



交流图片

图像和视频编码课程

2014 年, 第 213-253 页

第7章-无损压缩方法

大卫·R·布尔

显示更多 ▾

大纲 | 共享 引用

<https://doi.org/10.1016/B978-0-12-405906-1.00007-6>

获取权利和内容

摘要

本章介绍了无损图像编码的方法；这意味着压缩后，输入信号可以精确重建。它首先回顾了无损压缩的动机和要求，并证明了使用可变长度符号编码的理由。然后，它专注于两种主要方法，霍夫曼编码和算术编码。探讨了这两种方法的算法、性质和局限性，以及一些性能比较。基于无处不在的JPEG编码标准提供了一个完整的示例，并通过CABAC和CAVLC在HEVC和H.264/AVC标准中使用无损编码方法提供了更多详细信息。通篇提供了示例。

 [上](#)[下一](#)

关键词

无损图像编码; 霍夫曼编码; JPEGContext; -自适应可变长度编码 (CAVLC) ; H.264/AVC; Golomb; 编码; 算法编码; 上下文自适应二进制算术编码 (CABAC)

本章介绍了无损图像编码的方法；这意味着压缩后，输入信号可以精确重建。这些有两个主要用途。首先，当应用程序要求任何类型的损失都是不可接受的时，它们是有用的。例子包括一些医疗应用，其中压缩工件可以被视为影响诊断，或者在法律案件中，文件被用作证据。其次，更常

见的是，它们被用作无损压缩技术中的组件，结合变换和量化，以提供所生成系数的熵效率表示。

我们首先回顾了无损压缩的动机和要求，然后重点关注霍夫曼编码和算术编码这两种主要方法。霍夫曼编码被广泛使用，例如在大多数JPEG编解码器中，我们证明它可以与DCT结合，生成最小平均码字长度的压缩比特流。我们在第7.3节中探讨了霍夫曼编码的性质和局限性，然后在第7.6节中引入了另一种方法——算术编码。这是一种优雅灵活但简单的符号字符串编码方法，而不是像霍夫曼编码那样的单个符号。最后，第7.7节介绍了霍夫曼和算术编码之间的一些性能比较。

7.1。无损图像压缩的动机

对于某些图像压缩应用程序，解码信号是原始信号的精确复制品很重要。这些方法被称为“无损”，因为编码器和解码器之间没有信息丢失。它们不利用心理视觉冗余，但确实利用统计冗余来实现接近源熵的比特率。

使用无损压缩的论据可以总结如下：

- 对于一些应用程序，不确定哪些图像特征很重要，以及这些特征将如何受到有损图像编码的影响。例如，在关键的医学应用中，诊断或解释中的错误可以合法地归因于图像压缩过程引入的人工制品。
- 在某些应用程序中，数据必须完好无损地保存——例如计算机文件中的文本或数字信息。在这种情况下，如果数据被损坏，它可能会失去价值或故障。
- 无损压缩方法可以为表示有损压缩系统中变换系数量化产生的稀疏矩阵提供了一种有效方法。**空间**是通过使用装饰和量化来利用视觉冗余引入的，无损方法可以进一步利用符号的统计信息，以实现接近源熵的比特率。

7.1.1。应用程序

早期无损压缩的一个很好的例子是摩尔斯电码。为了减少通过电报系统发送的数据量，塞缪尔·莫尔斯在19世纪开发了一种技术，使用较短的码字来表示更常见的字母和数字。他的二进制（二合字母）系统基于点和破折号，有一个空格来分隔字母，一个更长的空间来指示单词之间的界限。例如，“a”和“e”是常见的字母，而“q”和“j”出现的频率较低，因此莫尔斯使用映射：

e ↠ . a ↠ .- q ↠ ---.- j ↠ .---

在实践中，摩尔斯信号是通过打开和关闭振荡器的音调来实现的，音调更长表示破折号，点持续时间更短。英文字母表字母的摩尔斯电码的完整结构如图7.1所示。未显示的其他符号，如数字和特殊字符，包括“CH”等常见字母对。注意图中使用的树状结构表示形式，左分支和右分支分别标记为点和破折号。稍后，当我们检查霍夫曼代码时，我们会看到类似的结构。还应当指出，摩尔斯映射要求码字之间的空间——否则读者无法区分两个“E”和一个“I”。我们稍后将看到霍夫曼编码如何巧妙地克服摩尔斯变长度编码的这一限制。

示例7.1摩尔斯电码

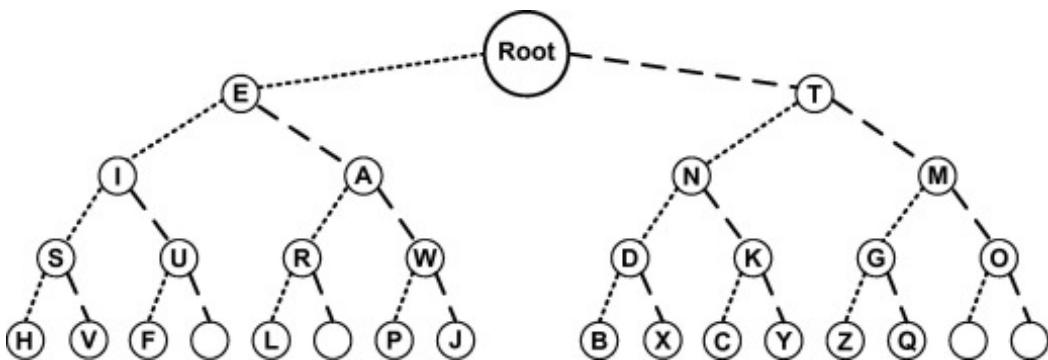
“摩斯电码”的摩尔斯电码是什么。

解决方案

从图7.1可以看出，短语MORSE CODE由以下表示：

The image shows a series of black dots connected by dashed horizontal lines. The dots are arranged in a sequence: a single dot on the far left, followed by a group of three dots connected by two horizontal dashed lines, then a single dot, another group of three dots connected by two horizontal dashed lines, and finally a single dot on the far right. This pattern repeats across the width of the image.

如果我们将它与（比如说）5位固定长度编码（实际上摩尔斯字母表中有66个符号）进行比较，后者需要45位（包括“空格”的符号），摩尔斯表示只需要23位，节省约50%。



[下载](#)：下载全尺寸图像

图7.1. 摩尔斯电码中英语字母表中的26个字母的符号代码映射。

许多人不知道，这种编码系统今天仍然被广泛使用——航空中使用的每个无线电导航信标都有摩尔斯电识别码，每个持牌[机场](#)也是如此。任何选择这种信标的飞行员在使用它导航之前都会识别它。这是为了确保它可以运行，并确保他们选择了正确的信标频率！

7.1.2。方法

存在几种方法可以利用数据统计来提供无损压缩。这些包括：

- **霍夫曼编码[1]**: 这是一种以接近一阶熵的速度编码单个符号的方法，通常与有损编解码器中的其他技术一起使用。
 - **算术编码[2], [3], [4]**: 这是一种更复杂的方法，能够实现符号的分数比特率，从而为更常见的符号提供更高的压缩效率。
 - **预测编码**: 这利用了数据相关性，而不是符号冗余，并且可以在不量化的情况下用于提供**无损图像压缩**。正如我们在**第三章**中所看到的，并将在本章中进一步研究，预测编码通常用作熵编码的预处理器。
 - **基于字典的方法**: LZ和LZW（伦佩尔-齐夫-韦尔奇）**[5][6]**等算法非常适合源数据统计未知的应用程序。它们经常用于文本和文件压缩。

以上方法经常是结合使用。例如，DC DCT系数通常使用DPCM和霍夫曼口算术编码的组合进行编码。此外，正如我们将在第8章中看到的那样，运动矢量也使用一种预测编码形式进行类似的

编码，以在熵编码之前对数据进行条件。

仅靠无损编码通常无法提供大多数现代存储或传输应用程序所需的压缩比——我们在[第一章](#)中看到，对100:1甚至200:1的压缩比的要求并不罕见。相比之下，如果孤立使用，无损压缩方法通常被限制在4:1.1左右的比例

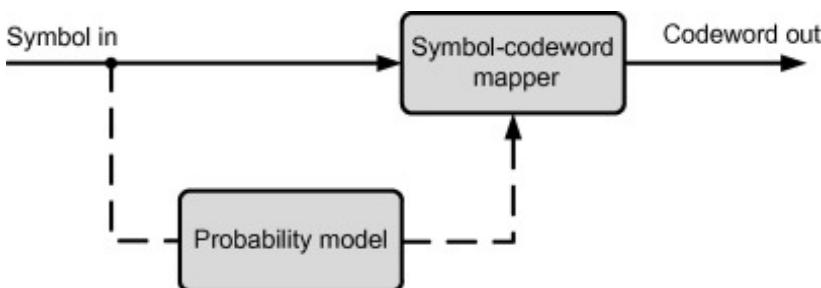
7.1.3。字典方法

一般来说，LZW（包括LZ77、LZ78）等字典方法不太适合已知或可以提前估计数据统计的图像编码应用程序。然而，它们确实在更通用的文本和文件压缩数据压缩应用程序中广泛使用，如ZIP、UNIX压缩和GIF（图形交换格式）。虽然这些方法对自然图像效果不佳，但PNG（便携式网络图形）等方法通过预处理图像以利用像素相关性来改进它们。PNG优于GIF，对于某些图像，可以与算术编码竞争。我们在这里不再进一步考虑这些技术。有兴趣的读者请参阅Sayood的书[\[7\]](#)以获得出色的介绍。

7.2。符号编码

7.2.1。无损压缩的通用模型

无损编码的通用模型如图7.2所示。这包括将源的输入符号映射到[代码词](#)的方法。为了提供压缩，码字映射器的符号必须由有代表性的概率模型来调节。概率模型可以直接从正在处理的输入数据、[先验假设](#)或两者的组合中导出。需要注意的是，压缩是无法保证的——如果概率模型不准确，那么可能不会产生压缩甚至数据膨胀。如[第三章](#)所述，代码映射器的理想符号将产生一个需要 $\log_2(1/P_i)$ 比特。



[下载：下载全尺寸图像](#)

图7.2。无损编码的通用模型。

7.2.2。熵、效率和冗余

如[第三章](#)所述， H 称为[源熵](#)，表示编码信号所需的每个符号的最小比特数。对于以平均速率（即平均码字长度）编码信号的给定编码器 T ，编码效率可以定义为：

$$E = \frac{H}{l} \quad (7.1)$$

因此，如果我们以最有效的方式为源代码编写代码，并且代码是明确的（唯一可解码的），那么代码的速率将等于其熵。

编码方案也可以以冗余为特征， R ：

$$R = \frac{\bar{l} - H}{H} \cdot 100\% \quad (7.2)$$

7.2.3。前缀代码和唯一的可解码性

我们早些时候注意到，[摩尔斯电码](#)需要空格来划分其可变长度的码字。在现代数字[压缩系统](#)中，我们不能为“空间”引入一个特殊的码字，因为它会产生开销。我们克服这种情况的方法就是使用所谓的前缀代码。这些具有确保没有码字是任何其他码字的前缀，因此（在没有错误的情况下）每个码字都是唯一可解码的。考虑四字母表的替代代码 $\{a_1, a_2, a_3, a_4\}$ 在[表7.1中](#)。

表7.1。给定概率分布的替代代码。

信件	概率	C ₁	C ₂	C ₃
a_1	0.5	0	0	0
a_2	0.25	0	1	10
a_3	0.125	1	00	110
a_4	0.125	10	11	111
平均长度		1.125	1.25	1.75

让我们检查一下这张表中的三个代码。我们可以看到，在C₁的情况下，存在模糊性 $a_2a_3 = a_1a_3$ 。同样，对于C₂， $a_1a_1 = a_3$ 。事实上，只有C₃是唯一可解码的，这是因为它是前缀代码，也就是说，没有码字是由其他码字的组合形成的。正因为如此，它是自同步的——一个非常有用的属性。问题是，我们如何为较大的字母表设计自同步代码？下一节将讨论这个问题。

示例7.2编码冗余

考虑一组具有相关概率和码字的符号，如下表所示。计算字母表的一级熵和编码的冗余。

符号	概率	码字
s_0	0.06	0110
s_1	0.23	10
s_2	0.3	00

s_3	0.15	010
s_4	0.08	111
s_5	0.06	0111
s_6	0.06	1100
s_7	0.06	1101

解决方案

1. 平均码字长度

下表重复，包括每个码字的平均长度。平均长度 \bar{l} 在这种情况下是2.71位/符号。

符号	概率	代码	平均长度
s_0	0.06	0110	0.24
s_1	0.23	10	0.46
s_2	0.3	00	0.6
s_3	0.15	010	0.45
s_4	0.08	111	0.24
s_5	0.06	0111	0.24
s_6	0.06	1100	0.24
s_7	0.06	1101	0.24
总体平均长度		2.71	

