一种基于系数状态表的 SPIHT 图像编码算法

张专成 武国斌 赵怀勋 闫小萍
(武警工程学院通信工程系,西安 710086)

摘 要 提出了一种新的基于系数状态表的 SPIHT(LPS-SPIHT, list of pixel stata-set partitioning in hierarchical trees)图像压缩编码算法,该算法具有以下 5 个特点:第一,定义了一种扩展的空间方向树,使 1 个结点含有 2×2 相邻的 4 个系数,并将基本 EZW(嵌入式小波零树)的符号定义应用于扩展树;第二,用 1 个廉价的系数状态表代替了 SPIHT 算法中的 LIS(不重要集合表)、LIP(不重要像素表)、LSP(重要像素表)等 3 个数据表,节省了内存;第三,通过扫描系数状态表,可一次性完成对图像数据的编码,使分类过程与细化过程合二而一;第四,利用一种树指数避免了重复计算,提高了处理速度;第五,通过重新组织编码过程,省去了对大量可推知位的编码,提高了压缩效率。实践证明,与目前公认的最为有效的 SPIHT 算法相比,该算法不仅性能优越,而且计算简单,容易实现。

关键词 系数状态表(LPS) 分级树集合分割(SPIHT) 基于系数状态表的分级树集合分割(LPS-SPIHT) 嵌入式编码 渐进量化 扩展树
中图法分类号: TN919.81 **文献标识码**: A **文章编号**: 1006-8961(2006)02-0162-07

A List of Pixel State Based SPIHT Image Coding Algorithm

ZHANG Zhuan-cheng, WU Guo-bin, ZHAO Huai-xun, YAN Xiao-ping
(*Department of Communication Engineering of Engineering College of the People's Armed Police Force of China, Xi'an* 710086)

Abstract In this paper, a new SPIHT(set partitioning in hierarchical trees) image compression coding algorithm based on the list of pixel states(LPS-SPIHT) is presented, which has five characters as follows: Firstly, a kind of extended spatial orientation tree is defined, which makes every point include four adjacent pixels of 2×2, and the essential signs of EZW (embedded zerotree wavelet) is applied to the extended spatial orientation tree; Secondly, the three lists of SPIHT(LIS、LIP、LSP) are substituted by a low-cost list of pixel states, saving the memory of program. Thirdly, the coding process is once accomplished by means of scanning the list of pixel states, making the sorting pass and refinement pass combine to one; Fourthly, using of one tree exponent gets rid of repeading computation and then results in, the speed up of the significance judgement of trees; Lastly, some predictable bits are omitted from the encoder output by rearranging the coding procedure, to reduce the redundancy of the coding and improve the compression efficiency. Practical experiments show that comparing with SPIHT algorithm which is among the best of existing coding methods, this algorithm not only has better performance, but also is easy to implement, especially, it provides a feasible referencing model for hardware design.

Keywords list of pixel state(LPS), set partitioning in hierarchical trees(SPIHT), LPS-SPIHT, embedded coding, successive-approximation quantization, extended spatial orientation tree

1 引 言

随着互联网,尤其是宽带技术的成熟和发展,人们对图像信息的传输和存储要求越来越高。而图像作为提供给人们视觉的一种特殊的 2 维感知信号,历来以其海量的数据制约着其有效地传输和存储。为了提高图像信息传输和存储的效率,人们一直在为寻找简单高效的图像数据压缩方法而不懈地努力。Shapiro 根据小波系数空间的结构特点,提出的 EZW(嵌入式小波零树)图像编码方法[1]用一个零树根的位置,代表相应空间位置上所有不重要系数的位置,

收稿日期:2004-10-20;**改回日期**:2005-06-09

第一作者简介:张专成(1948 ~),男。教授,硕士研究生导师。1982 年毕业于西安交通大学信息与控制工程系获自动控制专业学士学位。研究方向为数字图像处理。E-mail:zhangzhuancheng@yeah.net

