of the solutions. During this procedure, Bellman's principle can be applied to optimally prune the tree [15]. While the tree pruning can be used efficiently in some situations, like the widely used Viterbi algorithm, in many situations the number of nodes in the tree increases exponentially. This is typically the case when dynamic programming is used to solve the R-D optimization problem in JPEG-2000. Thus, while it guarantees optimality, in reality, dynamic programming is generally impractical for R-D optimization due to its prohibitively high computational complexity.

2) Generalized LM: The LM method has also been widely used to solve the constrained optimization problem. If the distortion function is differentiable, the set of truncation points $\{n_i\}$ can be found by minimizing the cost function [10]

$$(D(\lambda) + \lambda R(\lambda)) = \sum_i (D_i^{n_i^\lambda} + \lambda R_i^{n_i^\lambda}) \tag{6}$$

where λ is the LM. However, since the truncation point set is discrete, the optimal solution is not always easy to find.

To help resolve the problem of computing the LM for a discrete set of points, Everett [6] proposed the *generalized* LM method. The object of this method is to find the optimal LM λ_o that generates the least distortion under the given rate constraint. For each λ, there is a rate constraint $R(\lambda)$ based upon the target bit rate. Given this rate constraint, there is a corresponding set of truncation points $\{n_i^\lambda\}$ that minimizes this cost function. By sweeping λ over its range, the generalized LM method can find a λ_o that satisfies $R(\lambda_o) \le R^{\max}$ such that no other λ exists for which $R(\lambda_o) < R(\lambda) \le R^{\max}$. If $R(\lambda_o) = R^{\max}$, then the corresponding truncation point set $\{n_i^{\lambda_o}\}$ is optimal. If $R(\lambda_o) \ne R^{\max}$, the solution is not guaranteed to be optimal, but it is still a good approximation.

A second problem of the LM method with respect to JPEG-2000 is that only those coding passes whose R-D ratios fall on the convex hull of the R-D curve can be selected as potential truncation points [6]. Since truncation points in a code block do not always reside on the convex hull of the R-D curve, this means that the generalized LM method cannot always give an

optimal solution to the R-D problem in JPEG-2000. Fortunately, if the set of points residing on the convex hull is sufficiently dense, as is typically the case in JPEG-2000 [12], the generalized LM method can provide a good approximation to the optimal solution with much lower computational complexity than dynamic programming methods. In JPEG-2000, the fractional bitplane coding style gives a finely embedded bitstream such that the truncation points residing on the convex hull of the R-D curve are usually sufficiently dense to give a good approximation to the optimal solution. With this characteristic, the R-D optimization problem for JPEG-2000 is commonly solved through a two-step process. First, convex hull analysis is used to determine which coding passes reside on the convex hull of the rate-distortion curve. Then the generalized LM method is applied to approximate the optimal solution [10], [12].

We shall now explicitly define the convex hull with respect to R-D optimization in JPEG-2000. First, we start with the R-D curve for a code block, which is defined by the set of R-D ratios (i.e., R-D slopes) between each pair of truncation points in the code block. Let us define the R-D ratio (slope) $g_i(j,k)$ between a pair of truncation points j and k in the code block B_i as

$$g_i(j,k) = \frac{D_i^j - D_i^k}{R_i^k - R_i^j} (j < k). \tag{7}$$

Given this R-D curve, we shall define the *set of feasible truncation points* for a code block to be the set of all truncation points residing on the convex hull of the R-D curve for that code block. Then, the necessary and sufficient condition that a truncation point n_i of code block B_i is "feasible" becomes [12]

$$\min_{k<n_i} g_i(k,n_i) > 0 \tag{8}$$

$$\min_{k<n_i} g_i(k,n_i) > \max_{j>n_i} g_i(n_i,j). \tag{9}$$

Inequality (8) indicates that it is not possible to select an earlier truncation point in code block B_i with a lower R-D ratio (slope), and inequality (9) says that, if n_i is a feasible truncation point, then all of the R-D ratios (slopes) between n_i and the truncation points before it must be larger than all of the R-D ratios (slopes) between n_i and the truncation points after it.

An efficient convex hull analysis algorithm to find the set of feasible truncation points for each layer has been presented in [12], and the computation complexity has an upper bound of two times the number of coding passes (i.e., truncation points). In this paper, we define a *coding segment* as the set of coding passes between two consecutive feasible truncation points in a code block.[4]

After determining the set of feasible truncation points for each code block, the generalized LM method can be applied to approximate the optimal solution. As mentioned earlier and discussed in detail in [6], the set of optimal truncation points among all feasible truncation points can be found by sweeping the LM λ across its full range. However, since λ is real-valued and its range can be partitioned infinitely, in theory, the number of iterations needed to find the optimal solution can become quite large. Consequently, in practice, most implementations of the LM method stop the search once either: 1) the result lies within a specified bound or 2) the maximum number of iterations has been reached.

To reduce the time complexity of the generalized LM method, a bi-section search can be applied to find a suitable λ given rate constraint $R^{\max}$. To determine the time complexity of the generalized LM method using the bi-section search, let N be the number of code blocks, let P_s be the average number of feasible truncation points (segments) per code block, and let I be the maximum number of search iterations before the bi-section search stops. Then, the time complexity of the generalized LM method using the bi-section search is $O(NI \log P_s)$, which includes the cost of finding the set of feasible truncation points for each code block in the bi-section search.

In many scenarios, a quality scalable bitstream is desirable. Such is the case when a compressed image is sent to many users, with varying bandwidth constraints. Using quality scalable compression, only one bitstream needs to be generated, which can then be sent to all users with progressive quality. Quality-scalable compression denotes that the final bitstream contains multiple quality layers, with each representing an efficient compression of the original image at progressively lower distortions. Another advantage of a quality-scalable bitstream is that it facilitates unequal error protection, which is an important technique in achieving good image transmission over error-prone channels [16], [17]. When generating quality-scalable bitstreams, the computation complexity of the generalized LM method increases linearly with the number of quality layers. Assuming that the generalized LM method is being used to generate a quality-scalable bitstream with L quality layers, then the LM search must be applied L times, once for each quality layer. As a result, the time complexity of the generalized LM method for generating multiple quality layers becomes $O(LNI \log P_s)$. Since computation complexity is now linearly dependent upon both the number of search iterations and the number of quality layers, it therefore becomes desirable to either limit the number of quality layers or further limit the number of search iterations used in each quality layer.

3) Greedy Marginal Analysis Algorithm: From the previous discussion, it is apparent that significant computation complexity is required for the generalized LM method in performing the many search iterations needed to find the Lagrange multiplier λ_o that minimizes the distortion for the given rate constraint. Furthermore, this drawback becomes increasingly significant for many quality layers since the computation complexity increases linearly with the number of layers. An alternate, more efficient approach is to use a greedy algorithm for coding pass selection.