2. 一阶熵

这个字母表的第一阶熵由以下方式给出：

$$\begin{aligned}
 H &= -\sum P \log_2 P \\
 &= -(0.06 \times \log_2 0.06 + 0.23 \times \log_2 0.23 + 0.3 \\
 &\quad \times \log_2 0.3 + \cdots + 0.06 \times \log_2 0.06) \\
 &= 2.6849 \text{ bits/symbol}
 \end{aligned}$$

3. 冗余

这种编码的冗余是：

$$R = \frac{\bar{l} - H}{H} \times 100 = \frac{2.71 - 2.6849}{2.6849} \approx 1\%$$

7.3。霍夫曼编码

7.3.1。基本算法

霍夫曼编码代表每个符号 s_i 具有可变长度的二进制码字 c_i 。长度 c_i 由四舍五入决定 H_i 最多可达最近的整数。[霍夫曼代码](#)是前缀代码，即没有有效的码字是任何其他代码的前缀。因此，解码器可以在不参考之前或后续数据的情况下运行，并且不需要显式[同步信号](#)。

霍夫曼代码使用树形结构方法生成，如下所述。解码器使用类似形式的解码树解码[码字](#)——这意味着确切的树结构必须在解码之前传输，或者必须是

算法7.1霍夫曼树的形成

-
1. Create ranked list of symbols $s_0 \dots s_N$ in decreasing probability order;
 2. REPEAT
 3. Combine pair of symbols with lowest probabilities. i.e. those at bottom of list (indicated as s_i and s_j here);
 4. Update probability of new node: $P(s'_i) = P(s_i) + P(s_j)$;
 5. Sort intermediate nodes: order s'_i in the ranked list;
 6. UNTIL a single root node is formed with $P = 1$;
 7. Label the upper path of each branch node with a binary 0 and the lower path with a 1;
 8. Scan tree from root to leaves to extract Huffman codes for each symbol;
 9. END.
-

[下载：下载全尺寸图像](#)

先验地知道。对于每个符号，解码器从树根开始，并使用每个位选择分支，直到到达定义传输符号的[叶子节点](#)。

霍夫曼编码为源代码编码提供了一种有效的方法，前提是符号概率都相当小。对于子带数据的编码（例如在JPEG中），典型的效率可以在90%到95%之间实现。如果其中一个符号概率很高，则 H 会小，编解码器的效率可能会更低。霍夫曼编码要求编码器和解码器具有相同的代码词表。在某些系统中（例如JPEG）此表在数据之前传输（见图7.7）。

创建基本霍夫曼树的过程如[算法7.1](#)所述。应该清楚的是，这个过程确保结果是一个前缀代码，因为与[摩尔斯电码](#)不同，每个符号都由一个从根到叶子的图的唯一路径表示。

示例7.3基本霍夫曼编码

考虑一个产生五个符号的源 $\{s_1, s_2, s_3, s_4, s_5\}$ 在哪里

$P(s_1) = 0.2, P(s_2) = 0.4, P(s_3) = 0.2, P(s_4) = 0.15, P(s_5) = 0.05$ 。计算这个字母表的霍夫曼树。

解决方案

解决方案将分以下阶段进行描述。在第一阶段，符号按发生概率排序，列表中底部的两个符号组合成一个新节点 $s'_4(0.2)$ 其中括号中的值是两个组成节点概率之和。然后，我们确保新形成的节点列表也按降概率顺序排列如下所示：

$s_2(0.4)$ ————— $s_2(0.4)$

$s_1(0.2)$ ————— $s_1(0.2)$

$s_3(0.2)$ ————— $s_3(0.2)$

$s_4(0.15)$ ————— $s'_4(0.2)$

$s_5(0.05)$

[下载：下载全尺寸图像](#)

在第二阶段，我们在新形成的节点列表中重复这个过程，将底部的两个节点组合成一个新节点 $s'_3(0.4)$ 。这个新节点被放置在输出列表中，以确保概率值下降。这如下所示：

$s_2(0.4)$ ————— $s_2(0.4)$ ————— $s_2(0.4)$

$s_1(0.2)$ ————— $s_1(0.2)$ ————— $s'_3(0.4)$

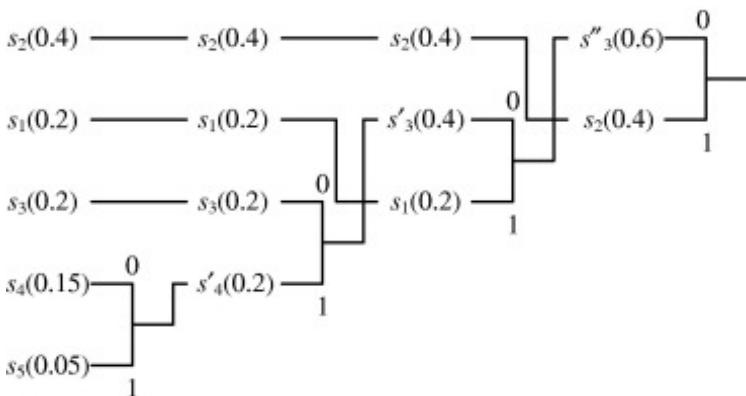
$s_3(0.2)$ ————— $s_3(0.2)$ ————— $s_1(0.2)$

$s_4(0.15)$ ————— $s'_4(0.2)$

$s_5(0.05)$

[下载：下载全尺寸图像](#)

最后，在第三阶段，我们形成了 $s''_3(0.6)$ 并将其与唯一剩余的节点组合 $s_2(0.4)$ 形成单个**根节点**，如预期的那样，合并概率为1：



[下载：下载全尺寸图像](#)

通过图表跟踪每条路径，我们可以看到这个字母表的霍夫曼码字是：

$s_1 = 01, \quad s_2 = 1, \quad s_3 = 000, \quad s_4 = 0010, \quad s_5 = 0011$

这给出了平均**码字长度**为2.2位/符号。

7.3.2。最小方差霍夫曼编码

最小方差霍夫曼编码最小化了最长码字的长度，从而最小化了传输比特率的动态变化。实现这个步骤与基本方法几乎相同，只是在形成新的节点排名列表时，将新形成的节点尽可能高地插入到列表中，而不破坏排名顺序。这确保了其重用将尽可能长时间延迟，而不破坏排名，从而最小化了通过图表的最长路径。修改后的算法在[算法7.2](#)中给出。

算法7.2 最小方差霍夫曼树的形成。

-
1. Create ranked list of symbols $s_0 \dots s_N$ in decreasing probability order;
 2. REPEAT
 3. Combine pair of symbols with lowest probabilities, i.e. those at the bottom of the list (indicated as s_i and s_j here);
 4. Update probability of new node: $P(s'_i) = P(s_i) + P(s_j)$;
 5. Sort intermediate nodes: insert s_i as high up as possible in the ranked list;
 6. UNTIL a single root node is formed with $p = 1$;
 7. Label the upper path of each branch node with a binary 0 and the lower path with a 1;
 8. Scan tree from root to leaves to extract Huffman codes for each symbol;
 9. END.
-

[下载：下载全尺寸图像](#)

示例7.4最小方差霍夫曼编码

考虑与[示例7.3](#)相同的问题，即源生成五个符号 $\{s_1, s_2, s_3, s_4, s_5\}$ 与
 $P(s_1) = 0.2, P(s_2) = 0.4, P(s_3) = 0.2, P(s_4) = 0.15, P(s_5) = 0.05$ 。计算此字母表的霍夫曼代码的最小方差集。

解决方案

在第一阶段，符号按发生概率排序，列表中底部的两个符号组合成一个新节点 $s'_4(0.2)$ 其中括号中的值是两个组成节点概率之和。我们保证新形成的节点尽可能高地插入有序列表中，如下所示：

$s_2(0.4)$ ————— $s_2(0.4)$

$s_1(0.2)$ ————— $s'_4(0.2)$

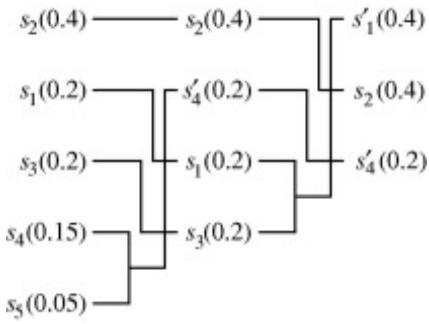
$s_3(0.2)$ ————— $s_1(0.2)$

$s_4(0.15)$ ————— $s_3(0.2)$

$s_5(0.05)$ —————

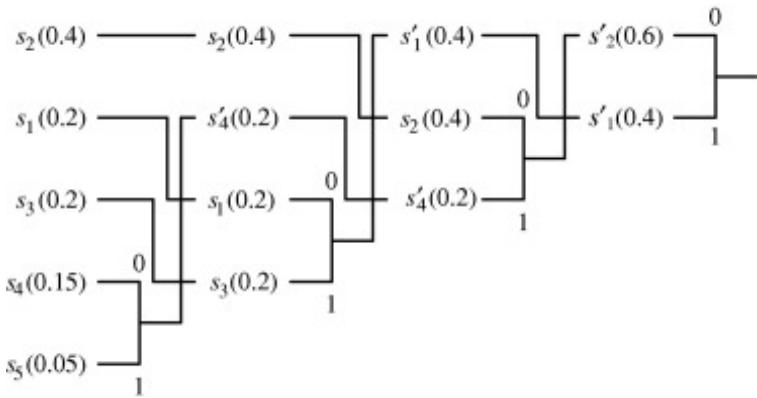
[下载：下载全尺寸图像](#)

这个过程在第二阶段以类似的方式继续：



[下载：下载全尺寸图像](#)

直到形成如下图所示的完整树：



[下载：下载全尺寸图像](#)

通过图表跟踪每条路径，我们可以看到这个字母表的霍夫曼码字是：

$$s_1 = 10, \quad s_2 = 00, \quad s_3 = 11, \quad s_4 = 010, \quad s_5 = 011$$

这再次给出了2.2位/符号的平均码字长度。然而，这一次，最小方差方法将最大码字长度减少到3位，而不是前一种情况下的4位。

7.3.3。霍夫曼解码

霍夫曼解码是一个简单的过程，它采用输入位流，从根节点开始，依次使用每个位在图的上下路径之间切换。这一直持续到到达叶子节点，这时相应的符号被解码。[算法7.3](#)描述了K位流，表示编码符号序列。为了便于理解，该算法

算法7.3 霍夫曼解码过程

-
1. INPUT bitstream representing a sequence of symbols: $b[0], b[1] \dots b[K-1]$; INPUT Huffman tree structure;
 2. $i=0$;
 3. REPEAT
 4. REPEAT
 5. Start at root node;
 6. IF $b[i]=0$ THEN select upper branch, ELSE select lower branch;
 7. $textit{i}=i+1$;
 8. UNTIL leaf node reached;
 9. OUTPUT symbol;
 10. UNTIL $i=K$;
 11. END.
-

[下载：下载全尺寸图像](#)

在解码过程中引用霍夫曼树作为数据结构。然而，应当指出，这通常使用状态机来实现。

7.3.4。修改后的霍夫曼编码

在实际情况下，霍夫曼编码的字母表可能很大。在这种情况下，代码的构建可能很费力，结果往往效率不高，特别是对于有偏见的概率分布。例如，考虑具有以下概率的8位源：

值 (x)	$P(x)$	$- (x)$	代码	比特
0	0.2	2.232	10	2
+1	0.1	3.322	110	3
-1	0.1	3.322	1110	4
+2	0.05	4.322	11110	5
-2	0.05	4.322	11111	5
所有其他值	0.5	1	0 + value	9

这意味着我们可以为前五个符号生成简短高效的代码词，并分配一个额外的1位标志作为转义代码，以指示不属于前五个符号之一的任何符号。然后，这些其他值可以用（在这种情况下）8位值进行固定长度编码。平均代码长度为2.05位。

另一种方法，有时称为扩展霍夫曼编码，也可以用于减少编码冗余，并以更接近其熵的速度对字母表进行编码。该方法在代码形成之前将符号分组在一起，而不是独立编码每个符号。正如我们在[第三章](#)中所看到的，这可以提供显著的优势。然而，有更有效的方法可以做到这一点，例如[算术编码](#)，我们将在第7.6节中查看这些方法。

