基于上下文模型的超长哈夫曼码校正算法

张永兴,吴睿振,贾晓龙,陈静静,孙华锦
(浪潮人工智能研究院有限公司,陕西 西安 710077)

摘　要:常见的 Gzip、Zlib 数据压缩标准都采用 Deflate 协议压缩封装数据,Deflate 协议中采用哈夫曼码编码源符号(Source symbols)。哈夫曼编码算法通过构建哈夫曼树生成哈夫曼码,Deflate 协议限定源符号的哈夫曼码的码长不能超过最大值。源符号的哈夫曼码长最大值等于哈夫曼树的高度,因此当哈夫曼树的高度超过限定值时,需要先把哈夫曼树进行“校正”,随后再为每个符号分配。Gzip、Zlib 软件参考代码中使用的基于二叉树搜索的“校正”算法,校正时需要遍历搜索哈夫曼树,寻找嫁接“节点”。校正流程时间消耗非常大,而且硬件实现难度较大。该文探索一种基于上下文模型校正超长哈夫曼树的算法,与参考二叉树搜索算法相比:该算法可以快速校正超长哈夫曼树,将校正的时间消耗降至为 0,而且对压缩效果几乎没有影响(压缩比平均下降率仅为 0.372%)。该算法也易于硬件化实现,可以实时校正超长哈夫曼码。

关键词:Deflate;哈夫曼编码;哈夫曼树;超长 Huffman 码;超长 Huffman 码校正
中图分类号:TP309.3　　**文献标识码**:A　　**文章编号**:1673-629X(2023)02-0092-07
doi:10.3969/j.issn.1673-629X.2023.02.014

Correction Algorithm for Ultra-long Huffman Codes Based on Context Model

ZHANG Yong-xing, WU Rui-zhen, JIA Xiao-long, CHEN Jing-jing, SUN Hua-jin
(Inspur Artificial Intelligence Research Institute Co., Ltd., Xi'an 710077, China)

Abstract: Common data compression standards such as Gzip and Zlib compress and encapsulate data with the Deflate format. In Deflate protocol, Source symbols are encoded with Huffman codes that are distributed by constructing the Huffman tree. In the "Deflate" protocol, the Huffman codes for the various alphabets must not exceed certain maximum code lengths. Therefore, if the height of First Huffman tree was greater than the maximum value, First Huffman tree needs to be "corrected" before generating Huffman code. The current "correction" algorithm based on the binary tree consumes a lot of time in the correction process and is difficult to implement in hardware. We explore an algorithm to quickly correct ultra-long Huffman trees based on context models. Compared to exist correcting algorithm, the ultra-long Huffman trees could be quickly corrected with a consuming time of nearly 0. Moreover, mean compression ratio is only reduced by 0.372%. The proposed algorithm is suitable for hardware implementation and can correct ultra-long Huffman codes in real time.

Key words: Deflate; Huffman coding; Huffman tree; ultra-long Huffman codes; correction of ultra-long Huffman codes correction

0　引　言

随着大数据等前沿科学技术的快速发展,催生数据爆发式的增长,海量数据对现有的存储设备带来巨大的压力。面对持续增长的海量数据,数据压缩成为减轻服务器存储负担,降低存储成本的最有效方法。

数据压缩在不丢失有用信息的前提下,缩减数据量以减少存储空间,提高传输、存储和处理效率。无损数据压缩一般通过两种方法来实现[1]:一种是通过字典的方式实现压缩的算法,包括 LZ 系列算法,这类算法能实现重复数据的搜索功能;另一种是基于统计模型的压缩算法,如 Huffman 码、算术编码等,这类算法的核心思想是依照符号出现频率分配码长,符号出现频率越高,其对应码长就越短。

当前主流的数据压缩算法通常会把原始数据切割分块,然后对每个数据分块独立压缩编码。压缩数据通常由一系列压缩块(compressed blocks)组成,这些压缩块对应于原始数据的连续块。最常见的数据压缩算法(Gzip、Zip、Zlib 等)会将原始数据块压缩编码成一种名为 Deflate 的压缩数据块[2]。Deflate 是一种将 LZ77 算法和 Huffman Coding 结合起来的无损数据压

收稿日期:2022-03-23　　**修回日期**:2022-07-26
作者简介:张永兴(1987-),男,硕士,通信作者,研究方向为数据压缩算法。

缩协议。

LZ 系列算法最早由 Ziv 和 Lempel[3-4] 在 1977 年和 1978 年的两篇论文中提出。这两篇论文构造了两个不同类型的数据压缩算法,LZ 系列的核心重复数据搜索查询。基于 1977 年论文的数据压缩算法都称为 LZ77 数据压缩算法。LZ77 算法是利用动态字典实现重复数据的查找:首先利用已有数据作为动态字典,通过检查字典,判断当前输入的数据是否在滑动窗口内的先前数据中出现过,随后对重复数据进行编码。