简单而有效,极大地提高了压缩比,而且这种编码方法是嵌入式渐进量化的,可以根据位率和清晰度的要求随时停止,在这一点上 EZW 编码方法的性能超过了广泛应用的旧式 JPEG 编码方法。旧式 JPEG 编码只能让用户选择品质因子,而不能选择目标位率[1]。在 EZW 之后,又相继提出了许多对 EZW 的改进方案,有的增强了 EZW 算法的符号定义[2~4],有的改进了 EZW 算法的扫描过程[5],被认为最有效的一种改进是 SPIHT(分级树集合分割)算法[6]。文献[7]又对 SPIHT 算法提出了减少编码冗余与加快执行速度的改进方法。基于系数状态表的 SPIHT(LPS-SPIHT)图像压缩编码算法,是对 SPIHT 算法的又一改进,它具有 5 个特点,都是其他算法所没有的。这些特点是,第一,定义了一种扩展的空间方向树,使 1 个结点含有 2×2 相邻的 4 个系数,并将基本 EZW 的符号定义应用于扩展树;第二,用 1 个廉价的系数状态表代替了 SPIHT 算法中的 LIS(不重要集合表)、LIP(不重要像素表)、LSP(重要像素表)等 3 个数据表,节省了内存;第三,通过扫描系数状态表,可一次性完成对图像数据的编码,使分类过程与细化过程合二而一;第四,利用一种树指数避免了重复计算,提高了处理速度;第五,通过重新组织编码过程,省去了对大量可推知位的编码,提高了压缩效率。

2　小波零树概念及算法

2.1　零树概念及 EZW 算法

一幅 2 维图像经过 n 层小波分解后形成 $3n+1$ 个子频带,恰如金字塔结构,从顶层到底层的子带依次为 LL_n,HL_n,LH_n,HH_n,HL_{n-1},LH_{n-1},HH_{n-1},…,HL_1,LH_1,HH_1。其中,LL_n 表示经过 n 级分解后水平和垂直方向均为低频的子带;HL_i 表示水平高频、垂直低频的子带;LH_i 表示水平低频、垂直高频的子带;HH_i 表示水平和垂直方向均为高频的子带。LL_n 子带只有 1 个,处于空间分辨率最低的位置,可对其继续进行小波分解,其余子带均为 n 个。小波分解系数可看作一系列 4 叉树,如果把 1 个系数看作 1 个结点,那么第 i 层上 1 个结点,在第 $i-1$ 层上有 4 个子结点。

对于给定的阈值 T,如果小波系数 $|c(x,y)|<T$,则称 $c(x,y)$ 是不重要的,反之,如果 $|c(x,y)|\geq T$,则称 $c(x,y)$ 是重要的。如果以 $c(x,y)$ 为根的空间树上包括 $c(x,y)$ 在内的所有系数都是不重要的,则称该空间树为零树,其上所有系数的坐标位置,都可由 $c(x,y)$ 的坐标来确定。由于图像经过小波分解之后,能量不断向低频子带集中,按照空间分辨率由低向高的方向,即由树根到树叶的方向,系数绝对值呈下降的趋势,因此,如果某一处在较低分辨率空间上的系数是不重要的,那么其处在较高分辨率空间上的后代,会以极大的概率全部表现为不重要的,于是可以用一个树根坐标表示位于这个树上的所有系数坐标,这就是零树压缩的依据,而且实践证明,即使处在较低分辨率空间上的某一系数是重要的,其处在较高分辨率空间上的后代,也会以极大的概率全部表现为不重要的,这一点促使了零树概念的发展和 EZW 编码算法的改进。一种改进方向就是增强零树符号的定义,比较典型的增强以后的符号是 ZTR、IZ、VZTR、VL,其中,ZTR 表示该系数及其后代都是不重要的,即零树;IZ 表示该系数是不重要的而其后代中至少有 1 个系数是重要的,即孤立零;VZTR 表示该系数是重要的,而其后代都是不重要的;VL 表示该系数是重要的,而其后代中至少有 1 个系数也是重要的。实践证明这种增强的符号定义对提高压缩比是十分有效的。