A good greedy algorithm for R-D optimization is Fox's marginal analysis Algorithm 1. The basic idea is to allocate the available bits step by step in such a way that in each step the incremental gain per bit is maximized. Based on this application of Fox's greedy algorithm, we propose the following algorithm for R-D optimization in JPEG-2000, as presented in Algorithm 1. When used to select the set of coding passes for a given target

[4]In this paper, the terms *coding segment* and *feasible truncation point* will be used interchangeably, since a feasible truncation point effectively denotes the coding segment immediately preceding it.

bit rate, this greedy algorithm limits the combination of coding passes that may be included in the final bitstream. When the compression ratio is relatively high, only a small number of all of the coding passes can be included in the final bitstream. Based on this observation and the fact that, after convex hull analysis, the R-D ratios of the coding segments in each code block are strictly decreasing, the proposed greedy method can be applied to find the coding segments (and correspondingly, the coding passes) to be included.

Algorithm 1: Greedy Algorithm for Rate Control

1: $R := 0$;

2: **while** $(R < R^{\max})$ **do**

3: Find an unselected feasible truncation point n_i^j with the maximum rate-distortion ratio $g_i(n_i^{j-1}, n_i^j)$ among all code blocks and select it;

4: $R := R + R_i^{n_i^j} - R_i^{n_i^{j-1}}$;

5: **end while**

From Algorithm 1, we can see that the computation complexity of this algorithm depends on how efficiently each iteration is able to find the next feasible truncation point to be included in the final bitstream. This requires searching through all N code blocks for the remaining coding segment with the greatest R-D ratio (i.e., the remaining coding segment that will generate the most distortion per bit, if not included in the final bitstream).

A straightforward implementation of this greedy algorithm, however, does not yield impressive results. Assuming that the total number of feasible truncation points (segments) to be selected is M_s, where $0 < M_s < NP_s$, then the overall time complexity is $O(\mathrm{NM}_s)$. For example, let us assume a gray-scale image with a resolution of 512×512, 4 levels of wavelet decomposition, a code block size of 32×32 (which therefore contains $N = 256$ code blocks), and an average of $P_s = 15$ coding segments per code block. For a target compression ratio of $32:1$ and only $L = 1$ quality layer, the number of coding passes to be included is approximately $NP_s/M_s = 10$. Assuming that the number of search iterations for the generalized LM method is $I = 32$, then the complexity of the generalized LM algorithm is about $LNI \log_2 P_s = 32005$, while the complexity of the greedy algorithm is about $M_s N = 98304$. Therefore, if not implemented in an efficient way, the greedy algorithm may be worse than the generalized LM method in time complexity, especially when the compression ratio is relatively low. To address this inefficiency, we present an optimized version of this greedy algorithm in Section III.

III. GHRAC

Here, we present an efficient implementation of the greedy marginal analysis algorithm using the *heap* data structure and its associated functions, which enables us to quickly find the next feasible truncation point to be included in the final bitstream in each iteration of Algorithm 1. Next, we first provide a brief introduction to the heap data structure. More detailed descriptions of the heap and associated theory are presented by Cormen *et al.* [5].

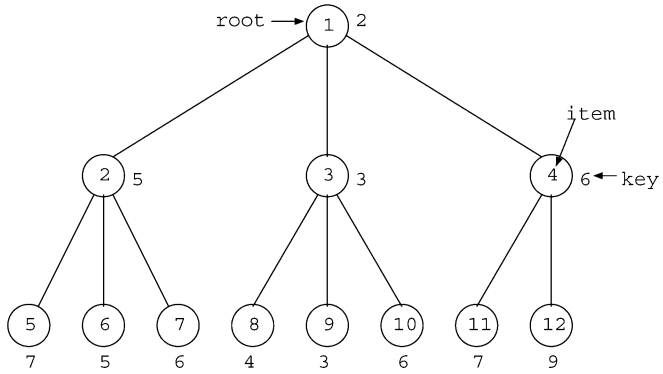


Fig. 3. d-heap data structure.

A *heap* is a multielement data structure which may be dually referenced as both a tree and an array. The flexibility of the heap data structure makes it ideal for implementing sorting or priority queues. The most basic heap structure is the binary heap, which is similar to a binary tree in that each nonleaf node has up to two child nodes. The more general heap implementation is the d-ary heap (where $d > 1$). The organization of the d-heap is such that nodes are numbered in breadth-first order, and each tree node in the heap (except for leaf nodes and the bottom-rightmost parent node) has d child nodes. As a consequence of this organization, the heap can be indexed as either an array or a tree, since the parent and child nodes of any node in the tree/array can be easily determined. Given the index i for any tree node (or array element), the index of its parent node and its jth child node can be computed as follows:

$$\mathrm{PARENT}(i) = \left\lfloor \frac{i + (d-2)}{d} \right\rfloor \tag{10}$$

$$\mathrm{CHILD}(i, j) = i * d - (d-2) + j \quad 0 \le j < d. \tag{11}$$

It is readily apparent that the depth of a d-heap with n nodes is $\le \lceil \log_d n \rceil$. An example d-ary heap with $d = 3$ is illustrated in Fig. 3.

The other key feature of the heap is that an ordered heap satisfies the heap property. The heap property states that the value of the key of a node is no greater than the value of its parent's key, which is mathematically stated as

$$A[\mathrm{PARENT}(i)] \ge A[i]. \tag{12}$$

The two major functions for use with the heap data structure are *makeheap* and *siftdown*. The makeheap function takes as input a completely unordered array and sorts all of the items so that the resulting order obeys the heap property. The siftdown function is similar, but assumes that only the root node of the heap is unordered, i.e., all child subheaps are ordered to satisfy the heap property. Consequently, the siftdown function checks whether the key of the root node is greater than or equal to all of its child nodes' keys. If not, it exchanges nodes with

the child node having the largest key and then recursively repeats the process with that child subheap, until the entire heap satisfies the heap property.

For our proposed rate-control algorithm GHRaC, we use the heap data structure to implement a priority queue for selecting the set of coding segments to be included in the final bitstream. In this implementation of a heap-based priority queue, each node in the heap corresponds to one of the available code blocks, and each node has as its key the R-D ratio of the first unselected coding segment (i.e., the first unselected feasible truncation point) in that code block. The heap organization of the nodes (code blocks) allows the indices for parent and child nodes to be calculated directly as discussed above, eliminating the need for pointers. The algorithmic description of the GHRaC rate-control procedure is presented in Algorithm 2.