7.3.5。霍夫曼码的性质

卡夫不等式

包含 N 个符号的字母表 $S = \{s_i\}$, 如果使用二进制前缀代码进行编码, 并根据它们的概率进行排名, P_i , 将有 $l_1 \leq l_2 \leq l_3 \leq \dots \leq l_N$ 。卡夫不等式指出, 这些长度必须满足以下表达式:

$$\sum_{i=1}^N 2^{-l_i} \leq 1 \quad (7.3)$$

这从每个符号的信息内容来看:

$$l_i = -\log_2 P_i = \log_2 1/P_i$$

在一并元源的情况下, 所有码字长度都是整数, 那么:

$$\bar{l} = \sum l_i P_i = H(S)$$

霍夫曼代码的长度

霍夫曼编码字母表中码字的平均长度将主要取决于字母表中概率的分布和字母表中的符号数量。我们已经在上面看到, 对于整数码字长度, 平均码字长度的下界是源熵 H 。然而, 在码字长度是非整数的情况下, 出于实际目的, 它们必须四舍五入到下一个最大的整数值。因此:

$$\bar{l} = \sum \lceil \log_2 1/P_i \rceil P_i < \sum (\log_2 (1/P_i) + 1) P_i < H(S) + 1$$

因此, 字母表 S 的霍夫曼代码具有平均长度 \bar{l} 受以下限制:

$$H(S) \leq \bar{l} \leq H(S) + 1 \quad (7.4)$$

霍夫曼码的最优化

为了使代码达到最佳状态, 它必须满足以下条件[7], [8]:

1. 对于任何两个符号 s_i 和 s_j 在字母表中 S , 如果 $P(s_i) > P(s_j)$ 然后 $l_i \leq l_j$ 其中 l 表示码字的长度。
2. 概率最低的两个符号的长度相等和最大。
3. 在与最佳代码对应的树中, 每个中间节点必须有两个分支。
4. 如果我们将连接到单个中间节点的几个叶子节点折叠成一个复合叶子节点, 那么树将保持最佳状态。

可以清楚地看到, 霍夫曼代码满足了这些条件, 并代表了单个符号编码的最佳表示形式。然而, 我们稍后会看到, 通过编码符号序列而不是单个符号可以提高效率。

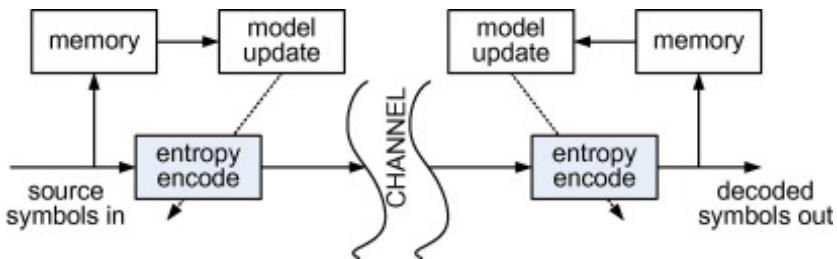
7.3.6。霍夫曼编码的自适应方法

霍夫曼编码有两个基本限制:

1. 它一次编码一个符号。
2. 它取决于预计算的编码和解码结构。

正如我们稍后将看到的, 这两个限制都通过算术编码来解决。然而, 自适应霍夫曼编码方法也已

设计出来，可以根据流行的源统计动态更新霍夫曼树。方法包括加拉格尔[9]和最近维特[10]的方法。与任何自适应系统一样，基本要素是编码器和解码器模型使用相同的方法以同情方式初始化和更新。这种方法的基础见图7.3。



[下载：下载全尺寸图像](#)

图7.3。自适应熵编码。

在简单的情况下，编码器和解码器都会用单个节点初始化它们的树，并且都知道使用的符号集。更新过程为每个叶子节点分配一个权重W，该权重W等于该符号遇到的次数。符号最初使用一组预先确定的基本代码传输，随着出现频率的变化，树会按照商定的程序进行更新。

算法7.4说明了该过程的一个示例。我们首先计算表示所有 N 个符号所需的节点数量 M （即 $2N - 1$ ）。我们用一个数字标记每个节点 $1 \dots M$ 并根据对应符号出现的频率追加当前W值（中间节点的赋值等于其兄弟之和）。我们定义了一个Escape节点，最初是标签为 N 和权重 $W = 0$ 的根节点。当第一个符号 s_i 遇到时，它使用字母表的预先商定的初始编码进行编码。然后，ESC节点创建一个具有两个兄弟姐妹的新根节点（标签 N ）： s_i 标记为 $N-1$ ，一个新的ESC节点标记为 $N-2$ 。新代码词 s_i 现在是1，ESC是0。因此，当遇到新符号时，它必须编码为ESC（当前为0），然后是预先商定的代码。这个过程仍在继续，随着遇到新符号，会产生新的节点。

随着节点的频率计数增加，树需要重组以解释这一点。这是通过将权重为 W 的节点与权重为 $W-1$ 的最高编号节点交换，然后重新排列节点以提供

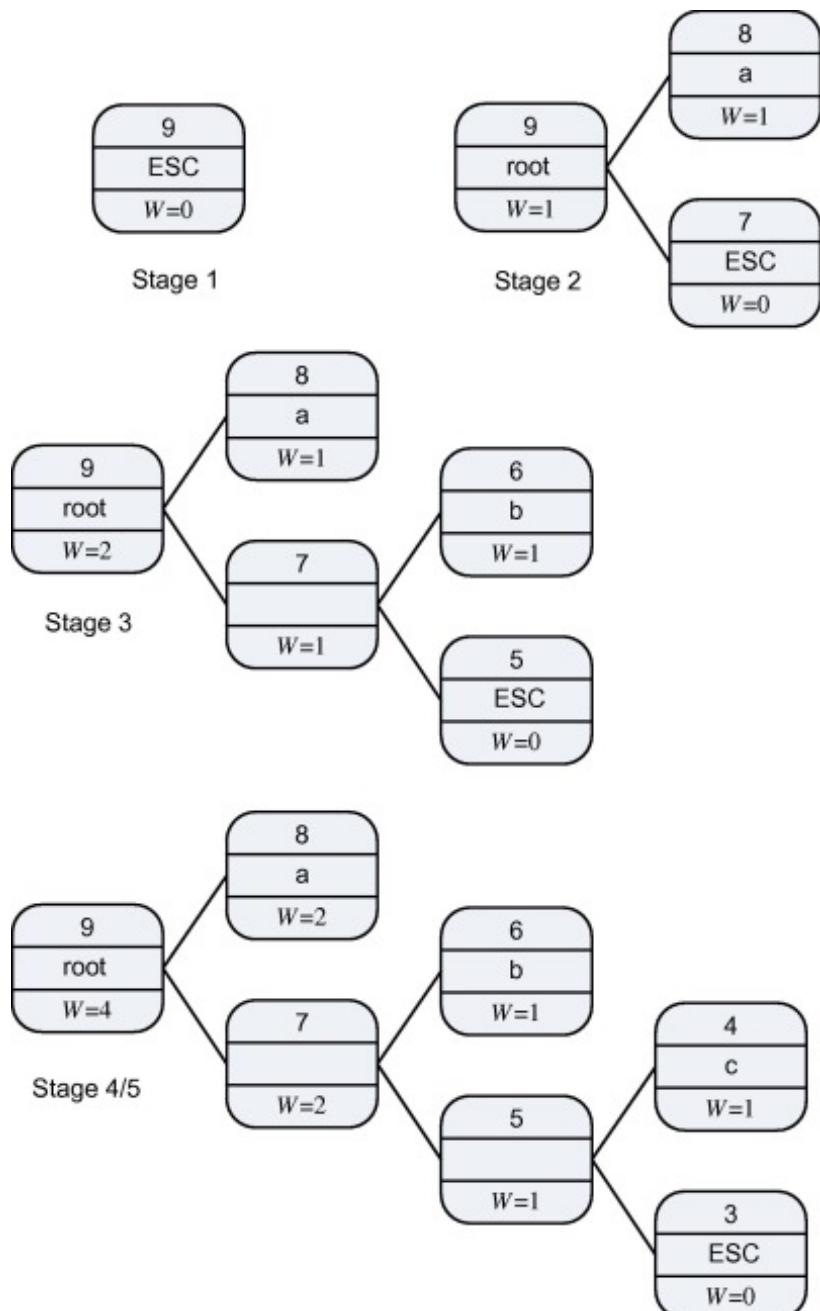
算法7.4自适应霍夫曼编码：树更新

1. INPUT symbol s_i ;
2. If symbol s_i is new then create new node with two siblings: one for s_i and a new ESC node;
3. Update weight for node s_i : $W = W + 1$;
4. Swap node with weight W with the highest numbered node with weight $W - 1$ and then rearrange nodes to provide a minimum variance structure.

[下载：下载全尺寸图像](#)

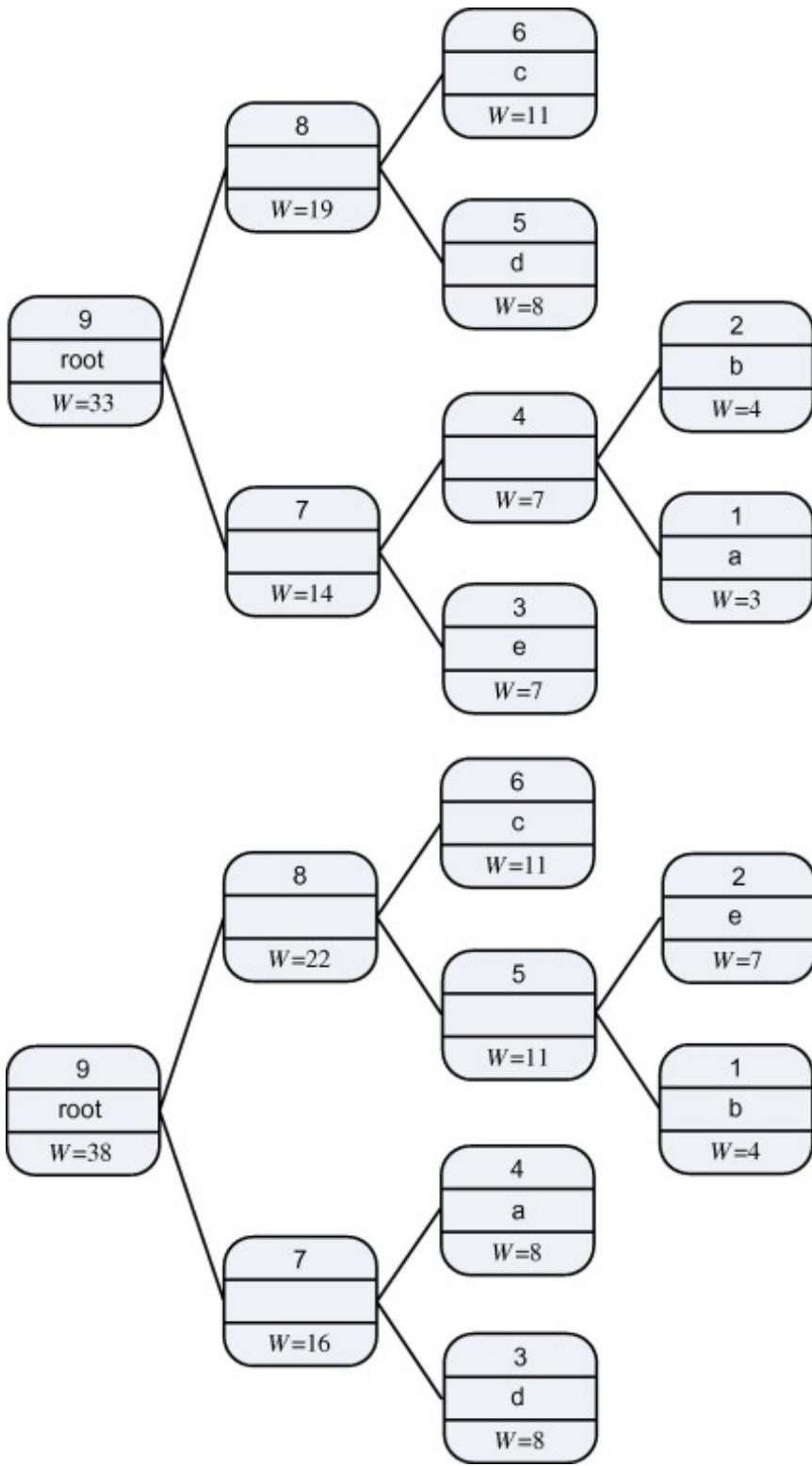
最小方差结构。这个过程能够不断适应符号统计的动态变化，提供最佳代码词。

图7.4提供了由五个符号 $\{a, b, c, d, e\}$ 组成的字母表的树演化过程的简单示例，其传递序列 $\{a, b, c, a\}$ 。**图7.5**说明了在符号 a 的权重增加五个计数后，节点交换用于更新树的情况。