2.2　SPIHT 算法

对 EZW 算法的另一种改进就是文献[6]提出的分级树集合分割算法,即 SPIHT 算法,它是目前公认的基于小波变换的最好的图像编码方法。SPIHT 算法的成功之处,在于它不仅考虑了单个系数的零树特性,而且还考虑了 2×2 相邻系数之间的相关特性,从而更加有效地说明了小波系数的空间频率局部化特性和子带之间的相似性。SPIHT 算法对以坐标 (i,j) 为根的树,定义了 3 种系数集合,它们是所有后代系数集合 $\boldsymbol{D}(i,j)$,儿子系数集合 $\boldsymbol{O}(i,j)$ 与除儿子以外的所有后代系数集合 $\boldsymbol{L}(i,j)$,如图 1 所示。$\boldsymbol{O}(i,j)$ 是 $\boldsymbol{D}(i,j)$ 的一种特殊情况,$\boldsymbol{L}(i,j)=\boldsymbol{D}(i,j)-\boldsymbol{O}(i,j)$。在处理过程中,所有系数都被组织在 3 个表中,它们是不重要集合表 LIS,不重要像素表 LIP,重要像素表 LSP。在 SPIHT 算法中,树根坐标,只不过是树的索引,并不影响树的特性。初始化阶段,将位于金字塔顶层的低频子带系数全部放入 LIP,将位于金字塔第 2 层的 3 个高频子带的系数在低频子带中对应的树根坐标标志为 A 类放入 LIS,将 LSP 清空。分类扫描阶段,首先检查 LIP,在输出重要系数标志的同时,将重要系数移入 LSP,然后检查 LIS。对 LIS 中的 A 类树根坐标

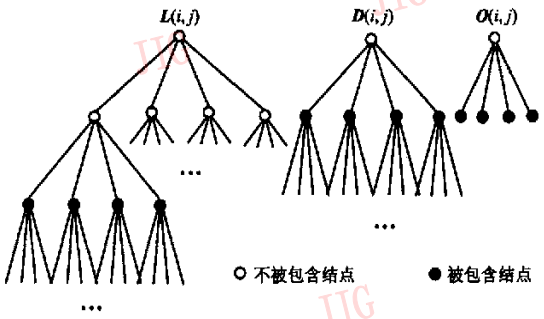

图 1 SPIHT 的 3 种树结构

Fig. 1 Three types of trees in SPIHT

(i,j)，检查 $\boldsymbol{D}(i,j)$，若 $\boldsymbol{D}(i,j)$ 中所有系数都是不重要的，输出代码 0；若 $\boldsymbol{D}(i,j)$ 中至少有 1 个系数是重要的，则要完成 2 个任务，一是将该树根坐标标志为 B 类，移到 LIS 末尾，二是输出代码 1xxxx，其中，xxxx 表示 $\boldsymbol{O}(i,j)$ 中 4 个系数的重要系数标志，与此同时，将不重要系数添加到 LIP 末尾，将重要系数添加到 LSP 末尾。对 LIS 中的 B 类树根坐标 (i,j)，检查 $\boldsymbol{L}(i,j)$，若 $\boldsymbol{L}(i,j)$ 中所有系数都是不重要的，输出代码 0；若 $\boldsymbol{L}(i,j)$ 中至少有 1 个系数是重要的，删去该树根坐标，将 $\boldsymbol{O}(i,j)$ 的 4 个坐标作为新的树根坐标，标志为 A 类添加到 LIS 末尾，并输出代码 1。和 EZW 算法一样，分类扫描过程结束之后，开始对 LSP 中的系数进行细化。

纵观 SPIHT 的分类扫描过程，若把某一树根在 A 类时候的输出，与在 B 类时候的输出结合在一起，就可获得关于某一 2×2 相邻系数块的完整的代码输出，当该 2×2 相邻系数块及它们的后代中无重要系数时输出代码为 0；当该 2×2 相邻系数块中至少有 1 个重要系数，而它们的后代中无重要系数时输出代码为 1xxxx0；不管该 2×2 相邻系数块中有无重要系数，只要它们的后代中至少有 1 个重要系数时输出代码为 1xxxx1。其中，xxxx 表示该 2×2 相邻 4 个系数的重要系数标志。这里所谓重要系数标志是指，若系数是不重要的，输出 0；若系数是重要的，输出 10(正数)或 11(负数)。

3 LPS-SPIHT 算法的实现

3.1 扩展树定义

如前所述，在 SPIHT 算法中，树根坐标，只不过是树的索引，并不影响树的特性。针对这一特点，可以构造一种新的空间方向树，在这种空间方向树中，其结点不再是 1 个系数，而是 2×2 相邻的 4 个系数，结点索引用 2×2 相邻系数块左上角系数的坐标来表示，为了叙述方便，将这样获得的结点索引称为结点坐标。这种扩展的空间方向树(以下简称为扩展树)也是一个 4 叉树结构，如图 2 所示，结点 (i,j) 有 4 个儿子结点，它们的坐标是 $(2i,2j)$、$(2i,2j+2)$、$(2i+2,2j)$ 及 $(2i+2,2j+2)$，用 $\boldsymbol{O}(i,j)$ 表示儿子结点集合。