Algorithm 2: GHRaC Algorithm Under Rate Constraint

1: **procedure** GHRaC-rate (**set** $\{B_i\}$, **int** L, **set** $\{R_l\}$)

2: **int** R, l; **heap** h;

3: h := makeheap($\{B_i\}$);

4: $l := 1; R := 0$;

5: **while** $(l \leq L)$ **do**

6: **while** $(R < R_l)$ **do**

7: $R := R + \text{rate}(\text{root}(h))$;

8: update(root(h), l);

9: siftdown(root(h), key(root(h)), h);

10: **end while**

11: $l := l + 1$;

12: **end while**

The GHRaC algorithm first initializes the d-heap tree by ordering the nodes in the heap using the makeheap($\{B_i\}$) function, based on the R-D ratios of the first coding segment of each code block B_i. After initialization, the heap is ordered such that the root node has the largest key, and the key of each node is no larger than the key of its parent. Meanwhile, all of the feasible truncation points are marked as unselected. The rate(B_i) function returns the rate for the first unselected coding segment of code block B_i. The update(B_i, l) function marks the first unselected coding segment of B_i to indicate that it will be included in the lth layer and then updates the key of code block B_i to be the R-D ratio of the next unselected coding segment in that block. The siftdown function updates the d-heap tree to conserve the heap property.

The time complexity of the makeheap operation is $O(2N)$, and the time complexity of siftdown is $O(d\log_d N)$, where N is the total number of code blocks [5]. The time complexity of the update is $O(1)$. From the algorithm, we see that makeheap and siftdown are performed once for each coding segment, so the overall time complexity becomes $O(2N + M_s d\log_d N)$, where M_s denotes the total number coding segments to be included in the final bitstream. For a fixed target compression ratio, the time complexity is independent of the number of quality layers to be generated, unlike the generalized LM method, whose time complexity increases linearly with the number of layers. The only extra storage/memory needed by the GHRaC algorithm is the heap data structure, where the number of nodes in the heap data structure is equal to the total number of code blocks. Since the total number of code blocks is usually not large (e.g., for a 1024×1024 grayscale image with a block size of 64×64, the total number of blocks is 256), and each entry only requires a few bytes so the extra storage/memory consumption is small.

Comparing this implementation with the original implementation of the greedy algorithm (which uses a brute-force sequential search in each iteration) whose complexity is $O(\text{NM}_s)$, it is evident that the GHRaC implementation may be up to $N/(d\log_d N)$ times faster. For example, when encoding an image using a four-level wavelet decomposition with one component of size 512×512, a code block size of 32×32 (i.e., there are $N = 256$ code blocks), and using a binary heap (i.e., $d = 2$), then a speedup of up to 16 can be achieved.

Now, let us return to the example from Section II-B3 to compare the computation complexity of the GHRaC algorithm, which has complexity $O(N + M_s d\log_d N)$, with the generalized LM method, which has complexity $O(LNI\log_2 P_s)$. Assuming $N = 256$ code blocks, an average of $P_s = 15$ coding segments per code block, $I = 32$ search iterations for the generalized LM method, a compression ratio of $M_s = NP_s/10$, $L = 1$ quality layers, and a binary heap $(d = 2)$ data structure, then the generalized LM method still requires about 32005 *comparison* operations in order to find all of the coding passes to be included in the final bitstream. Conversely, the number of *comparison* operations needed by the GHRaC algorithm is now only about 6656, giving approximately a $5\times$ speedup. Since the computation complexity of the GHRaC algorithm does not depend on L, when increasing L from $L = 1$ to $L = 10$, the number of comparisons for the GHRaC scheme remains unchanged because M_s does not change. Conversely, the number of operations for the generalized LM method increases linearly with L to become 32005, so the GHRaC algorithm provides significant gain. As will be shown in Section V-A, experimental performance results are given for the two methods across a variety of images and parameterizations, confirming this theoretical analysis.

As mentioned earlier, the GHRaC scheme can also be easily modified to solve the second rate-distortion optimization problem, which is the problem of minimizing the bit rate under a specified quality (or distortion) constraint. Assuming the specified quality constrains the distortion to $\{D_l\}$ for each layer l, where $D_1 > D_2 > \cdots > D_L$, then all that we need to change in Algorithm 2 is to replace D with R, replace $D := D_{\text{total}}$ with $R := 0$, replace $D > D_l$ with $R < R_l$, and replace $D := D - \text{distortion}(\text{root}(h))$ with $R := R + \text{rate}(\text{root}(h))$. Here, D_{total} denotes the total distortion when no coding passes are selected, and $\text{distortion}(b)$ denotes the distortion reduction by moving the truncation point of code block b to the next feasible one. The modified scheme, illustrated in Algorithm 3, has the same computation complexity as Algorithm 2.

Algorithm 3: GHRaC Algorithm Under Quality Constraint

1: **procedure** GHRaC-quality (**set** $\{B_i\}$, **int** L, **set** $\{D_l\}$)
2: **int** D, l; **heap** h;
3: h := makeheap($\{B_i\}$);
4: $l := 1; D := D_{\text{total}}$;
5: **while** $(l \leq L)$ **do**
6: **while** $(D > D_l)$ **do**
7: $D := D -$ distortion(root(h));
8: update(root(h), l);
9: siftdown(root(h), key(root(h)), h);
10: **end while**
11: $l := l + 1$;
12: **end while**

IV. Integrated Rate Control and Entropy Coding

One key characteristic of most current implementations of JPEG-2000 is that rate allocation, which selects the coding segments to be included in the final bitstream, is not performed until all of the coding passes have been generated. While this implementation is effective for those situations that demand either very high image quality or many quality layers (some of which are very high quality), this is generally not the case for most users when performing image compression.

Conversely, it is much more common that image compression is performed to generate a bitstream targeting a specified bit rate or quality level. Since entropy coding is the most computationally intensive and power-intensive routine in JPEG-2000 encoding, and execution time and power consumption increase roughly linearly with the number of coding passes to be generated, it is very inefficient to first generate all of the coding passes and then select which coding passes should be included in the final bitstream, when the majority of the coding passes will not be used. This is especially true when the number of coding passes to be included in the final bitstream is only a small fraction of the total number of coding passes. All of the unselected coding passes are thrown away, so a significant amount of work is wasted in performing entropy coding on these passes.

A. Overview of Selective Entropy Coding

To avoid wasted effort, selective entropy coding should be applied, that is, if a coding pass is predicted to have no chance of being included in the final bitstream, then entropy coding should not be performed on it. To perform partial entropy coding, two types of methods can be used. The first one is referred to as model-based methods, which determine which coding passes are to be generated based on the target bit rate and some predetermined coefficient weighting models, such as the weighting factors for coefficients in different subbands and/or the weighting factors for bits in different bitplanes.

Masuzaki *et al.* [18] proposed one example of a model-based method, which provided an adaptive rate-control scheme for JPEG-2000 such that subbands are ordered in the final bitstream based on their significance. By predicting the adequate number of coding passes and updates adaptively, their scheme is able to reduce both computational cost and working memory size for bitstream buffering. Another example of a model-based method is to specify the quantization level for each subband such that, after quantization, bitplanes with all zero coefficients do not need to be coded [1], [10], [12]. Although such approaches can give good results in some situations, they have the drawback that no general model exists to precisely model all image types. For example, after being wavelet-transformed, the *baboon* image has substantial information in the high-resolution subbands, whereas *Lena* contains little information in these subbands. As a result, when using model-based methods to predetermine the set of coding passes on which to perform entropy coding, the quantization step sizes should be conservative enough to work for all images.