[下载：下载全尺寸图像](#)

图7.4。自适应霍夫曼编码树演化示例。



[下载：下载全尺寸图像](#)

图7.5。自适应霍夫曼编码示例，显示权重更新符号[a](#)后节点交换（底部）五次计数。

7.4. 符号形成和编码

7.4.1。处理稀疏矩阵

许多数据源包含一个特定符号的长字符串。例如，由变换或基于小波的压缩系统生成的量子子带数据通常很稀疏。我们在第五章中看到，在应用正向变换和量化后，得到的矩阵包含相对较小比例的非零条目，其大部分能量压实到较低的频率（即矩阵的左上角）。在这种情况下，运行长度编码（RLC）可以通过将相同值分组到一个符号中来有效地表示相同值的长字符串，该符号对值和重复次数进行编码。这是一个简单有效的减少序列冗余的方法。对于稀疏矩阵来说，一个很小的变体是将零与下一个非零条目的值相结合来编码运行。

为了执行运行长度编码，我们需要将二维系数矩阵转换为一维向量，此外，我们希望以最大化零运行的方式做到这一点。例如，考虑 6×6 数据块及其变换系数，图7.6。如果我们使用之字形模式扫描矩阵，如图所示，那么这比按行或列扫描更节能。

示例7.5稀疏能量-紧凑矩阵的运行长度编码

考虑 4×4 下面量化了DCT矩阵。使用{run/value}符号执行之字形扫描，以生成紧凑的一维矢量和运行长度代码。

$$\mathbf{C}_Q = \begin{bmatrix} 8 & 5 & 2 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

解决方案

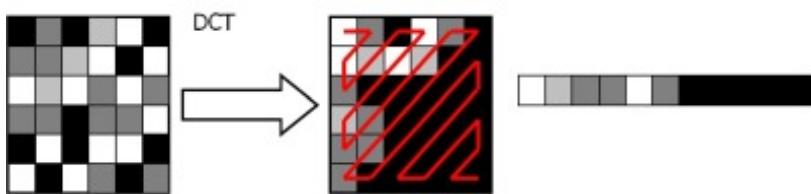
执行之字形扫描以生成一维矢量：

$$\mathbf{c}_Q = [8 \ 5 \ 3 \ 0 \ 0 \ 2 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

接下来用{run/value}符号表示这个向量：

$$\mathbf{s}_Q = [(0/8) \ (0/5) \ (0/3) \ (2/2) \ (3/1) \ \text{EOB}]$$

请注意，为了避免编码最后一次运行的零，用块结束（EOB）符号替换这些是正常的。



[下载：下载全尺寸图像](#)

图7.6。在可变长度编码之前进行之字形扫描。

7.4.2。JPEG中的符号编码

JPEG仍然是普遍采用的静态图像编码标准。它于1992年首次推出，此后几乎没有变化[11]。基线JPEG编码器的操作如图7.7所示。首先输入图像，通常在4:2:0YC_bC_r格式，被分割成非重叠 8×8

块，这些块与装饰相关，使用 8×8 DCT和量化，如第5章所述。这些量化系数的熵编码通常基于霍夫曼编码²，并按以下方式进行：

- 直流系数相对于前一个区块的相应值进行脉冲编码调制（DPCM）差异。然后，霍夫曼使用一组专门为直流系数设计的表格对预测残差的大小类别（如表7.4所示）进行编码。然后，预测残差的振幅以一个补码的形式附加到这个码字上。
- 如上所述，交流系数首先是之字形扫描，然后进行运行长度编码。每个非零交流系数使用的运行长度符号是(*run / size*)，其中运行是下一个非零系数之前的零数，大小与该系数的值有关，如表7.4所示。然后，这些符号使用为交流系数设计的一组霍夫曼表进行熵编码，系数值再次以一个补体形式附加到码字上。

算法7.5、算法7.6更详细地描述了这些过程。图7.8中的图表说明了这种方法，该图表显示了尺寸值的分布

算法7.5 JPEG直流系数符号编码

-
1. Form DPCM residual for the DC coefficient in the current block DC_i , predicted relative to the DC coefficient from the previously encoded block, DC_{i-1} : $DC_{i(pred)} = DC_i - DC_{i-1}$;
 2. Produce $symbol_{DC} = (size)$ for the DC coefficient prediction;
 3. Huffman encode $symbol_{DC}$ using the Table for DC coefficient differences;
 4. OUTPUT bits to JPEG file or bitstream;
 5. OUTPUT *amplitude* value of the coefficient difference in ones complement format to the file or bitstream;
 6. END.
-

[下载：下载全尺寸图像](#)

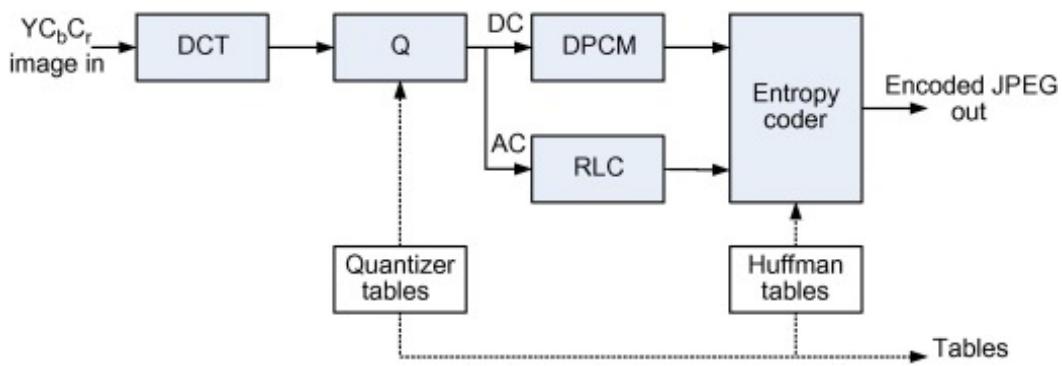
算法7.6 JPEG交流系数符号编码

-
1. Form a vector of zig-zag scanned AC coefficient values from the transformed data block as described in Figure 7.6;
 2. $i=0$;
 3. REPEAT
 4. $i = i + 1$;
 5. Produce $symbol_{AC,i} = (run/size)$ for the next non-zero AC coefficient in the scan;
 6. Huffman encode $symbol_{AC,i}$ using Table for AC coefficients;
 7. OUTPUT codeword bits to the JPEG file or bitstream;
 8. OUTPUT *amplitude* value of the non-zero coefficient in ones complement form to the JPEG file or bitstream;
 9. UNTIL $symbol_{AC,i-1} = (0/0)$;
 10. END.
-

[下载：下载全尺寸图像](#)

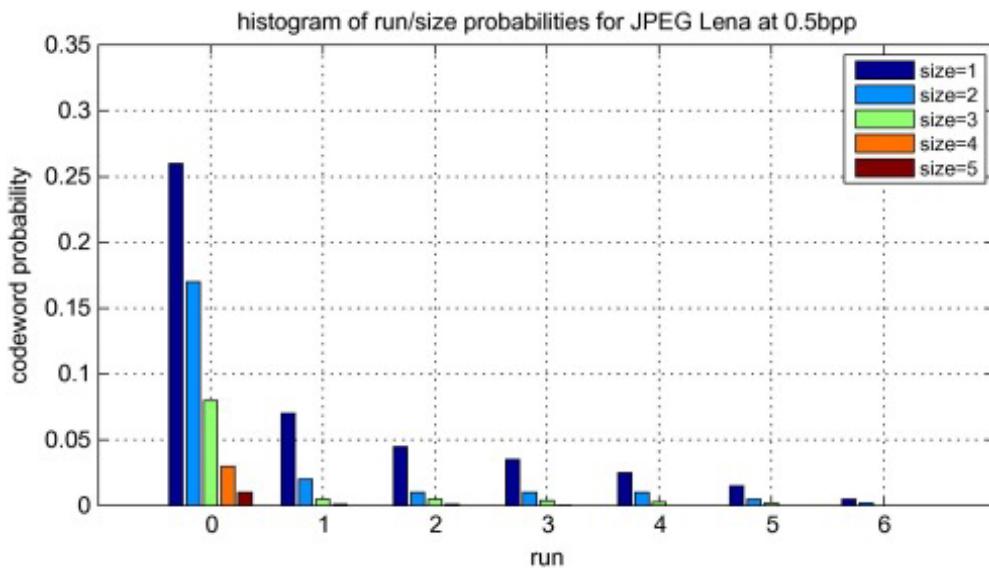
对于JPEG编码的Lena映像的不同运行长度。该图显示了0.5 bpp的统计数据，但其他压缩比也获得了类似的特征。可以看出，随着运行长度的增加和尺寸的缩小，概率会迅速下降。因此，在这

种情况下，较长的码字是合理的，为较低的运行值和大小保留较短的码字。



[下载：下载全尺寸图像](#)

图7.7。JPEG基线图像编码器架构。



[下载：下载全尺寸图像](#)

图7.8。在典型的JPEG编码中分配运行/大小值。

JPEG实际上将分配运行的比特数和大小限制为每位4位，两者的范围为0-15。如果运行大于15个零（这不太可能），则将其分为15个倍数加上剩余长度。这有效地限制了熵代码表的大小。因此，在JPEG中，运行/大小符号实际上由：RRRSSSSS表示。因此，大小为0的唯一运行/大小符号是EOB (0/0) 和ZRL (15/0)。

直流和交流亮度系数的默认JPEG霍夫曼代码的子集见表7.2，表7.3。可以从参考文献中获取完整的集合。[\[11\]](#)。

示例7.6JPEG系数编码

考虑交流亮度系数值-3，前面有五个零。这个运行和系数值的JPEG符号是什么？使用默认的JPEG霍夫曼表，计算此系数和由此产生的输出**比特流**的霍夫曼代码。

解决方案

从表7.4可以看出，尺寸 $(-3) = 2$ 。因此，符号是 $s = (5/2)$ 。

从默认的JPEG赫夫曼亮度交流系数表中， $(5/2)$ 的霍夫曼代码是11111110111。非零系数的值为-3，在一补格式中为00。

因此，为该序列生成的输出比特流是：

1111111011100

示例7.7 JPEG矩阵编码

考虑以下量化亮度DCT矩阵。假设前面一个块的直流系数值是40，那么这个矩阵的JPEG符号序列是什么？使用默认的JPEG霍夫曼表，计算其霍夫曼代码和结果的输出比特流。假设原始像素是8位值，那么这个编码的压缩比是多少？

$$\mathbf{C}_Q = \begin{bmatrix} 42 & 16 & 0 & 0 & 0 & 0 & 0 & 0 \\ -21 & -15 & 0 & 0 & 0 & 0 & 0 & 0 \\ 10 & 3 & -3 & 0 & 0 & 0 & 0 & 0 \\ -2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

解决方案

1. 直流系数

当前直流系数的预测剩余值为 $\text{DC}_{i(\text{pred})} = \text{DC}_i - \text{DC}_{i-1} = 2$ 。因此，尺寸=2，振幅=2。

表7.2中的霍夫曼码字是011。因此，在这种情况下，直流系数的比特流是01110。

2. 交流系数

上面矩阵的之字形扫描向量是：

$$\mathbf{c}_Q = [16 \ -21 \ 10 \ -15 \ 0 \ 0 \ 0 \ 3 \ -2 \ 0 \ \dots \\ \dots \ 2 \ -3 \ 0 \ 0 \ 0 \ 0 \ 2 \ -1 \ 0 \ \dots]$$

对于这些非零系数中的每个系数，我们现在可以计算（运行/大小）符号、其霍夫曼代码（来自表7.3）和相应的系数互补振幅表示。下表概述了这些。

3. 输出比特流

该矩阵的输出码流是直流和交流码的串联，如下所示：

(0111011010100001101001010101110101011000011111
 ...
 ... 0111110101110111001001111110111100001010)