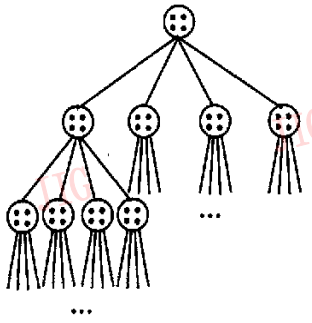

图 2 LPS-SPIHT 算法的扩展树结构

Fig. 2 Extened tree structure in LPS-SPIHT

在这里，所谓结点是不重要的，是指这个结点上所有系数都是不重要的；所谓结点是重要的，是指这个结点上至少有 1 个系数是重要的，用函数 $S(i,j)=1$ 或 0 来表示结点 (i,j) 是重要的或不重要的。可以用类似于 EZW 的方法定义 4 个符号 ZTR、IZ、VZTR、VL，这 4 个符号的基本编码模型分别为 0、111、10xxxx 和 110xxxx。由于 1 个结点中含有 4 个系数，对扩展树来说，VZTR 及 VL 的编码除符号标志外，还必须伴有重要系数标志 xxxx，以便区分哪些系数是不重要的，哪些系数是重要的。当然对金字塔底层(HL_1、LH_1、HH_1 子带)的结点，因为它们没有后代，所以若是重要结点，输出 1xxxx，反之，输出 0(等于 ZTR)。

3.2 系数状态表

系数状态表是一个与图像大小相等的字节型矩阵，用 $\boldsymbol{PS}$ 表示，其结构与图像的小波系数结构完全一致。扩展树的 1 个结点 (i,j)，对应 $\boldsymbol{PS}$ 中 2×2 相邻的 4 个字节，它们是 $PS(i,j)$、$PS(i,j+1)$、$PS(i+1,j)$、$PS(i+1,j+1)$。其中，$PS(i,j)$ 描述结点状态，$PS(i,j+1)$ 存放树指数，$PS(i+1,j)$ 及 $PS(i+1,j+1)$ 未用。结点状态字节由 3 个数据段组成，如图 3 所示，其中，重要系数标志 $S_3 \sim S_0$ 与 2×2 相邻 4 个系数一一对应，$S_i=1$，表示相应系数

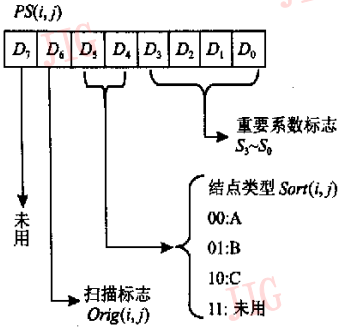

图 3　结点状态字节

Fig. 3　Node state byte

是重要的,反之,则是不重要的;结点类型 $Sort(i,j)$ 等于 0、1、2 时,分别表示结点类型 A、B、C,其中 A 类和 B 类与 SPIHT 算法中的 A 类和 B 类一样,分别表示不重要集合 $\boldsymbol{D}(i,j)$ 和 $\boldsymbol{L}(i,j)$,C 类节点表示只需要输出其 4 个系数的重要性标志或细化值;扫描标志 $Orig(i,j)$ 有 2 个状态,$Orig(i,j)=1$ 表示该结点还没有从父辈结点所表示的零树中分化出来,仍处于原始状态,对这类结点在编码扫描过程中什么也不做,反之,用 $Orig(i,j)=0$ 表示;树指数为

$$Exp(i,j)=\mathrm{Int}(\log_2 C_{\max}(i,j))$$

式中,$C_{\max}(i,j)$ 表示以结点 (i,j) 为根的空间方向树上最大的系数绝对值,为了应用方便,规定树指数函数的变元可以是结点集合,例如,$Exp(\boldsymbol{O}(i,j))$ 是指 $\boldsymbol{O}(i,j)$ 中 4 个结点的树指数最大值。