Alternatively, the second, and more versatile type of selective entropy coding, is content-adaptive entropy coding. Given a target bit rate, the generation of the coding passes depends not only on the weighting factors, but also on the characteristics of the image content. Compared with the model-based methods, content-adaptive methods can more precisely model each individual image. The drawback of content-adaptive methods is that, while they provide good approximations to the optimal solution, they typically entail much greater computational complexity for calculating the image characteristics and deciding which passes should be entropy coded.

B. Base IREC Algorithm

In this paper, we propose the IREC algorithm for selective entropy coding, which combines the accuracy of content-adaptive entropy coding with the efficiency of the GHRaC rate-control algorithm (which was presented in Section III). In this joint method, only those coding passes that have a good chance of being included in the final bitstream are entropy coded. Therefore, all other coding passes are not entropy coded, resulting in dramatic reductions in execution time and power consumption with negligible performance loss in most situations. The IREC scheme can be further improved by incorporating the model-based approaches, as will be discussed later in this section.

Operating in conjunction with the GHRaC rate-control method, the basis of the IREC selective entropy-coding algorithm is that only one coding segment needs to be available for each code block at any point in time. Since the GHRaC method only uses as the key for each node the R-D ratio of the next unselected feasible truncation point (coding segment), and the value of that key is in no way dependent upon any other coding segments in that code block, then it is sufficient that only the next coding segment be available for the GHRaC method in order for it to generate its result. Consequently, it is only necessary that entropy coding be performed on the next unselected coding segment in each code block.

However, as described earlier in Section II-B, the precise determination of coding segments is based on convex-hull analysis of the R-D ratios of coding passes in a coding block. The coding

TABLE I
STATISTICS OF THE NUMBER OF CODING PASSES BETWEEN TWO CONSECUTIVE FEASIBLE TRUNCATION POINTS

Number	1	2	3	4	5	6	≥ 7
Barbara	495	232	22	5	2	1	0
Fruits	1231	437	53	37	3	2	0
Lena	1139	491	46	42	3	4	0
Baboon	1332	699	40	41	1	0	0
Total	4197	1859	161	125	9	7	0
percent	66.0	29.2	2.53	1.97	0.14	0.11	0

passes that do not reside on the convex hull of the R-D curve are incorporated into the next coding segment. Consequently, for a coding segment containing m coding passes, $m+1$ coding passes must be generated via entropy coding. Since the number m of coding passes in the next segment is not known before $m+1$ passes are generated, the coding passes in a code block must be adaptively entropy coded until the requisite $m+1$ passes have been generated. An alternate, more deterministic method is to conservatively estimate the number of coding passes required in the next coding segment. This is the approach we take for the IREC method.

According to the statistical information presented in Table I, we can see that coding segments require only 1.4 coding passes on average. Furthermore, the vast majority of coding segments (over 95%) contain no more than two coding passes, and very few (less than 0.25%) contain more than four coding passes. In fact, this small number of coding passes per coding segment is characteristic of most images and is the foundation of the convex-hull approach to rate control commonly used by the generalized LM method, as well as our GHRaC method, as discussed in Section II-B. Likewise, this characteristic shows up in model-based entropy-coding methods that assign different weights to different bitplanes; for example, when using mean square error (MSE) as the quality criterion, the ratio between the bits in two consecutive bitplanes is about 4. We can likewise use this characteristic in the design of our IREC method.

Based on the characteristic average number of coding passes per coding segment, the IREC method estimates the R-D ratios for the next unselected coding segmented using only the next K coding passes. In effect, the IREC computes a R-D ratio for the next unselected feasible truncation point, but it does so only using the next K unselected coding passes. The remaining coding passes in the code block are not generated, and so cannot be used to measure the ratio. Consequently, when K coding passes are generated, the R-D ratio can be computed exactly if the coding segment contains K or fewer coding passes,[5] but, if it contains $K+1$ coding passes, then the calculated ratio will only be an estimate of the exact ratio. Fortunately, as shown below, this estimate is typically quite accurate, and inaccuracies have little impact on compression performance.

In other words, the IREC scheme only entropy codes the next K unselected coding passes in each code block. Then rate control performs convex-hull analysis and coding segment selection on only those K coding passes, in effect only using *local* information to find the set of *local* feasible truncation points. Based on the fact that each coding segment includes only a small number of coding passes, we expect that, if K is sufficiently large to cover most cases, in most situations, the set of *local* feasible truncation points will be nearly equivalent to the set of *global* feasible truncation points. By using only the set of *local* feasible truncation points, the number of coding passes that need to be generated can be reduced dramatically, and the IREC scheme provides an effective mechanism for accomplishing this.

The algorithmic description of the IREC scheme is presented in Algorithm 4. The IREC method operates in conjunction with the GHRaC method, so the makeheap, siftdown, and update functions are effectively the same as those in Algorithm 2. The entropyencoder(b, K) function generates the next K coding passes for code block b. The convexhull(b, K) function performs the convex hull analysis on these K unselected coding passes to find the set of local feasible truncation points for each code block b. Notice that, as K increases, the set of local feasible truncation points more accurately represents the set of global feasible truncation points. Consequently, the size of K provides a tradeoff between the execution time and the effectiveness of coding segment selection.

Algorithm 4: Integrated Rate Control and Entropy Coding

1: **procedure** IREC(**set** $\{B_i\}$, **int** L, K, **set** $\{R_l\}$)
2: **int** R, l; **heap** h; **segment** s; **cblk** b; **int** n;
3: $l := 1$;
4: **for**(each $b \in \{B_i\}$) **do**
5: entropyencoder(b, K);
6: convexhull(b, K);
7: **end for**
8: $h :=$ makeheap$(\{B_i\})$;
9: $R := 0$;
10: **while** $(l \leq L)$ **do**
11: **while** $(R < R_l)$ **do**
12: $s :=$ first unmarked local feasible truncation point of root(h);
13: $R_c := R_c +$ rate(root(h));
14: $n :=$ number of coding passes in s;
15: entropyencoder$(b, K-n)$;
16: convexhull(root(h));
17: update(root$(h), l)$;
18: siftdown(root(h), key(root(h))$, h)$;
19: **end while**
20: $l := l + 1$
21: **end while**

[5]Note that, while a coding segment of size K will give the exact R-D ratio, the codec will not know that it is the exact value since the $K+1$th coding pass is not available.