4. 压缩比

忽略任何头或边信息，此块的位总数为 $5 + 82 = 87$ bits。相比之下，原始矩阵有64个条目，每个条目为8位。因此，它需要512位，压缩比为5.9:1或1.36 bpp。

系数	运行/大小	霍夫曼代码	振幅	#bits
16	0/5	11010	10000	10
-21	0/5	11010	01010	10
10	0/4	1011	1010	8
-15	0/4	1011	0000	8
3	3/2	111110111	11	11
-2	0/2	01	01	4
2	1/2	11011	10	7
-3	0/2	01	00	4
2	5/2	1111110111	10	13
-1	0/1	00	0	3
EOB	0/0	1010	-	4
交流电位总数				82

表7.2。亮度直流系数差异表。

类别	代码长度	码字
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110

6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

表7.3。亮度交流系数表，显示运行/大小（R/S）值、霍夫曼代码及其长度（L）。

R/S	L	码字
0/0	4	1010 (EOB)
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	11111111110000010
0/A	16	11111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	11111111110000100
1/7	16	11111110000101
1/8	16	11111110000110

1/9	16	11111110000111
1/A	16	11111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	11111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	11111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	11111111110010001
3/7	16	11111111110010010
3/8	16	11111111110011
3/9	16	11111111110010100
3/A	16	11111111110010101

表7.4。JPEG系数大小类别。

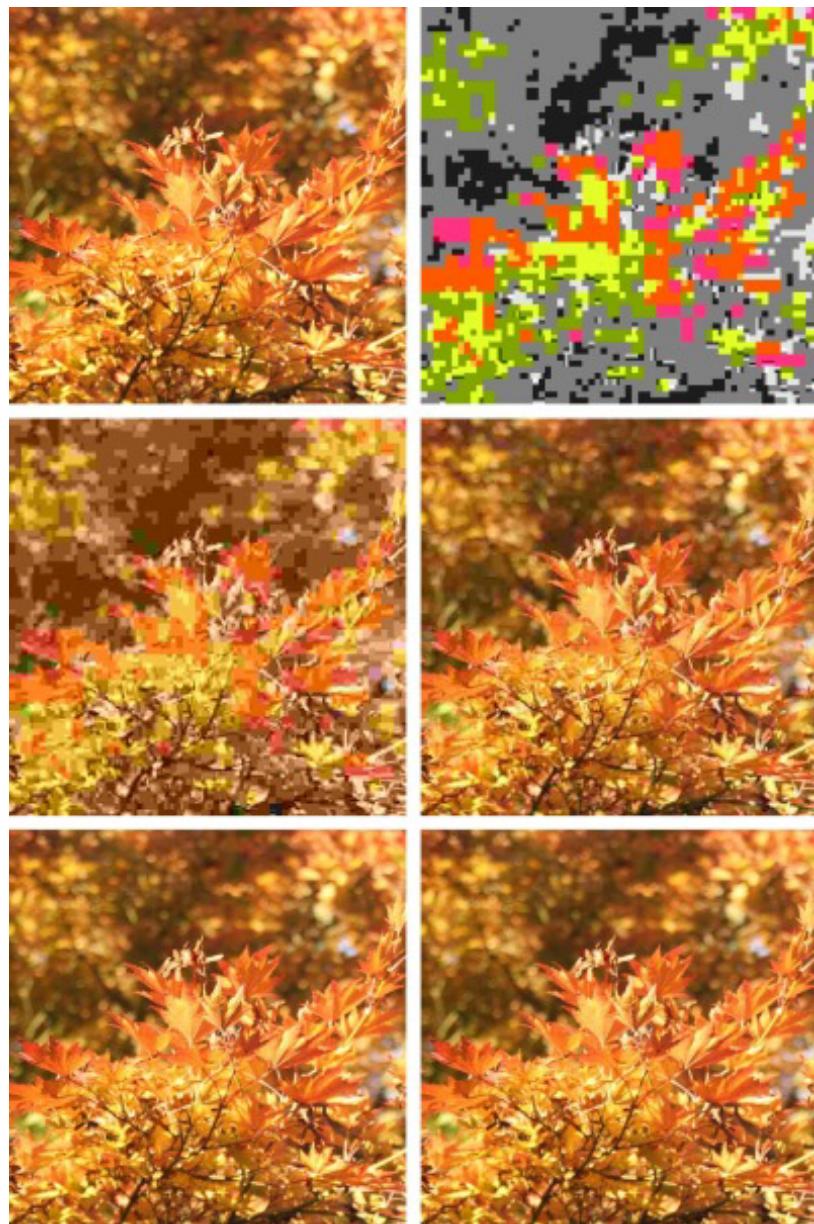
类别	振幅范围
0	0
1	-1, +1
2	-3, -2, +2, +3

3	$-7 \cdots -4, +4 \cdots +7$
4	$-15 \cdots -8, +8 \cdots +15$
5	$-31 \cdots -16, +16 \cdots +31$
6	$-63 \cdots -32, +32 \cdots +63$

等等。等等。

7.4.3。JPEG性能示例

一些演示JPEG编码性能的例子如图7.9所示，图7.10用于高度纹理的图像（枫树）和结构更清晰的图像（木偶）。两个原始图像的字长为8位。可以观察到，结果在0.25 bpp以上开始在心理视觉上可以接受，并在1bpp时变得良好。主观上，由于高频纹理的贡献较低，在压缩过程中量化得更重，木偶的结果在0.22 bpp时比枫树更容易接受。



[下载：下载全尺寸图像](#)

图7.9。JPEG结果用于各种压缩比 512×512 枫树图像。左上角：原始。右上角：0.16 bpp。左中角：0.22 bpp。右中：0.5 bpp。左下角：1 bpp。右下角：2 bpp。



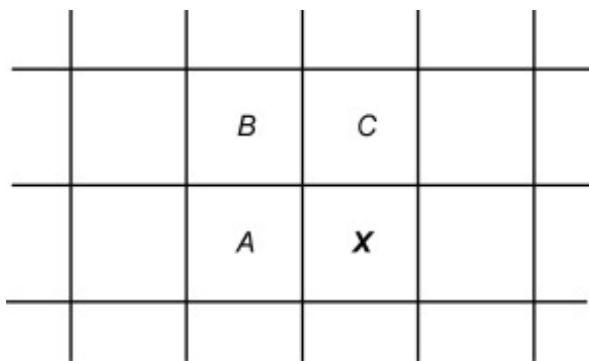
[下载：下载全尺寸图像](#)

图7.10。JPEG结果用于各种压缩比 512×512 木偶形象。左上角：原始。右上角：0.16 bpp。左中角：0.22 bpp。右中：0.5 bpp。左下角：1 bpp。右下角：2 bpp。

7.4.4。无损JPEG模式

JPEG标准定义了一个简单的[无损模式](#)，它将预测编码与霍夫曼或算术编码相结合。预测框架如[图7.11所示](#)，预测器的选择如[表7.5所示](#)。预测器模式从表中选择，并在标头中作为侧面信息发出信号，并且在整个扫描过程中通常保持不变。预测后，残差是霍夫曼或算术编码的。在实践中，

重建的像素值，而不是实际的像素值，用于预测，以消除编码器和解码器之间的数值不匹配。



[下载：下载全尺寸图像](#)

图7.11。无损JPEG的预测样本。

表7.5。无损JPEG的预测模式。

模式	预测器
0	没有预测
1	$\widehat{X} = A$
2	$\widehat{X} = B$
3	$\widehat{X} = C$
4	$\widehat{X} = A + B - C$
5	$\widehat{X} = A + (B - C)/2$
6	$\widehat{X} = B + (A - C)/2$
7	$\widehat{X} = (A + B)/2$

7.4.5。H.264/AVC中的上下文自适应可变长度编码（CAVLC）

在基线和扩展配置文件中，H.264/AVC使用两种熵编码方法。第一个用于除转换系数外的所有语法元素，基于所有语法元素（SE）的单个码字表。SE到该表的映射是根据数据统计信息进行调整的。使用Exp-Golomb代码（见第7.5节），该代码在提供简单解码过程的同时有效。

在变换系数的情况下，使用上下文自适应可变长度编码（CAVLC）。这提供了更好的性能，并支持一些与SE相关的表，根据流行的数据统计信息，这些表可以在两者之间切换。虽然CAVLC不是基于霍夫曼编码的，但它在这里值得一提，因为它显示了上下文切换的力量。参考文献中给出

了CAVLC编码的好例子。[\[12\]](#)。

CAVLC利用了我们之前看到的变换矩阵的稀疏性，但也利用了扫描系数向量的其他特征，最明显的是：(i) 扫描通常以一系列 ± 1 值和(ii) 相邻块中的系数数是相关的。该方法根据本地上下文使用不同的VLC表。

CAVLC分别编码非零系数的数量以及这些系数的大小和位置。根据数据的特点，可以调用多种不同的编码模式。例如，CAVLC可以编码：

- **非零系数和尾随系数 (T1)** 的数量：T1是值为 ± 1 在扫描结束时（扫描通常以 ± 1 ）。根据相邻块中的系数数量，使用四个VLC表来做到这一点。
- **非零系数的值**：T1可以表示为符号；其他系数按反向扫描顺序编码，因为扫描结束时的可变性较低。根据最近编码系数的大小，使用六个Exp-Golomb代码表进行自适应。
- **TotalZeroes和RunBefore**：这些指定了扫描开始到最后一次非零系数之间的零数。RunBefore指示这些零点是如何分布的。

理查森[\[13\]](#)提供了一个CAVLC编码的好例子。

7.5。戈伦布编码

戈伦姆家族包括乌纳里、戈伦、戈伦-赖斯和指数戈伦姆代码[\[14\]](#)、[\[15\]](#)。这些是可变长度的代码，通过索引和码字之间的固定映射生成。与霍夫曼代码一样，较短的码字可以分配给概率较高的符号，反之亦然。Golomb代码的优点是，它们可以简单地通过算法编码和解码，而无需查找表。它们还具有这样的性质，即对于任何几何分布，都可以找到最佳前缀代码的编码。因此，自适应方法在JPEG-JPEG-LS的无损形式等标准中得到了开发和利用。

7.5.1。一元代码

最简单类型的一元代码将非负整数*i*用*i* 0s和单个1（或相反）编码。前几个一元代码如表7.6所示，其中假设索引值对应于按概率递减的字母符号。这种编码通常效率不高，除非概率是2的连续幂。

表7.6。索引值的 Golomb 代码*i* = 0 … 9。

我	一元	戈隆-赖斯($m = 4$)	吉隆
0	1	1 00	1
1	01	1 01	01 0
2	001	1 10	01 1
3	0001	1 11	001 00

4	00001	01 00	001 01
5	000001	01 01	001 10
6	0000001	01 10	001 11
8	00000001	01 11	0001 000
9	000000001	001 00	0001 001
:	:	:	:

7.5.2。Golomb和Golomb-Rice代码

Golomb代码将所有索引值*i*划分为大小为*m*的等大小的组。然后，代码词由描述每个组的一元代码构建，然后是固定长度代码*v_i*指定已编码的索引的其余部分。Golomb代码残差可以使用方程(7.5)简单生成，其中*v_i*表示索引*i*的余数：

$$v_i = i - \lfloor \frac{i}{m} \rfloor m \quad (7.5)$$

Golomb-Rice代码是一个特殊情况*m* = 2^k并如表7.6所示。可以观察，戈伦-赖斯码比一元码增长得慢，因此通常效率更高。

7.5.3。指数哥伦码

这些是由Teuhola[16]提出的，Golomb-Rice代码将字母表划分为大小相等的组，而Exp-Golomb代码将其划分为指数级增长的组大小。Golomb-Rice和Exp-Golomb代码的比较性能取决于字母表的大小和*m*的选择，但对于较大的字母表，Exp-Golomb编码具有显著优势。

符号索引*i*的Exp-Golomb代码包括一元代码 (ζ_i 零，后跟一个1)，与固定长度的代码连接，*v_i*。可以简单生成如下：

$$\begin{cases} v_i = 1 + i - 2^{\zeta_i} \\ \zeta_i = \lfloor \log_2(i+1) \rfloor \end{cases} \quad (7.6)$$