哈夫曼编码[5](Huffman coding)是由 David A. Huffman 于 1952 年提出的一种基于频率统计的变长编码。在数据压缩过程中,通过构建 Huffman 树来生成源符号的 Huffman 码。Huffman 树的高度决定源符号的最长 Huffman 码。Deflate 协议限定 Huffman 码长不能超过 15,因此当 Huffman 树的高度超过限定值时,需要对 Huffman 树“校正”,再生成哈夫曼码。Zlib 等参考软件代码里的基于二叉树搜索的校正算法,需要遍历搜索整棵 Huffman 树,寻找嫁接“节点”,校正流程时间消耗非常大,而且不利于硬件化实现。

1　Huffman 算法原理

Huffman 编码[6]是一种基于数据统计的变长编码算法。哈夫曼编码已经被证明是一种编码效率最佳的变长编码方案,所以哈夫曼编码也被叫做最佳编码。当前哈夫曼编码被广泛应用于图像、视频、文本等不同类型数据压缩编码中,JPEG、JPEG2000 等图像压缩格式中运用哈夫曼码编码图像数据[7-8],H263 视频编解码标准[9]中使用哈夫曼码编码视频流数据,Gzip[10]、Zlib[11]等数据压缩标准都用到哈夫曼编码实现文本数据压缩。

Huffman 编码算法利用符号的频率分布构建 Huffman 树,从而生成每个符号相对应的哈夫曼码[5]。Huffman 编码算法的流程如图 1 所示,大体可分解为如下三个步骤:

(1)统计源符号中各个符号出现的频率,并按照频率对符号进行排序。Deflate 协议中的源符号(Source symbols)是原始数据 LZ77 算法搜索查重后输出的待编码符号[2],包括原文字母(Literal)、匹配长度(Match length)、偏移距离(Match distance)。

(2)构建源符号的 Huffman 树。

(3)利用 Huffman 树生成 Huffman 码表。

图 1　Huffman 编码流程

Huffman 编码算法的精妙之处在于,该算法完全依照各个符号的频率,合理精确地为每个符号分配码长,Huffman 编码是最接近信息熵的变长编码方案。其中算法原理 Huffman 编码的精髓在于构建 Huffman 树。该文以“abcdefghacdefghacdefhacdehadehaehaee”字符串为例,详述 Huffman 树的构建算法。

构建 Huffman 树时,需要统计各个符号出现的次数,按照出现的次数从小到大对符号进行排序。上述字符串由“a”、“b”、“c”、“d”、“e”、“f”、“g”、“h”八个符号组成,按照它们的出现次数从大到小排序如表 1 所示,下面详细描述 Huffman 树的构建流程。

表 1　源符号排序序列

序列	字符	频率
1	e	8
2	g	7
3	h	6
4	d	5
5	c	4
6	f	3
7	g	2
8	b	1

1.1　Huffman 树构建算法

Huffman 树数学形式上就是一种二叉树,文献[5]中 David A. Huffman 详述了 Huffman 树的构建过程,对源符号(节点)进行多轮迭代排序及合并,构建 Huffman 树。在每一轮中,会把符号序列中频率最小的两个符号合并生成一个新符号(父节点),新符号的频率为两个符号的频率相加值,同时需要将这两个符号移出符号系列。生成 Huffman 树所需的轮数与源符号数目减 1。下面以表 1 所示符号序列为例,详述整个 Huffman 树构建流程:

第 1 轮:原始序列中最小的两个频率相加(“1”和“2”),合并为一个概率为 3 的节点,新序列重新排序结果为 8,7,6,5,4,3(1+2),3;

第 2 轮:把第 1 轮生成新序列中最小的两个频率相加(“3”和“3”),合并为一个概率为 6 的节点,新序列重新排序结果为 8,7,6(3+3),6,5,4;

第 3 轮:把第 2 轮生成新序列中最小的两个频率相加(“4”和“5”),合并为一个概率为 9 的节点,新序列重新排序结果为 9,8,7,6,6;

第 4 轮:把第 3 轮生成新序列中最小的两个频率相加(“6”和“6”),合并为一个概率为 12 的节点,新序列重新排序结果为 12,9,8,7;

第 5 轮:把第 4 轮生成新序列中最小的两个频率相加(“7”和“8”),合并为一个概率为 15 的节点,新序列重新排序结果为 15,12,9;

第 6 轮:把第 5 轮生成新序列中最小的两个频率

相加(“9”和“12”),合并为一个概率为 21 的节点,此时序列中仅剩两个节点 15,21;