We refer to the IREC scheme that generates only the next K coding passes as the IREC *local-K* scheme. Conversely, we refer to the original method, which first entropy codes all coding passes before doing any rate control, as the *global* scheme. Examining the IREC *local-K* scheme, we can see that the number of extra coding passes that have been generated but not yet selected is at most NK, where N is the total number of code blocks. So if the total number of coding passes to be included in the final bitstream is M_c, then the total number of coding passes that are entropy coded is $M_c + NK$. In contrast, the global scheme entropy codes all P_c coding passes in all N code blocks before doing rate control, for a total of NP_c coding passes entropy-coded. Consequently, the speedup in entropy coding for the IREC method can be up to $NP_c/(M_c + NK)$, where P_c is the total number of coding passes in a code block in average and is computed as $P_c = 3 * (n - 1) + 1$ for n bitplanes.[6] For example, assuming $M_c = NP_c/10$, or 10% of the coding passes are included in the final bitstream, then the IREC *local-K* scheme with $K = 3$ enables a speedup of approximately 4.2 over the *global* scheme.

The overall computation complexity of the IREC method is the sum of the computation complexities of the GHRaC rate control, which is $O(N + M_s d\log_d N)$, and the entropy coding in IREC. We saw above that the number of coding passes that are entropy coded is $M_c + NK$. If we let T_{EC} denote the average computation needed to entropy code each coding pass, then the computation complexity of entropy coding in IREC is $O((M_c + NK)T_{\mathrm{EC}})$. Therefore, the overall computation complexity of the IREC *local-K* scheme is upper-bounded by $O(N + M_s d\log_d N + (M_c + NK)T_{\mathrm{EC}})$. In practical applications, the contribution from entropy coding $O((M_c + NK)T_{\mathrm{EC}})$ dominates the computation complexity.

Similar to the GHRaC rate-allocation algorithm, the IREC algorithm can likewise be easily modified to solve the second R-D optimization problem, minimizing the bit rate under a specified quality constraint. The necessary modifications are straightforward, and the resulting computation complexity is the same.

C. Enhanced IREC Scheme With a Coefficient Weighting Model

While the IREC scheme is effectively a content-adaptive entropy-coding method, it can be further enhanced by incorporating coefficient weighting models. Specifically, for a particular set of wavelet filters, the corresponding coefficient weighting factors can be used to predict which coding passes among the many subbands and color components have a greater chance of being included in the final bitstream. We therefore propose using these coefficient weighting factors to further reduce the number of coding passes that must be entropy coded prior to rate control. In particular, we use the coefficient weighting factors, in conjunction with charateristics of the most significant bitplane of each code block, to estimate the R-D ratios of coding passes before entropy coding, and perform rate control based on these estimates. We shall refer to this the improved IREC scheme that combines coefficient weighting models as E-IREC, the *enhanced IREC local-K*, scheme.

In the E-IREC *local-K* scheme, before the heap data structure is initialized, the upper bound on the R-D ratio of the first coding segment in each code block is estimated without generating any coding passes. The R-D estimate is based on both its coefficient weighting factor and some characteristic statistics of the code block, such as the number of 1's in the most significant non-zero bitplane. The heap is then initialized using only the upper bound estimates of the R-D ratios for the code blocks. The benefit of this approach is that many of the code blocks with small estimates will never need to be entropy coded.

Similar to the IREC scheme, the E-IREC *local-K* scheme also iteratively selects coding passes to be included in the final bitstream. However, while the key of the root node in the original IREC scheme was always the R-D ratio computed for the first unselected coding segment, the key may now be an estimate of the R-D ratio of the first unselected coding segment. At the beginning of each iteration, if the key of the root node (the node with the maximum R-D ratio) is an estimated upper bound on the R-D ratio, which means that no actual coding passes have been generated for this code block, then E-IREC entropy codes the first K coding passes of the code block, and finds the first local feasible truncation point for it. Then E-IREC updates the key of this node with the actual R-D rate and adjusts the heap based on the updated information to preserve the heap properties. E-IREC then terminates this iteration and begins the next iteration. Note that no coding passes were selected for inclusion in the final bitstream in this iteration.

When E-IREC begins an iteration in which the key of the root node an actual R-D ratio, then E-IREC works exactly the same way as the original IREC scheme. First, the associated unselected coding segment is included in the final bitstream. Then, additional coding passes are entropy coded such that the total number of unselected coding passes equals K. Then, a local feasible truncation point is found for the code block, and the key of this node is updated to the R-D ratio of this next unselected feasible truncation point. Finally, the heap is adjusted based on the updated information to preserve the heap properties, and another iteration begins.

The advantage of E-IREC's approach over IREC is that many of the code blocks will never need any entropy coding. Whereas IREC requires that at least K coding passes be entropy coded for each code block, E-IREC only requires that the root node of the d-heap contain K entropy-coded coding passes. Other nodes in the heap can be merely estimates. Of course, as the heap is updated after each iteration, many of the code blocks will eventually become root nodes and have at least K coding passes entropy-coded. Conversely, many of the code blocks with small R-D estimates may never become the root node and will not require any entropy coding. The E-IREC method achieves a considerable speedup over the IREC method by not entropy coding these code blocks with small R-D estimates.

When analyzing the computation complexity of E-IREC, recall that, for the IREC *local-K* scheme, in addition to the coding passes included in the final bitstream, IREC entropy codes an additional K coding passes for each of the N code blocks. Thus, the total number of coding passes that are entropy-coded

[6]For example, if postquantization coefficients are represented using 8 b, then the total number of coding passes is $P_c = 22$.

TABLE II
DESCRIPTION OF TEST IMAGES

Name	Resolution	bpp	Comp.	Color
Fruits	512x512	24	3	Yes
Baboon	512x512	24	3	Yes
Lena	512x512	24	3	Yes
Barbara	512x512	8	1	No

is M_c+NK, where M_c is the total number of coding passes included n the final bitstream. Conversely, for the E-IREC *local-K* scheme, if a code block contributes nothing to the final bitstream, then there is a high probability that no coding passes will be generated for this code block. Consequently, only N_e code blocks will have an extra K coding passes entropy-coded for them, where $N_e \leq N$. We expect N_e will often be much less than N, and such was found in the experimental results, presented in Section V. However, the upper bound on the number of coding passes that E-IREC may generate is still the same as IREC, M_c+NK. Consequently, the overall computation complexity for E-IREC, including the computation complexity of GHRaC, is still the same as IREC, $O(N+M_s d\log_d N+(M_c+NK)T_{\mathrm{EC}})$, though the heuristic approach of E-IREC will generally enable significant speedup for E-IREC over IREC. Experimental results for both the IREC *local K* schemes and E-IREC *local-K* schemes are presented in Section V and are contrasted with the *global* scheme.