Exp-Golomb编码已用于H.264/AVC和HEVC。

示例7.8 Exp-Golomb编码和解码

(i) 符号索引22810的Exp-Golomb代码是什么？

(ii) 显示Exp-Golomb码字0000100111将如何解码并计算相应符号索引的值。

解决方案

(i) 这种情况下的码字组由以下方式给出：

$$\zeta_i = \lfloor \log_2(i+1) \rfloor = 7$$

剩余部分由以下方式给出：

$$\nu_i = 1 + i - 2^{\zeta_i} = 101_{10} = 1100101_2$$

因此，完整的码字是00000011100101。

- (ii) 该算法很简单，对于每个新码字，我们计算前导零的数量 (ζ_i)，忽略下面的1，然后从下一个解码索引 $i\zeta_i$ 比特。在这种情况下，有五个前导零和编码的剩余值 $\nu_i = 00111_2 = 7_{10}$ 。然后，可以从方程 (7.6) 计算出指数，作为 $i = \nu_i - 1 + 2^{\zeta_i} = 134$ 。

7.6。算术编码

尽管霍夫曼编码仍然广泛应用于JPEG等标准，但近年来，一种替代的、更灵活的方法越来越受欢迎。在大多数视频编码标准中，[算术编码](#)的变体被用作首选熵编码机制，基于上下文的自适应二进制算术编码 (CABAC) 现已成为H.264/AVC和HEVC的基础[17]。算术编码基于使用累积概率分布作为符号序列代码生成的基础。香农[18]、艾布拉姆森在他的信息理论书[2]和杰利内克[19]中提到了这种方法的一些最早。这项技术最近更普及归功于Rissanen[3]和其他人。优秀的概述可以在Refs中找到。[\[4\]](#), [\[7\]](#), [\[8\]](#)。

7.6.1。基本的算术编码算法

算术编码器将输入符号串表示为二进制分数。在算术编码中，根据所编码符号的概率对半开编码范围[0,1)进行细分。编码的第一个符号将属于给定的细分；该细分形成一个新的区间，然后以相同的方式细分以编码第二个符号。这个过程会迭代，直到序列中的所有符号都经过编码。

如果已知一些基本的附加信息（即传输的符号数量或EOB符号的存在），传输的完整符号序列因此可以在最终迭代时由最终符号范围内的任何分数数（称为标记）唯一定义。

通过使用二进制符号并将概率近似为2的幂，只需移动和添加操作即可实现算术编码器。这实现了与霍夫曼编码相当甚至更好的计算复杂性，同时提供卓越的压缩性能。[算法7.7](#)说明了基本的算术编码算法。

我们在[第三章](#)中看到，源的平均码字长度在很大程度上取决于符号的创建和编码方式，通过将符号分组组合，我们可能实现更好的性能。简单地说，这就是算术编码的作用，但它以一种相当优雅和灵活的方式做到了。

让我们假设我们有一组 M 符号组成我们的[源字母表](#)， $S = \{s_0, s_1, s_2, \dots, s_{M-1}\}$ ，其中每个符号映射到一个整数 x 在射程内 $\{0, 1, 2, 3, \dots, M-1\}$ 对应于其符号索引。因此，源符号概率被定义为：

$$P(m) = P(x = m); \quad m = 0 \cdots M-1$$

累积概率分布现在可以根据这些概率定义如下：

$$P_x(m) = \sum_{k=0}^m P(k); \quad m = 0 \cdots M-1 \tag{7.7}$$

算法 7.7 算术编码

-
1. Subdivide the half open interval $[0, 1)$ according to the symbol cumulative probabilities: [equation \(7.7\)](#);
 2. Initialize upper and lower limits: $l(0)=0; u(0)=1$;
 3. INPUT sequence of symbols $\{x_i\}$;
 4. $i=0$;
 5. REPEAT
 6. $i=i+1$;
 7. Compute lower interval limit: $l(i) = l(i-1) + (u(i-1) - l(i-1)) P_x(x_{i-1})$;
 8. Compute upper interval limit: $u(i) = l(i-1) + (u(i-1) - l(i-1)) P_x(x_i)$;
 9. UNTIL $i=N$;
 10. OUTPUT arithmetic codeword, $v \in [l(N), u(N)]$;
 11. END.
-

[下载：下载全尺寸图像](#)

在[第三章](#)和[第7.3.5节](#)讨论代码[优化](#)后，我们可以看到，一个优化编码系统将为S中的每个符号编码，平均长度为 $-\log_2 P_i$ 比特。

因此，现在我们可以继续开发迭代符号编码过程。首先，我们定义一个数字线，为区间内的半开区间 $0 \dots 1$ ，即 $[0, 1)$ 。然后，我们可以根据每个迭代的下限和上限（或等价地，其长度）来定义每个累积概率区间。所以，如果我们有一个N个输入符号序列 $X = \{x_i\}; i = 1 \dots N$ ，我们定义 $l(i)$ 是编码和 $u(i)$ 是相应的上限，然后在编码开始时：

$$\begin{aligned} l(0) &= P(-1) = 0 \\ u(0) &= P_x(M-1) = 1 \end{aligned}$$

第一个符号编码后， $i = 1$ ：

$$\begin{aligned} l(1) &= P_x(x_1 - 1) \\ u(1) &= P_x(x_1) \end{aligned}$$

同样，在第二个符号 $i = 2$ 之后，我们有：

$$\begin{aligned} l(2) &= l(1) + (u(1) - l(1)) P_x(x_2 - 1) \\ u(2) &= l(1) + (u(1) - l(1)) P_x(x_2) \end{aligned}$$

一般来说，在 i th符号， x_i ，我们有：

$$\begin{aligned} l(i) &= l(i-1) + (u(i-1) - l(i-1)) P_x(x_{i-1} - 1) \\ u(i) &= l(i-1) + (u(i-1) - l(i-1)) P_x(x_i) \end{aligned} \tag{7.8}$$

这最好用一个简单的例子（[例如7.9](#)）来说明。

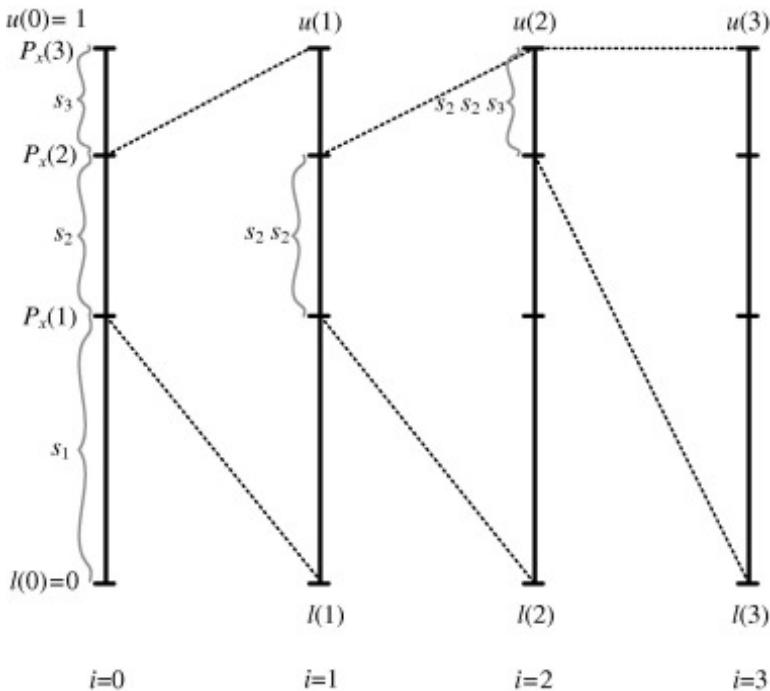
示例7.9算术编码

考虑一系列符号 $\{s_2, s_2, s_3\}$ 来自字母表 $\{s_1, s_2, s_3\}$ 具有概率 $\{P(1), P(2), P(3)\}$ 。显示与此序列对应的三个迭代的算术编码过程。

解决方案

使用我们上面的符号，序列映射到 $X = \{x_0, x_1, x_2\} \Rightarrow \{2, 2, 3\}$ 。该序列的算术编码器的三个迭代如

下所示：



[下载：下载全尺寸图像](#)

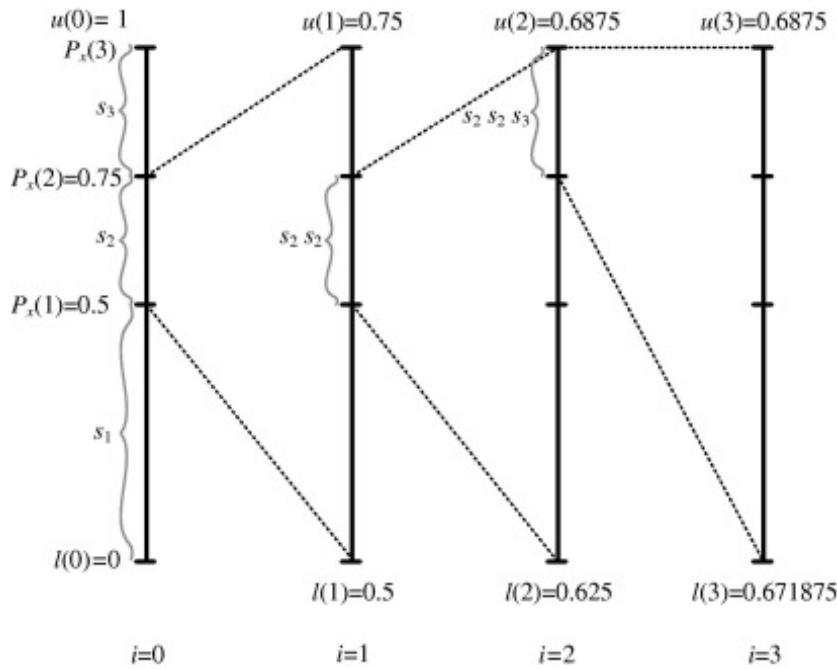
我们可以看到 $[l(3), u(3)]$ 范围内的任何值都唯一地定义了这个序列，当然，我们知道何时停止解码——但稍后会更多。

示例7.10算术编码-数字示例

使用与[示例7.9](#)相同的符号和序列，假设符号概率为： $P(1) = 0.5, P(2) = 0.25, P(3) = 0.25$ 。计算序列的[算法代码](#) $\{s_2, s_2, s_3\}$ 。

解决方案

该序列的算术编码器的三个迭代如下所示，包括每个阶段的标记值：



[下载：下载全尺寸图像](#)

作为计算临时间隔限制的示例，考虑 $i = 2$ ($i = 1$ 可以通过检查计算)：

$$\begin{aligned} l(2) &= l(1) + (u(1) - l(1))P_x(1) \\ l(2) &= 0.5 + (0.75 - 0.5)0.5 = 0.625 \\ u(2) &= l(1) + (u(1) - l(1))P_x(2) \\ u(2) &= 0.5 + (0.75 - 0.25)0.75 = 0.6875 \end{aligned}$$

在迭代3中重复此操作会导致一个可以假定区间内任何值的算术代码

$v \in [0.671875, 0.6875] = [0.101011_2, 0.1011_2]$ 。为了简单起见，我们可能会选择下限，或者在下限和上限之间取值。在实践中，最好使用最短的二进制表示形式，在这种情况下也是下限，即0.101011，或者，如果我们删除前导零，因为它是多余的， $\hat{v} = 101011$ 。

7.6.2。基本的算术解码算法

第7.6.1节中描述的编码算法创建一个算术码字， $\hat{v} \in [l(N), u(N)]$ 。解码器的目的是生成相同的输出序列， $\hat{x}_i = \{x_1, x_2, \dots, x_N\}$ 就像在编码器上处理的那样，在那里 $\{x_i = k\} \leftarrow \{s_k\}$ 。我们通过递归更新半开放区间的上限和下限来做到这一点。初始化这些：

$$\begin{aligned} l(0) &= P_x(0) = 0 \\ u(0) &= P_x(M-1) = 1 \end{aligned}$$

然后我们计算符号索引的序列 \hat{x}_i 使用：

$$\hat{x}_1 = \{x : P_x(x_1 - 1) \leq \hat{v} < P_x(x_1)\} \quad (7.9)$$

因此，一般来说，我们需要找到下一个符号索引，以便：

$$\hat{x}_i = \{x : l(i) \leq \hat{v} < u(i)\}; \quad i = 1, 2, \dots, N \quad (7.10)$$

其中，对于 i 的每个值，我们需要更新下限和上限：

$$\begin{aligned} u(i) &= l(i-1) + (u(i-1) - l(i-1))P_x(x_i) \\ l(i) &= l(i-1) + (u(i-1) - l(i-1))P_x(x_i - 1) \end{aligned} \quad (7.11)$$