引入树指数的目的是为了加快扫描速度。大家知道,小波零树编码在分类扫描过程中,必须对每个树的重要性进行编码。为了判断树的重要性,可以直接把树上每个系数的幅度与阈值进行比较。因为树可能是不重要的,或者说树上的许多系数深藏在经过分支以后的子树上,所以用这种方法,绝大多数系数都要被反复检查许多次,从而降低了扫描速度。引入树指数以后,在分类过程中,判断以结点 (i,j) 为根的扩展树在阈值 $T=2^k$ 时是否为重要,只要判断 $Exp(i,j)$ 是否大于或等于 k 就可以了。

3.3　算法描述

3.3.1　初始化系数状态表

(1) 对 LL_n、LH_n、HL_n 及 HH_n 子带,令 $Orig(i,j)=0$,其余,令 $Orig(i,j)=1$。

(2) 对 LL_n 子带,令 $\mathrm{Sort}(i,j)=\mathrm{C}$,其余,令 $\mathrm{Sort}(i,j)=\mathrm{A}$。

(3) 计算树指数。如图 4 所示,计算树指数的扫描顺序从金字塔底层开始,到顶层结束,正好与图 5 所示的编码扫描顺序相反,这样,计算较低分辨率空间上某一结点的树指数时,只要根据它的 4 个系数的最大绝对值以及已经计算出来的它的 4 个子结点的树指数结果就可以了,不需要再用树上所有的系数来计算。

(4) 将所有的重要系数标志置 0。

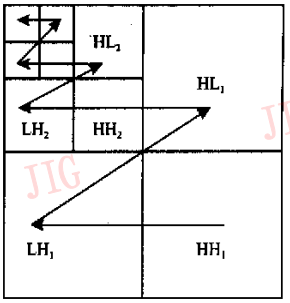

图 4　计算树指数扫描顺序

Fig. 4　Scanning sequence of computing tree exponent

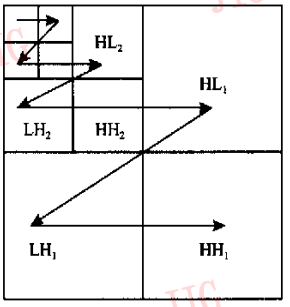

图 5　编码扫描顺序

Fig. 5　Scanning sequence of coding

3.3.2　编码查询处理流程图

阈值 $T=2^k$ 时,对结点 (i,j) 的编码查询处理流程图如图 6 所示,其中的输出就是编码所产生的比特流。由流程图可知:

(1) 如果扫描标志 $Orig(i,j)=1$,什么也不做,否则继续往下查询。

(2) 对 A 类结点,检查其树指数是否小于当前阈值指数 k,如果是,输出 0(ZTR);否则,检查该结点是否在金字塔底层,如果是,输出 1xxxx,修改重要系数标志 S_i,结点类型变为 C;否则,检查 4 个子结点的树指数是否都小于 k,如果是,输出 10xxxx (VZTR),修改重要系数标志 S_i,结点类型变为 B;否