V. EXPERIMENTAL RESULTS AND COMPARISON

Here, we present the experimental results and performance comparisons. The test images used in the experiments are described in Table II. *Jasper* [19], which is a reference JPEG-2000 codec implementation is used as the test platform. The experimental parameters include the Daubechies $9/7$ floating point wavelet filter [8] with four levels of wavelet decomposition. The code block size is 32×32, so each color component includes 256 code blocks. The irreversible wavelet filter is used. Before performing entropy coding, the default quantization in *Jasper* is conducted to reduce the number of coding passes that need to be generated for the *global* scheme. The maximum number of iterations for the bi-section search in the generalize LM rate-control method is set to $I=32$. In packetization stage, the LRCP progression order is used in the experiments. The experiments were performed on an IBM Thinkpad with a 1.13-GHz Pentium III CPU and 256-M RAM running Redhat Linux with kernel version 2.4.16.

Since *Jasper* is not optimized for execution, to make the comparison fair and independent of specific implementations, the post compression rate-control procedure in *Jasper* has been separately rewritten and optimized, and the generalized LM method has been implemented in a more efficient way. For the integrated rate control and entropy coding, we use the *entropy-coding speedup* as the performance criterion, which is defined as the ratio of the total number of coding passes generated by the *global scheme* over the number of coding passes generated by the proposed scheme. With a reasonable assumption that the execution time and power consumption are proportional to the number of coding passes, this criteria can precisely exhibits the execution speedup independent of a specific implementation.

A. Performance of the GHRaC Rate-Control Algorithm

This section compares the performance of the GHRaC rate-control scheme to the generalized Lagrange multiplier method. The performance is compared for two situations, varying compression ratios for a single quality layer, and varying numbers of quality layers for a fixed compression ratio, respectively.

Before presenting the experimental results, one issue needs to be addressed. When generating the final bitstream, some header information must be included in addition to the coding passes, such as the main header and packet headers [1]. The number of bits needed by the main header can be readily obtained. Conversely, the number of bits needed for packet headers in each layer is more difficult to determine. One way to do this is to first select the coding passes, then test how many bytes are needed for the packet headers, as been done in some reference codecs [19]. However, this method is not particularly efficient at determining the size of packet headers since packet header length estimation takes quite some time. In this paper, we use a more efficient method to estimate the length of packet headers. We select coding passes until their accumulated rate is within a given distance of the desired final bit rate (in the following experiments, we use 90% of the desired rate for this layer), then estimate the length of the packet headers by simulating the header generation procedure, and using this value as the packet header length. Although this method may cause some small variations in bit rate (usually several bytes), the complexity is much lower since we only need to estimate packet header length one time for each layer. In the following experiments, we apply this packet header length estimation scheme for both the GHRaC algorithm and the generalized LM method.

The speedup of the proposed GHRaC scheme over the generalized LM algorithm is illustrated in Figs. 4 and 5. The speedup is defined as the ratio of the execution time for the generalized LM algorithm over the execution time for the GHRaC method. Figs. 4(a) and 5(a) present the experimental results at different compression ratios, when the number of layers is fixed to one, and Figs. 4(b) and 5(b) present the experimental results for different numbers of layers when the total compression ratio is fixed to 8:1.

As expected, in all of these experiments, the proposed GHRaC scheme always outperforms the generalized LM method, especially when the compression ratio is high or there is a large number of quality layers. Notice also that a speedup is achieved even with only one quality layer and minimal compression. The results show that even in such conservative cases at least $2\times$ speedup is achieved. As predicted by the formal analysis, speedup increases proportional to both the number of quality layers and the log of the compression ratio. Comparing Figs. 4 and 5, we can also see that the speedup in the case of code block size 32×32 is higher than in the case of code block size 64×64. The reason is that the number of generated code blocks with code block size 32×32 is much higher than the number of generated code blocks with code block size 64×64, which provides more optimization space for the proposed GHRaC rate-control scheme. However, the

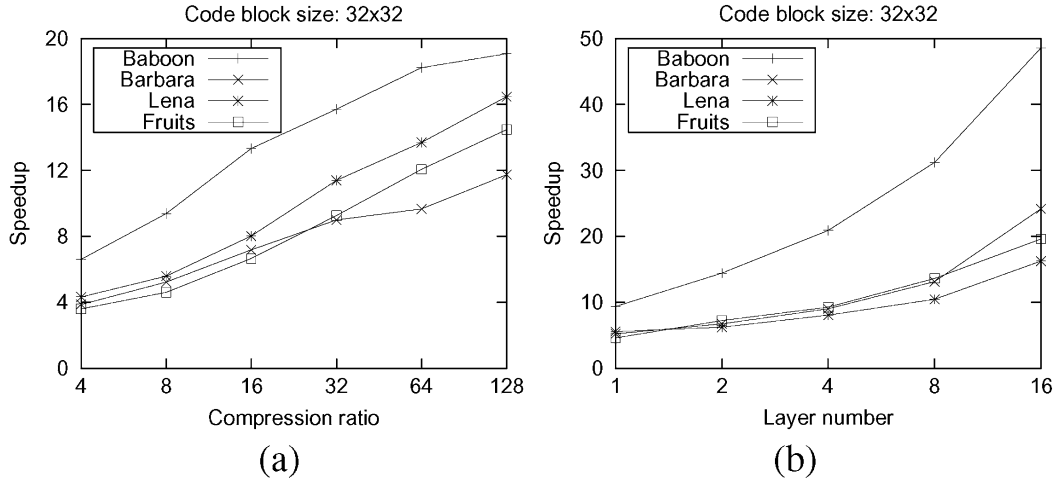


Fig. 4. Execution speedup of the GHRaC procedure over generalized LM method, code block size: 32×32.

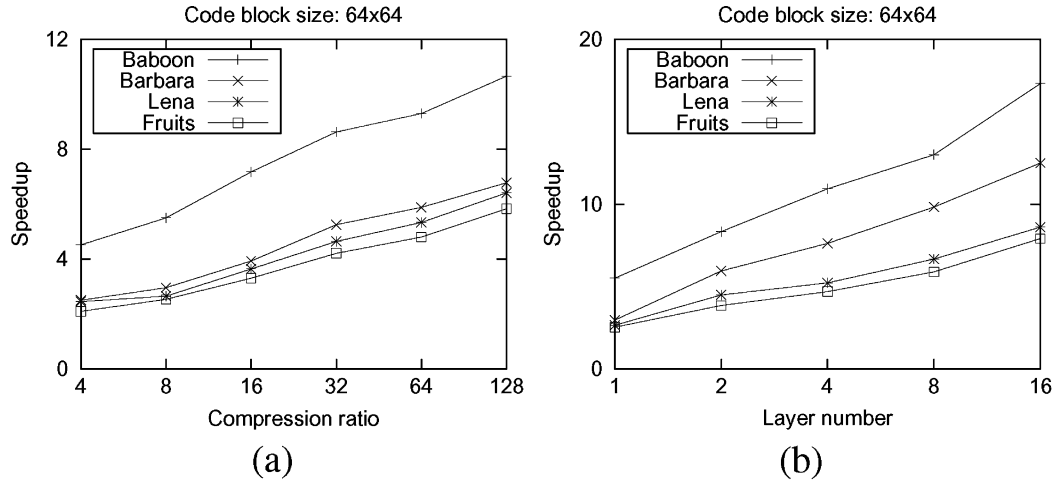


Fig. 5. Execution speedup of the GHRaC procedure over generalized LM method, code block size: 64×64.

tradeoff is that the working memory required in the case of code block size 32×32 is 4 times of the working memory required in the case of code block size 64×64.