这在**算法7.8**中形式化。

例 7.11 算术解码——数字例

考虑从字母表中解码表示三个符号序列的算术代码 $0.538S = \{s_1, s_2, s_3, s_4\}$ 在哪里 $P(s_1) = 0.6, P(s_2) = 0.2, P(s_3) = 0.1, P(s_4) = 0.1$ 。这个算术代码所表示的符号序列是什么？

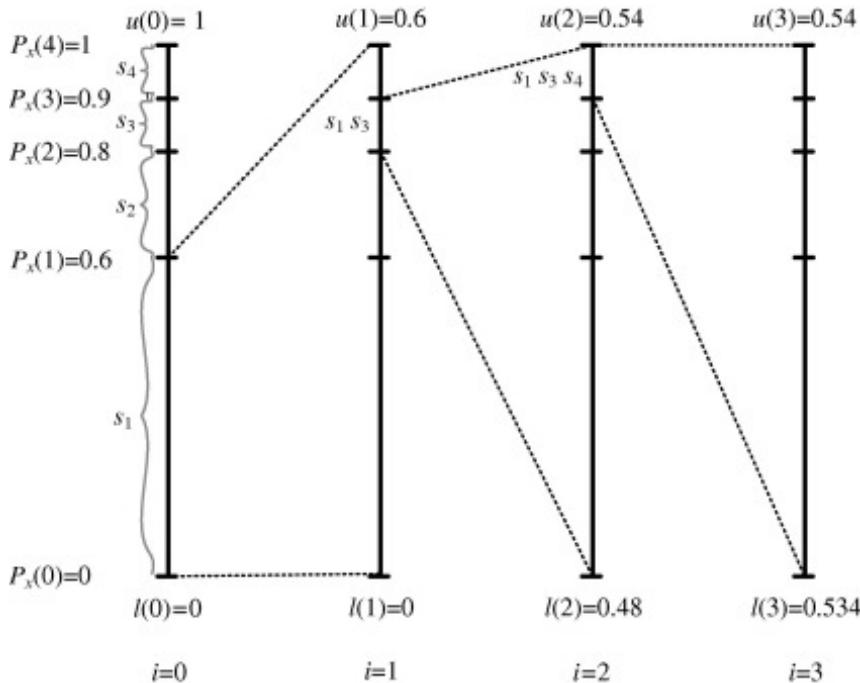
解决方案

此码字的算术解码过程的迭代如下所示：

算法7.8 算术解码

1. Initialize upper and lower limits: $l(0)=0; u(0)=1;$
2. INPUT arithmetic code tag, \hat{v} ;
3. $i=0$;
4. REPEAT
5. $i=i+1$;
6. Compute symbol index i : $\hat{x}_i = \{x : l(i) \leq \hat{v} < u(i)\}$ where
upper interval limit: $u(i) = l(i-1) + (u(i-1) - l(i-1))P_x(x_i)$ and
lower interval limit: $l(i) = l(i-1) + (u(i-1) - l(i-1))P_x(x_i - 1);$
7. OUTPUT symbol s_{x_i} ;
8. UNTIL $i = N$;
9. END.

[下载：下载全尺寸图像](#)



[下载：下载全尺寸图像](#)

我们可以看到，在第一阶段，对于解码符号1，标签位于[0,0.6]范围内，因此符号输出是 s_1 。计算

第二阶段修改后的累积概率值，我们看到标签在符号序列对应的范围内 $[0.48, 0.54]\{s_1, s_3\}$ 。

在迭代3中重复此操作会导致标签在表示序列的区间 $[0.534, 0.54)$ 中下降： $\{s_1, s_3, s_4\}$ 。

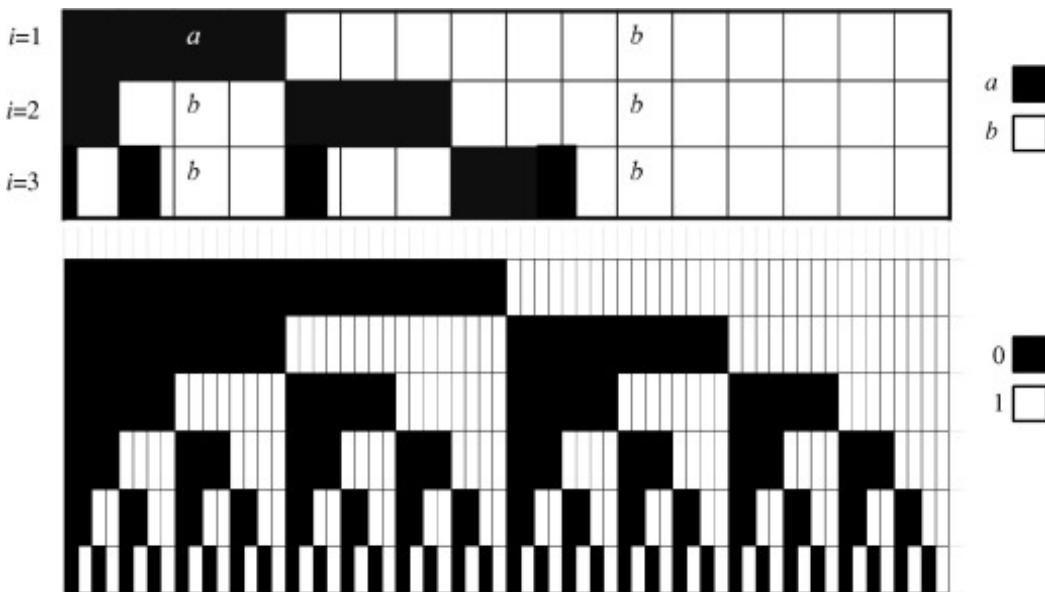
7.6.3。算术编码的优势

算术编码具有许多重要属性，使其成为熵编码的首选方法。这些包括：

1. 霍夫曼编码使用大符号字母表可以高效，其中任何单个符号的概率最高 p_{\max} 规模较小，分布相对均匀。然而，对于较小的字母表和概率分布有偏差的地方，霍夫曼编码可能效率低下。在算术编码中，**码字**表示符号字符串，从而提供了分数字长的潜力，并相应地降低总体比特率。
2. 算术编码将建模过程与编码过程分开。每个单独的符号都可以有一个单独的概率（通常称为条件概率），该概率可以预先确定，也可以根据之前的数据自适应计算。
3. 符号统计可以简单地适应不断变化的条件，而无需对编码表进行重大**重新计算**。

7.6.4。二元算术编码

在实际的算术编码系统中，二进制实现是必不可少的。这在有限的字长效应和唯一性方面对操作造成了一些限制，但也在高效实现方面提供了机会。为了帮助理解这个过程，请考虑图7.12中的插图。这显示了算术编码器或解码器的三个迭代，对于具有两个符号 $\{a, b\}$ 的系统，其中 $p(a) = 0.25$ 和 $p(b) = 0.75$ 。顶部图表显示了与我们以前看到的概率区间相似，但可视化略有不同。底部图显示了相应的**二进制码字**。通过从给定的序列垂直扫描，我们可以轻松地识别唯一定义该序列的二进制码字，并选择长度最小的码字。例如，考虑序列 $\{a, b, b\}$ ；向下扫描到底部图，我们可以看到唯一标识该序列的最短码字是001。同样，对于 $\{b, b, b\}$ ，它是11。



下载： [下载全尺寸图像](#)

图7.12。算术编码和解码过程的图形说明。顶部：三次迭代的概率区间。底部：码字间隔。显示 $P(a) = 0.25$ 和 $P(b) = 0.75$ ，示例序列为 abb 和 bbb 。

二进制编码可以在编码和解码阶段提供标签值的顺序位处理优势。如下例所示。

示例 7.12 二进制算术编码

考虑一个具有四个具有概率符号的来源： $P(s_0) = 0.5, P(s_1) = 0.25, P(s_2) = 0.125, P(s_3) = 0.125$ 。对序列进行算术编码： $S = \{s_1, s_0, s_2\}$ 。

解决方案

编码进行如下：

阶段1

第一个符号 s_1 定义区间 $I_1 = [0.5, 0.75]_{10} = [0.10, 0.11]_2$ 。检查这个区间的上下限，我们可以看到二进制点 (1) 之后的第一个二进制数字是不变的，因此可以传输或附加到输出文件中。

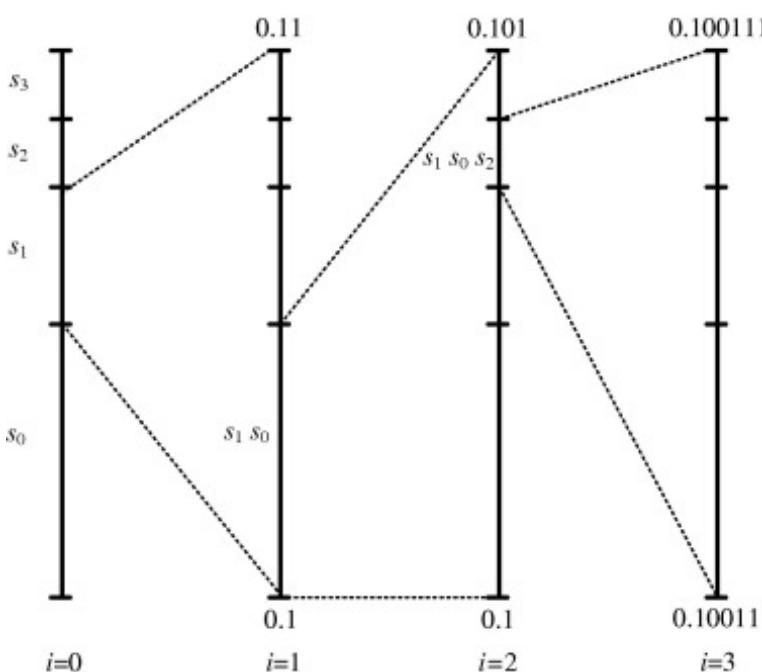
阶段2

第二个符号 s_0 定义区间 $I_2 = [0.5, 0.625]_{10} = [0.100, 0.101]_2$ 。检查这个区间的上下限，我们可以看到二进制点 (10) 之后的前两个二进制数字是不变的，因此可以传输或附加到输出文件中。

阶段3

第三个符号 s_2 定义区间和算术代码： $I_3 = [0.59375, 0.609375]_{10} = [0.10011, 0.100111]_2$ 。检查这个区间的上下限，我们可以看到二进制点 (10011) 之后的前五个二进制数字是不变的，因此可以传输或附加到输出文件中。我们可以将这个范围内的任何值作为我们的算术代码传输， $\hat{v} \in [0.10011, 0.100111]_2$ 。在这种情况下，我们只需发送一个额外的零来区分下限和上限，即 $\hat{v} = 0.100110$ 。

下图所示是该算法编码过程的简化图。



下载：下载全尺寸图像

示例 7.13 二进制算术解码

从[示例7.12](#)解码传输序列。

解决方案

下表概述了解码过程。当每个位被加载时，可以对其进行处理，以进一步区分符号。当收到所需数量的符号或EOB符号时，解码终止。考虑表的第一行：第一行没有唯一定义符号——它可以是 s_2 或者 s_3 在这种情况下。然而，当第二个位被处理时，它唯一地定义了 s_1 。重复此操作，直到解码完整序列。

Rx位	区间	符号
1	$[0.5,1)_{10} = [0.1,1.0)_2$	-
0	$[0.5,0.75)_{10} = [0.10,0.11)_2$	s_1
0	$[0.5,0.625)_{10}$ $= [0.100,0.101)_2$	s_0
1	$[0.5625,0.625)_{10}$ $= [0.1001,0.1100)_2$	-
1	$[0.59375,0.625)_{10}$ $= [0.10011,0.10100)_2$	-
0	$[0.59375,0.609375)_{10}$ $= [0.100110,0.100111)_2$	s_2

7.6.5。带缩放的标签生成

在传输的符号序列较大的情况下，算术编码过程必须迭代大量次数，每次迭代时，间隔的大小会缩小（平均减少2倍）。为了在实际情况下处理这个问题，字长必须足够确保间隔和标记能够以足够的数值精度表示。在实践中，每次传输一点标签时缩放间隔的过程都会缓解非常长的字长。考虑[示例7.12](#)中的第一阶段；此处 $l(1) = 0.5$ 和 $u(1) = 0.75$ 。一旦传输第一个二进制1，这意味着码字永远被限制在数字行的上半部分。因此，在不失去普遍性的情况下，我们可以将区间缩放为2倍，如下所示：

$$l'(1) = 2(l(1) - 0.5) = 0$$

$$u'(1) = 2(u(1) - 0.5) = 0.5$$

然后生成下一阶段的方程如下：

$$u(i) = l'(i-1) + (u'(i-1) - l'(i-1))P_x(x_i) \quad (7.12)$$

$$l(i) = l'(i-1) + (u'(i-1) - l'(i-1))P_x(x_i - 1)$$

每次新符号将标签完全包含在上半段或下半段中时，都会重复此缩放过程。如果标记区间横跨数字线的中心，我们可以执行类似的过程，但当区间在 $[0.25, 0.5]$ 以内时，可以缩放。因此，我们