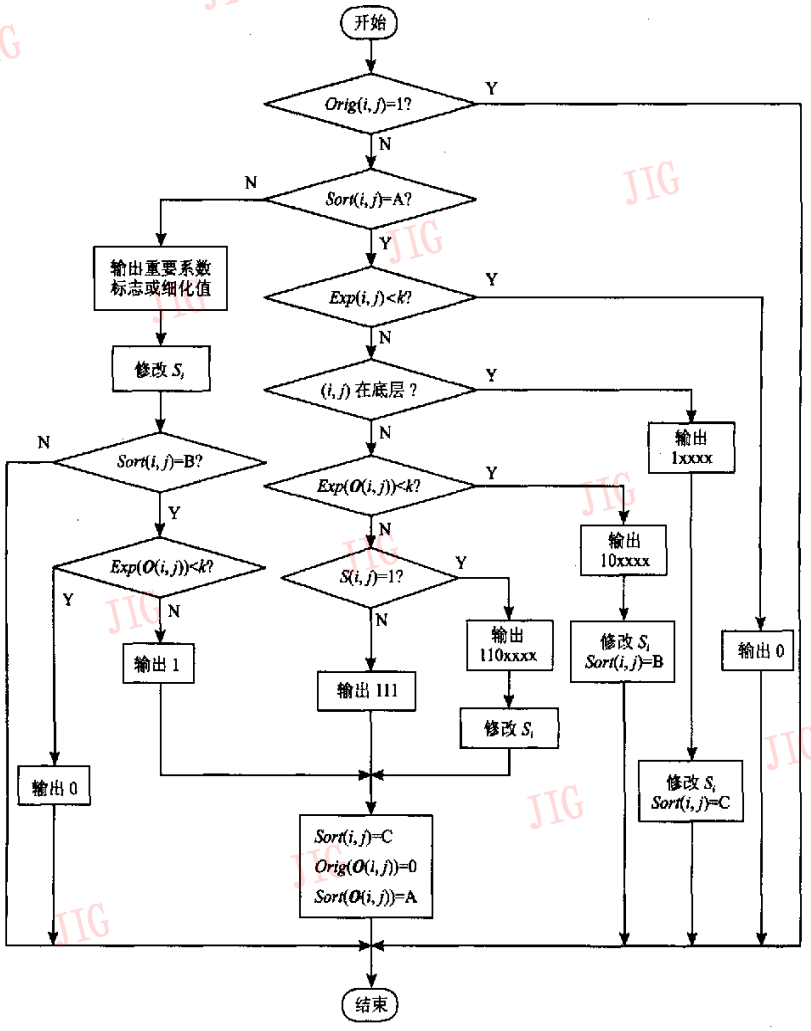

图 6 阈值 $T=2^k$ 时对结点 (i, j) 的编码查询处理流程图

Fig. 6 The coding flow diagram for node (i, j) at $T=2^k$

则，根据结点 (i, j) 中有无重要系数，输出 111(IZ)或 110xxxx(VL)并修改重要系数标志 S_i，执行功能框，即将结点类型变为 C，4 个子结点的扫描标志置零，结点类型设置为 A。

(3) 对 C 类结点，输出重要系数标志或细化值，修改重要系数标志 S_i。

(4) 对 B 类结点，接着 C 类操作之后，再检查 4 个子结点的树指数是否都小于 k，如果是，输出 0；否则，输出 1，然后合流于流程 2，执行同一功能框。

流程图中的所谓输出重要系数标志和细化值是指如果某一系数在以前阈值下是不重要的，那么在当前阈值下若是重要的，就输出 10(正数)或 11(负数)，否则输出 0，如果在以前阈值下是重要的，就输出该系数绝对值与当前阈值相对应的有效位(0 或 1)；所谓修改重要系数标志 S_i 是指一旦发现重要系数就将系数状态表中相应的重要系数标志位置 1。

显而易见，图 6 所示的 LPS-SPIHT 算法的处理过程与 SPIHT 算法的处理过程基本上是一致的，只不过 LPS-SPIHT 算法把有关信息统一存放在了 1 个系数状态表中，而不是像 SPIHT 算法那样，把它们分别存放在 3 个数据表中，即 LIS、LIP 及 LSP。在某一阈值 T 下，LPS-SPIHT 算法按图 5 所示的顺序扫描系数状态表，根据结点状态来决定当前应该输出的是重要系数标志，还是系数的细化值，或者是零树符号，这样就把分类过程与细化过程融为了一体。

在阈值 T 下编码过程结束之后，令 $k=k-1$，开

始下一个量化层上的编码过程,直至满足所需的精度要求。解码为编码的逆过程。

4 LPS-SPIHT 算法的效率

4.1 处理速度

无论是基本的 EZW 算法,还是 SPIHT 算法,在分类扫描过程中,对 1 个树,总是要从根结点一直检查到叶子结点,才能决定其是否为重要集合,这种周而复始的重复计算占用了大量机时。LPS-SPIHT 算法依靠系数状态表,一次完成分类和细化两个功能,在进行分类操作时,只需要判别 1 个树指数,就可决定其是否为重要集合,而树指数又是在初始化系数状态表时,按图 4 所示顺序一次计算出来的,避免了大量的重复计算,提高了处理速度。此外,速度的提高,还在于 LPS-SPIHT 算法用 1 个固定的系数状态表代替了 SPIHT 算法中的 3 个数据表,不需要把机时耗费在内存的动态分配上。测试结果表明,在同一信噪比下,LPS-SPIHT 所用时间大约为 SPIHT 所用时间的 83% 左右。