Compression efficiency experiments have also been performed. In most situations, the proposed GHRaC rate-control scheme gives the same results as the LM method, and, in some situations, due to the imprecise header length estimation, the results may vary by several bytes. However, when such variation occurs, comparison of the disparate bitstreams shows that the one of the bitstreams is likely a superset of the other. In other words, bitstreams generated separately by the GHRaC method and the LM method will mostly contain the same set of coding segments, but the order of the segments may vary a little, with one of the bitstreams containing a separate or extra coding segment or two towards the end of the bitstream. Such variation in the bitstreams happens primarily when the methods encounter equivalent R-D ratios in different coding segments in the image; while one method will select one of the equivalent coding segments, the other method may select the other segment. Coding selection through the remainder of the image may then result in further disparity in the order or selection of coding segments, since the two methods will have different sets of possible coding segments available following this decision.

In addition to execution speed, memory requirements is also an important factor. In the proposed GHRaC scheme, the only extra memory requirement is the space to store the heap data structure. Since there are only N (N is the number of generated code blocks) elements in the heap data structure, the extra memory requirement is no more than $O(N)$ bytes, which is negligible compared with the image size.

B. Performance of the IREC Schemes

Here we present the experimental results of the proposed IREC schemes as well as the modified version. Both coding efficiency and entropy-coding speedup are presented. In Fig. 6, the experimental results for the four test images are presented, with the top graph illustrating the PSNR results and the bottom graph showing the entropy-coding speedups between the global scheme, the IREC local-2 scheme, IREC local-3 scheme, and IREC local-4 scheme. In these experiments, the code block size is set to be 32×32. In examining the results for PSNR, we can see that using as few as $K = 2$ coding passes for the IREC local-K gives very good coding segment selection, while using

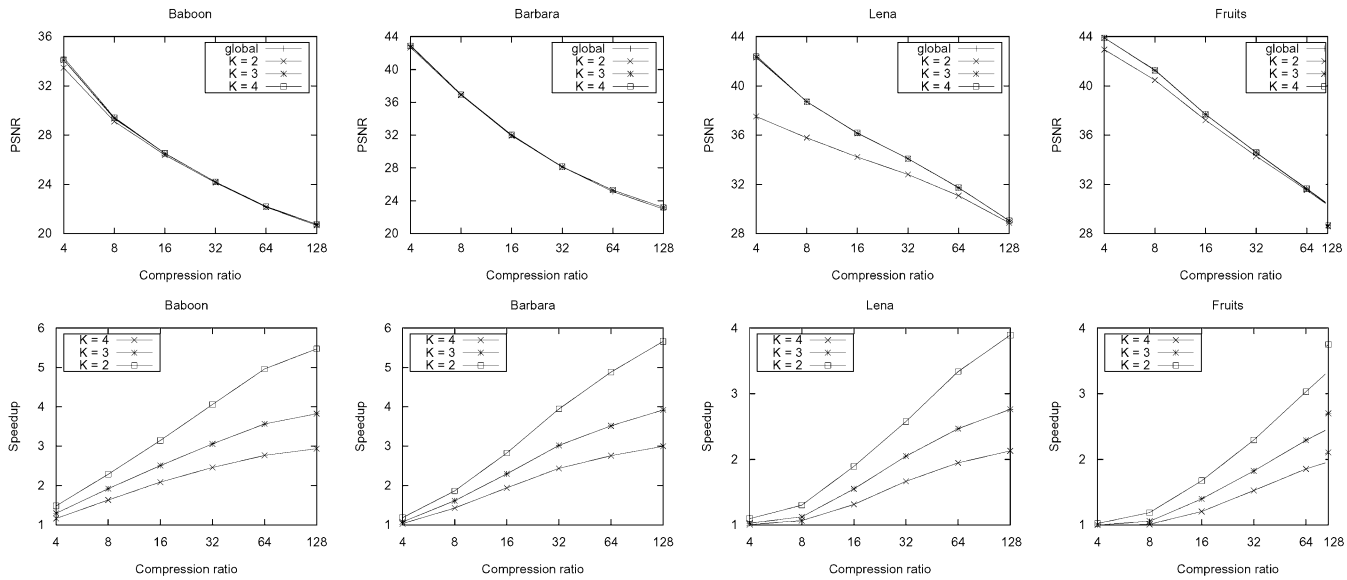


Fig. 6. Experimental results for the proposed IREC schemes.

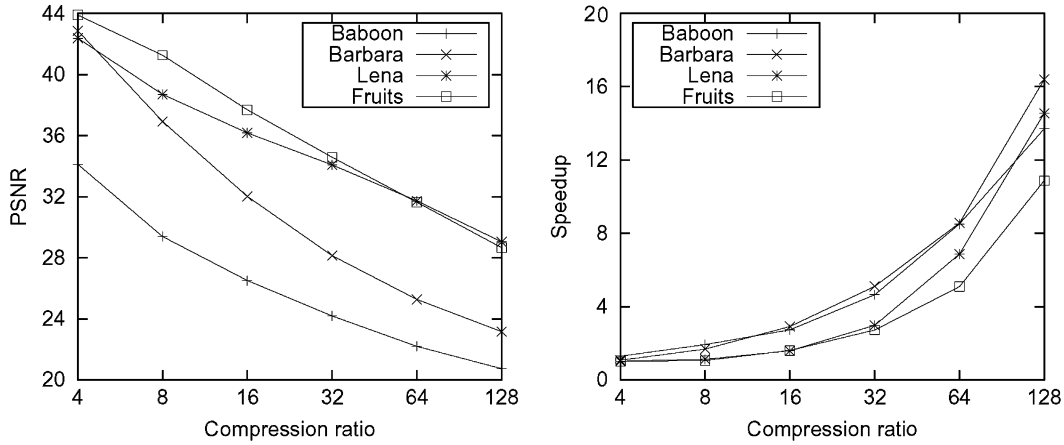


Fig. 7. Experimental results for the E-IREC local-3 scheme.

$K \geq 3$ coding passes provides results that are nearly always equivalent to the global scheme. In terms of speedup, we can see that the speed of entropy coding improves with decreasing K and with increasing compression ratio. For the particular case of $K = 3$ coding passes, it is evident that an average speedup of 3 can be achieved with negligible loss in PSNR.