有三种类型的缩放或重整合化，可以在区间细分后使用[4]，[7]：

$$\begin{aligned} E_1 &: [0,0.5) \rightarrow [0,1); \quad E_1(x) = 2x \\ E_2 &: [0.5,1) \rightarrow [0,1); \quad E_2(x) = 2(x - 0.5) \\ E_3 &: [0.25,0.75) \rightarrow [0,1); \quad E_3(x) = 2(x - 0.25) \end{aligned} \tag{7.13}$$

当标签位被处理时，不难看出如何在解码器上模拟此过程。

7.6.6。上下文自适应算术编码

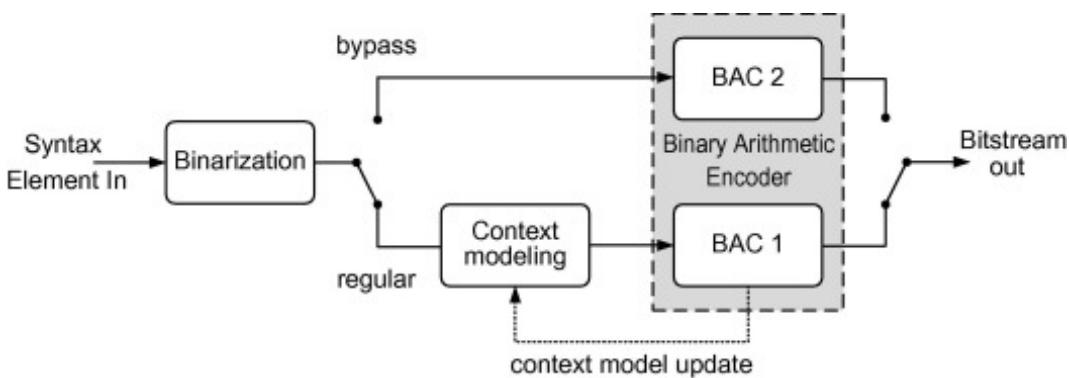
算术编码在JPEG、JPEG2000、H.263、H.264/AVC和HEVC等许多标准中得到了支持。然而，直到它被纳入H.264/AVC，它才在实践中得到普遍使用。这种采用是由于上下文自适应二进制算术编码（CABAC）[17]比以前的方法提供了显著的性能提升。CABAC现在是HEVC的标准熵编码方法。

已经证明，通过考虑当前环境并相应地调整条件概率，熵编码器可以得到显著改进。在本章前面，我们看到了一个如何使用霍夫曼编码做到这一点的例子，并查看了CAVLC。这里我们将在算术编码的上下文中简单考虑。田等人很好地回顾了自适应编码方法。[12]。

文献中报道了自适应算术编码引擎的几种实现。这些包括Q编码器[21]和MQ编码器[22]，它利用细分后重整合化的想法（如上所述）以避免数值精度问题（JPEG中使用的扩展称为QM编码器）和MQ编码器[22]。后者用于在JPEG2000 EBCOT（带优化截断的嵌入式块编码）编码器中量化小波系数码块的位平面编码。

CABAC已被H.264/AVC和HEVC的主要和高轮廓采用，并采用基于上下文的适应性来提高性能。这使得编码能够动态适应流行的符号统计信息。在CABAC中，条件概率基于以前编码的语法元素的统计信息，这些元素用于在多个概率模型之间切换。它提供了一种快速准确的方法，在短时间内估计条件反射概率，并具有出色的计算效率。

CABAC包括三个主要阶段：二进制化、上下文建模和二进制算术编码。这些如图7.13所示。这些步骤将在下面简要描述。读者请参阅参考文献[12]，[17]以获取更详细的操作解释。



下载：下载全尺寸图像

图7.13。上下文自适应二进制算术编码。

改编自参考文献。[17]。

二元化: 将非二进制值SE映射到二进制字符串（二进制决策序列）。这通过为每个非二进制SE生成一组中间码字（bin字符串）来减少符号字母表。它还可以在**子符号**级别进行上下文建模，其中**条件概率**可用于更常见的垃圾桶，用于更简单的方案用于更频繁的垃圾桶。在CABAC中，二元化使用许多方案及其组合来实现。这些包括：一元和截断一元、固定长度、 k th订购Exp-Golomb和这些组合。

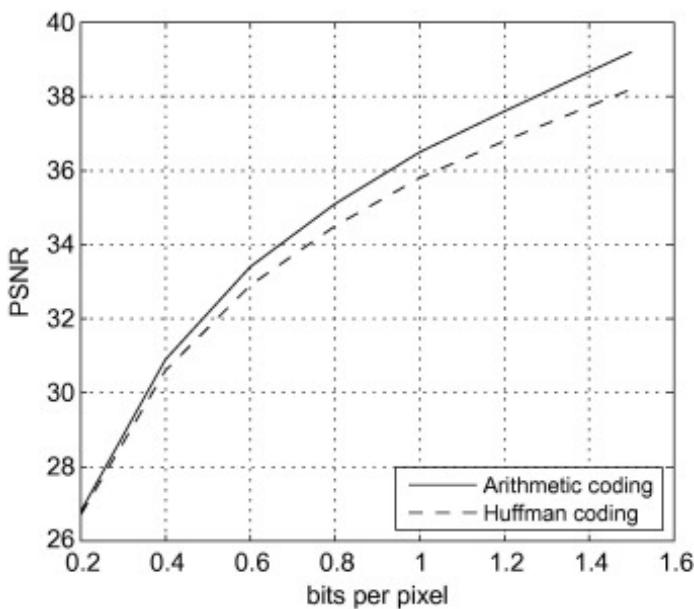
上下文建模: 我们之前已经看到，算术编码的主要优势之一是它能够将建模与编码分开。CABAC利用了这一点。建模器为正在处理的符号分配概率分布，这用于条件BAC。在CABAC中，上下文建模仅限于少数本地符号，以便在建模复杂性和性能之间提供良好的权衡。如果SE显示均匀或接近均匀概率分布，则调用旁路模式。

二元算术编码: CABAC采用无乘法算法编码方法，并使用基于Witten方法的递归区间细分和重整化[23]。

在H.264/AVC和HEVC标准中，CABAC用于编码参数（例如宏块类型、空间和时间预测模式、切片和宏块控制信息）以及预测残差。据报道，CABAC比CAVLC改进高达20%，通常节省9%至14%[17]。

7.7。性能对比——霍夫曼VS算术编码

基本算术和霍夫曼编码实现的压缩性能比较见图7.14。这些结果表示DCT编解码器与两种类型的熵编解码器的组合。这表明与霍夫曼编码相比，**算术编码**具有优势（压缩比较低时可达1分贝）。正如我们所看到的，使用比霍夫曼编码更适合算术编码的更复杂的自适应和基于上下文的技术，可以进行进一步改进。



下载：[下载全尺寸图像](#)

图7.14。霍夫曼与**算术编码**。

7.8。摘要

本章介绍了无损图像编码的方法。我们看到有用代码需要具有前缀属性的要求，以避免需要显式同步。[霍夫曼代码](#)满足此属性，并能够以接近其熵的速度从源编码单个符号。然后，我们描述了[算术编码](#)，它具有性能优势，因为它可以表示组中的多个符号，从而提供分数符号字长。它还更适合自适应编码——支持动态修改条件概率。

Recommended articles

Citing articles (0)

参考文献

[1] D.A.霍夫曼

一种最小冗余代码的构建方法

IRE会议记录, 40 (9) (1952年), p.1098-1101

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[2] N.艾布拉姆森

信息理论和编码

麦格劳-希尔 (1963)

[谷歌学术](#)

[3] J. 里萨宁

广义卡夫不等式与算术编码

IBM研究与发展杂志, 20 (1976年), 第198-203页

[CrossRef](#) [谷歌学术](#)

[4] A.赛义德

算术编码入门——理论与实践

K. Sayood (编辑), 无损压缩手册, 学术出版社 (2003)

[谷歌学术](#)

[5] J. Ziv, A. 伦佩尔

一种通用的数据压缩算法

IEEE 信息理论交易, 23 (3)(1977), p.337-343

[在Scopus中查看记录](#) [谷歌学术](#)

[6] T. 韦尔奇

一种高性能数据压缩技术

IEEE Computer (1984), 第8-19

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[7] K. 萨耶乌德

数据压缩简介

(第三版), 摩根·考夫曼 (2006)

[谷歌学术](#)

- [8] K. Sayood (编辑) , 无损压缩手册, 学术出版社 (2003)
[谷歌学术](#)
- [9] R.加拉格尔
霍夫曼关于主题的变体
IEEE 信息理论交易, 24 (6) (1978), p.668-674
[谷歌学术](#)
- [10] J. 维特
动态霍夫曼码的设计与分析
ACM杂志, 34 (4) (1987年) , 第34页。825-845
[在Scopus中查看记录](#) [谷歌学术](#)
- [11] ISO/IEC国际标准10918-1, 信息技术-连续色调静止图像的数字和编码-要求和指南, 1992年。
[谷歌学术](#)
- [12] X. 田, T. Le, Y. 廉
H.264/AVD标准的熵编码器: 算法和VLSI体系结构
斯普林格 (2011)
[谷歌学术](#)
- [13] 我。理查森
H.264高级视频编码标准
(第二版) , 威利 (2010)
[谷歌学术](#)
- [14] 美国。戈隆
运行长度编码
IEEE 信息理论交易, 12 (3) (1966) , pp.399-401
[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)
- [15] D.陶布曼, M. 马塞林
JPEG2000图像压缩基础、标准和实践
克鲁尔 (2002)
[谷歌学术](#)
- [16] J. 特霍拉
一种压缩方法, 聚类位向量
信息处理信件, 7 (1978年) , p.308-311
[文章](#) [!\[\]\(c201cde4a054ed614790508fa55bc512_img.jpg\) 下载PDF](#) [在Scopus中查看记录](#) [谷歌学术](#)
- [17] D.马普, H. 施瓦茨, T. 维冈德
H.264/AVC视频压缩标准中基于上下文的自适应二进制算法编码
IEEE 视频技术电路和系统交易, 13 (7) (2003) , p.620-636
[在Scopus中查看记录](#) [谷歌学术](#)

C.香农

[18] **一种通信的数学理论**

贝尔系统技术期刊 (30) (1948年), p.623-656

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[19] F。耶利内克

概率信息理论

麦格劳·希尔 (1968)

[谷歌学术](#)

[20] J.-R.欧姆

多媒体通信技术

斯普林格 (2003)

[谷歌学术](#)

 [下载](#)

Q编的奇日适应—进制异步编的奇基半原理论述

IBM研究与发展杂志, 32 (1988年), 第717-726页

[CrossRef](#) [在Scopus中查看记录](#) [谷歌学术](#)

[22] D.陶布曼, E。奥登特利希, M。温伯格, G。血清学

JPEG 2000中的嵌入式块编码

信号处理: 图像通信, 17 (2002年), 第49-72页

[文章](#)  [下载PDF](#) [在Scopus中查看记录](#) [谷歌学术](#)

[23] 我。威滕, R。尼尔, J。克利里

数据压缩的算术编码

ACM的通信, 30 (6) (1987年), 第520-540

[在Scopus中查看记录](#) [谷歌学术](#)

* 有关图7.9和7.10的彩色和更高质量的版本, 请参阅电子版本或网站。

¹ 这高度依赖于数据和应用程序。

² 需要注意的是, JPEG标准也支持算术编码, 但这在实践中很少使用。

版权所有©2014爱思唯尔有限公司。保留一切权利。



关于ScienceDirect

远程访问

购物车

广告

 RELX™

[联系和支持](#)[条款和条件](#)[隐私政策](#)

我们使用cookie来帮助提供和增强我们的服务，并定制内容和广告。继续即表示您同意**使用cookie**。

版权所有©2021爱思唯尔B.V.或其许可方或贡献者。ScienceDirect®是爱思唯尔B.V.的注册商标。

ScienceDirect®是爱思唯尔B.V.的注册商标。

[FEEDBACK](#) 