4.2 内存占用

LPS-SPIHT 算法节省内存,是不言而喻的事实,因为它用 1 个字节类型的系数状态表代替了 SPIHT 算法中的 3 个数据表。

4.3 压缩效率

为了便于比较两种算法的压缩效率,将两种算法的编码模型列在表 1 中。

表 1　LPS-SPIHT 与 SPIHT 编码模型比较

Tab. 1　Code model comparison between LPS-SPIHT and SPIHT

	LPS-SPIHT 编码	对应 SPIHT 编码
ZTR	0	0
IZ	111	1 0000 1
VZTR	10 xxxx	1 xxxx 0
VL	110 xxxx	1 xxxx 1

LPS-SPIHT 算法压缩效率的提高,主要表现在以下两个方面:

(1)符号编码最优化　LPS-SPIHT 算法由于明确地定义了 4 种符号,所以有利于实现基于统计的最大熵编码。表 2 就是对 512 × 512 × 8bit 的 Lena 标准测试图,采用 9/7 小波进行 5 级分解后,各类符号在不同位平面出现的次数统计,表 2 中的 D_{max} 表示最高有效位平面,其余依此递减。根据统计数字,LPS-SPIHT 把 ZTR、VZTR、VL、IZ 分别编码为 0、10、110、111,这是一种典型的最大熵霍夫曼编码,其平均码长最短。由表 1 不难发现,对 ZTR、VZTR 的编码,LPS-SPIHT 与 SPIHT 是完全等效的。对 VL 的编码,LPS-SPIHT 增加了 1bit,但是对 IZ 的编码,LPS-SPIHT 又减少了 3bit。表 2 的统计数字显示,一般情况下,VL 出现次数不会超过 IZ 出现次数的 3 倍,所以总的统计结果是 LPS-SPIHT 算法对符号的编码优于对应的 SPIHT 算法的编码。

表 2　各个位平面上扩展树 4 种符号的出现次数统计

Tab. 2　The statistic of 4 diferent symbols of entended tree at each bit-plane

	位平面									
	D_{max}	D_{max-1}	D_{max-2}	D_{max-3}	D_{max-4}	D_{max-5}	D_{max-6}	D_{max-7}	D_{max-8}	D_{max-9}
ZTR	30	66	205	648	1 665	3 615	7 196	12 837	21 254	33 306
VZTR	21	56	137	384	939	2 071	3 596	5 338	7 374	9 559
VL	1	6	27	62	162	311	548	921	1 643	2 975
IZ	0	2	5	51	98	162	248	414	753	1 144

(2)重要系数标志编码最优化　如表 1 所示,LPS-SPIHT 算法的编码由符号段与重要系数标志段 xxxx 两部分组成。xxxx 用来区分结点中那些系数是重要的,那些系数是不重要的。为了减少冗余,提高编码效率,可根据 xxxx 中所含重要系数的多少采用不同的编码。为此,LPS-SPIHT 算法对 xxxx 中重要系数的个数分布进行了统计,统计结果列在表 3 中(数据来源同表 2)。由表 3 可以看出,绝大多数重要结点,仅含 1 个重要系数,其次是含有 2 个重要系数。LPS-SPIHT 算法把重要系数标志编码分为两种情况,一种情况是只含 1 个重要系数,编码为 0xx,其中,xx 表示重要系数的位置,另一种情况是含 2 个或 2 个以上重要系数,编码为 1xxxx。在后一种情况下,含有 2 个重要系数的情况又占绝大多数,

表 3 各个位平面上重要结点中重要系数的个数分布统计

Tab. 3 The number distribution statistic of significant pixels in significant node at each bit-plane

	位平面									
	D_{max}	D_{max-1}	D_{max-2}	D_{max-3}	D_{max-4}	D_{max-5}	D_{max-6}	D_{max-7}	D_{max-8}	D_{max-9}
结点数	29	73	178	462	1 117	2 480	4 939	9 385	16 882	28 620
1 个	21	45	122	361	748	1 577	3 078	5 795	10 360	17 276
2 个	7	17	40	112	308	756	1 512	2 848	5 017	8 498
2 个以上	1	11	16	34	61	147	349	742	1 505	2 846