However, as we discussed in Section IV, the number of coding passes that need to be generated can be further reduced by incorporating the coefficient weighting models. Experiments for the E-IREC local-K scheme have also been conducted and are presented in Fig. 7. In the experiments, the upper bound for the R-D ratio of each code block was estimated by: 1) counting the number of 1's in the first two most significant nonzero bitplanes of that code block, and calculating the distortion reduction for including these two bitplanes in the final bitstream; 2) estimating the total number of compressed bits in the two bitplanes by dividing the total number of bits in these two bitplanes by a constant, called the maximum compression ratio; and 3) generating a final estimate on the upper bound of the R-D ratio by dividing the distortion reduction with the number of compressed bits in these bitplanes. The maximum compression ratio is set to be 100. Fig. 7 shows that the PSNR results using the E-IREC local-3 scheme are the same as those from using the IREC local–3 scheme. These results are as expected, since the maximum compression ratio is sufficiently large. However, in comparing the speedup results, it can be seen that the E-IREC scheme enables much greater speedups than the IREC scheme, especially when at high compression rates.

VI. Conclusion

In this paper, we have addressed two core components in JPEG-2000, rate-control and entropy-coding procedures, and proposed two novel methods to reduce the computation complexity. First, we have proposed GHRaC, an efficient postcompression rate-control scheme by combining the greedy algorithm with the heap sort. GHRaC can speed up the postcompression rate-control procedure dramatically, especially when the compression ratio is high, or multiple quality layers are generated. Based on the proposed GHRaC rate-control scheme, some statistical information, the existing weighting model, and

the fact that the entropy coding is the most time and power consuming part, we have proposed IREC to reduce the computation complexity of entropy-coding procedure. The IREC scheme can reduce the unnecessary operations during the entropy-coding procedure and consequently speed up the overall system performance. Both theoretical analysis and experimental results show that the proposed scheme can dramatically speed up the execution and reduce the computation complexity of the entropy-coding procedure, which consequently reduces the the execution time and power consumption.

REFERENCES

[1] *Information Technology—JPEG 2000 Image Coding System—Part 1: Core Coding System* (in ISO/IEC), ISO/IEC 15444-1:2000: , 2000.

[2] D. Santa-Cruz, T. Ebrahimi, J. A. M. Larsson, and C. Christopoulos, "JPEG 2000 still image coding versus other standards," in *Proc. SPIE 45th Annu. Meeting Applic. Digit. Image Process. XXIII*, San Diego, CA, Jul. 2000, vol. 4115, pp. 446–454.

[3] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 still image coding system: An overview," *IEEE Trans. Consumer Electron.*, vol. 46, no. 4, pp. 1103–1127, Nov. 2000.

[4] B. Fox, "Discrete optimization via marginal analysis," *Management Sci.*, vol. 13, no. 3, pp. 210–216, Nov. 1966.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithm*. Cambridge, MA: MIT Press, 1990.

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

[7] I. Aouadi and O. Hammami, "Analysis and hardware design of a scalable dual JPEG-2000 entropy coder," in *Proc. Euromicro Symp. Digit. Syst. Design*, 2004, pp. 227–233.

[8] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Process*, vol. 1, no. 2, pp. 205–220, Feb. 1992.

[9] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with reversible embedded wavelets," in *Proc. Data Compression Conf.*, 1995, pp. 212–221.

[10] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Process.*, vol. 9, no. 7, pp. 1158–1170, Jul. 2000.

[11] S. Mallat, "A theory of multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 4, pp. 674–693, Jul. 1989.

[12] D. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Practice and Standards*. Norwell, MA: Kluwer, 2002.

[13] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Process. Mag.*, pp. 23–50, Nov. 1998.

[14] W. Yu, F. Sun, and J. Fritts, "An efficient packetization algorithm in JPEG2000," in *Proc. IEEE Int. Conf. Image Process.*, Rochester, NY, Sep. 2002, pp. 208–212.

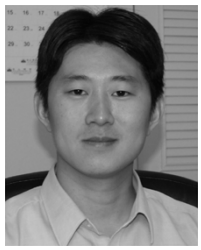
[15] B. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[16] R. Qiu, W. Yu, and J. Fritts, "Motion-JPEG2000 video transmission over active networks," *SPIE Electron. Imaging—Image and Video Commun. Process.*, pp. 350–357, Jan. 2003.

[17] W. Yu, Z. Sahinoglu, and A. Vetro, "Energy effcient JPEG 2000 image transmission over wireless sensor networks," in *Proc. IEEE GLOBECOM*, 2004, pp. 2738–2743.

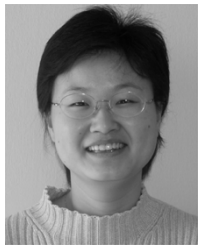
[18] T. Masuzaki, H. Tsutsui, T. Izumi, T. Onoye, and Y. Nakamura, "Adaptive rate control for JPEG2000 image coding in embedded systems," in *Proc. IEEE Int. Conf. Image Process.*, Rochester, NY, Sep. 2002, pp. 77–80.

[19] M. D. Adams, The JPEG-2000 still image compression standard Dec. 2002, ISO/IEC JTC 1/SC 29/WG 1 N 2412.



Wei Yu (S'04) received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2000, and the M.S. degree in computer science from Washington University, St. Louis, MO, in 2002. He is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Maryland, College Park.

From 2000 to 2002, he was a Graduate Research Assistant with Washington University. From 2002 to 2005, he was a Graduate Research Assistant with the Communications and Signal Processing Laboratory and the Institute for Systems Research, University of Maryland. His research interests include security, wireless communications and networking, game theory, and wireless multimedia.



Fangting Sun (S'05) received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2000, and the M.S. degree in computer science from Iowa State University, Ames, in 2002. She is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Maryland, College Park.

From 2000 to 2002, she was a Graduate Research Assistant with Iowa State University. From 2002 to 2006, she was a Graduate Research Assistant with the Department of Electrical and Computer Engineering, University of Maryland. Her research interests include networking, wireless communications, multimedia, and algorithms.

Jason E. Fritts (M'00) received the B.S. degree from Washington University, St. Louis, MO, in 1994, and the M.S. and Ph.D. degrees from Princeton University, Princeton, NJ, in 2000, all in electrical engineering.

From 2000 to 2005, he was an Assistant Professor with the Computer Science and Engineering Department, Washington University. In 2005, he joined Saint Louis University, St. Louis, MO, as an Assistant Professor with the Department of Mathematics and Computer Science. He is the Director of the MediaBench Consortium, whose goal is continued development and refinement of benchmark suites for multimedia systems research. His work spans a range of topics including media processing design and workload evaluation, objective image segmentation evaluation, content-based image retrieval, reconfigurable computer architecture, and multimedia systems.

Dr. Fritts is a member of the IEEE Communications Society.