对此，如果前面两个系数都是不重要的，后面两个系数必然都是重要的，如果前面 3 个系数中只有 1 个是重要的，最后 1 个系数必然是重要的，这些可推知的重要系数标志的编码都可以省去。总结以上各种情况，若用 ΔB 表示可减少的比特数，X 表示重要结点总数，P_1 表示重要结点中仅含 1 个重要系数的概率，P_2 表示含有 2 个重要系数的概率，那么可得：

$$\Delta B = P_1 X + 2P_2 X/3 - (1 - P_1)X$$
$$= (2P_1 + 2P_2/3 - 1)X$$

上式表明，如果重要结点中仅含 1 个重要系数的概率 P_1 大于 0.5，按照上述方法对重要系数标志编码，都可以提高压缩比，根据表 3 所给统计数据可求得 $P_1 \approx 0.6 \sim 0.78$，所以，LPS-SPIHT 算法对重要系数标志采用上述方法编码使压缩比有了显著提高。

5 实验结果与结论

为了验证 LPS-SPIHT 图像压缩编码算法的高效性，选择 9/7 小波[8]，以 512 × 512 × 8bit Lena 标准测试图和 Barbara 标准测试图为例，用 LPS-SPIHT 算法进行了 5 级小波分解的压缩编码实验，并与 SPIHT 算法进行了比较，实验结果如表 4 所示。实验结果表明，对 Lena 图，LPS-SPIHT 算法比 SPIHT 算法的峰值信噪比提高了 0.14 ~ 0.28dB，对 Barbara 图，LPS-SPIHT 算法比 SPIHT 算法的峰值信噪比提高了 0.03 ~ 0.29dB。LPS-SPIHT 算法采用系数状态表结构，大大地节省了内存，避免了重复计算，提高了处理速度，并把分类与细化融为一体，简化了处理过程，有利于硬件设计。

表 4 LPS-SPIHT 和 SPIHT 两种算法压缩编码后的峰值信噪比对照

Tab. 4 PSNR Comparison for image coded of LPS-SPIHT and SPIHT

单位：dB

图像	算法	位率(bpp)				
		1.0	0.75	0.5	0.25	0.125
Lena	LPS-SPIHT	40.64	39.18	37.41	34.39	31.31
	SPIHT	40.40	39.04	37.21	34.11	31.09
Babara	LPS-SPIHT	36.69	34.39	31.68	27.77	24.88
	SPIHT	36.41	34.25	31.39	27.58	24.85

参考文献(References)

1 Shapiro J M. Embedded image coding using zerotrees of wavelet coefficients[J]. IEEE Transactions on Signal Processing, 1993, 41(12): 3445 ~ 3462.

2 Muzaffar T, Choi T S. Simplified EZW image coder with residual data transmission[A]. In: Proceedins of IEEE International Conference on Multimedia and Expro(I)[C], New York City, NY, USA, 2000: 111 ~ 114.

3 Rajpoot N, Wilson R. Progressive image coding using augmented zerotrees of wavelet coefficients[R]. Research Report CS-RR-350, UK: Department of Computer Science, University of Warwick, 1998.

4 Barreto C S, Mendonca G V. Enhanced zerotree wavelet transform image coding exploiting similarities inside subbands[A]. In: Proceedings of IEEE Conference on Image Processing[C], Lausanne, Switzerland, 1996, 2: 549 ~ 551.

5 ZHANG Hai-xiang, CHEN Chun, ZHUANG Yue-qing. Embedded zerotree wavelet image coding algorithm based on single list and recursive scan[J]. Journal of Image and Graphics, 2002, 7(7): 709 ~ 715. [张海翔，陈纯，庄越挺. 基于单队列递归扫描的嵌入式零树图象编码方法[J]. 中国图象图形学报, 2002, 7(7): 709 ~ 715.]

6 Said A, Pearlman W A. A new fast and efficient image codec based on set partitioning in hierarchical tree[J]. IEEE Transactions on Circuits and systems for Video Technology, 1996, 6(6): 243 ~ 249.

7 JIAN Zhu, Lawson S. Improvements to SPHIT for lossy image coding[J]. International Conference on Electronics, Circuits, and Systems, 2001, 3: 1363 ~ 1366.

8 Antonini M, Barlaud M, Mathieu P. Image coding using wavelet transform[J]. IEEE Transactions on Image Processing, 1992, 1(2): 205 ~ 220.