第 7 轮:把第 6 轮生成最后的两个节点相加,合并为一个概率为 36 的根节点。

以上迭代排序及合并流程,可以用图 2 形象描述。

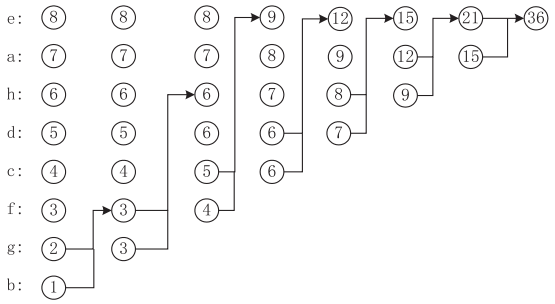


图 2 源符号迭代排序流程

图 2 所示的迭代排序流程,就是构建 Huffman 树的流程,上述例子最终生成的 Huffman 树如图 3 所示。

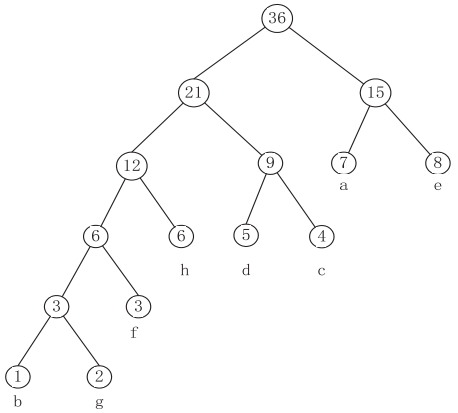


图 3 Huffman 树示意图

1.2 超长 Huffman 码的问题

在构建 Huffman 树时,Huffman 树的高度通常定义为 Huffman 树上叶子节点的最大码长。如图 3 所示,Huffman 树的深度为 5。根据 Huffman 算法原理可知,Huffman 树的深度与源符号的(叶子节点)的概率分布相关。

由二叉树相关理论可知[12],理论最大高度 $Height_{max}$ 等于二叉树的叶子节点数目 $Num_{leaf}-1$,即式(1):

$$Height_{max} = Num_{leaf} - 1 \quad (1)$$

当二叉树呈现出如图 4 所示(树上的字母(“a”–“g”)表示 Huffman 树的叶子节点,树上数字表示枝节点,图中右侧的数字表示每个叶子节点的码长)的树形,此时就是理论最大高度的情形。Deflate 格式的编码数据块由三种类型不同的字符集构成:原文字母、匹配长度、偏移距离。Deflate 协议中,需要构建两种类型的哈夫曼树:Literal & length 树和 Distance 树。Literal & length 树的源符号总数为 286,其中值 0…255 表示原文字母,值 256 表示块结束符,值 257…285 表示匹配长度,Distance 树的源符号总数为 30。因此,这两棵树的理论最大深度为 285 和 29。Deflate 协议中规定这两棵哈夫曼树的最大高度都为 15[2],因此当由源符号经频率排序生成的 Huffman 树(该文把此阶段的 Huffman 树称为原树)高度超过规定最大值(15)时,需要先对原树“修整”,然后再用校正后的 Huffman 树生成 Huffman 码。Deflate 协议规定 Huffman 码长的最大值,但是没有规定校正算法。

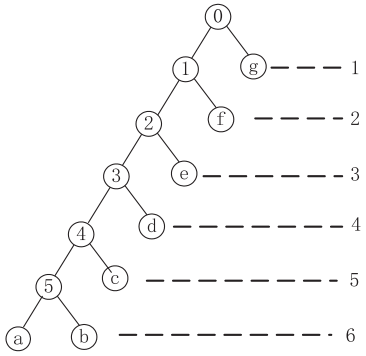


图 4 Huffman 树图距离

1.3 超长 Huffman 码校正算法

在 Zlib、Gzip 等压缩标准的参考代码里,使用一种基于二叉树搜索的校正方案。以图 4 为例,假定此树的规定最大高度为 5,需要对该树上码长大于 5 的叶子节点进行校正。校正流程描述如下:

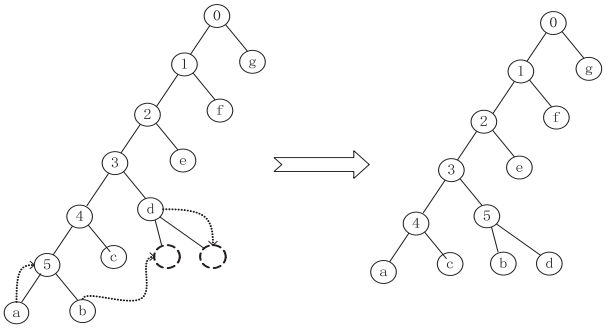
第 1 步:遍历整棵 Huffman 树,找到一对码长最长的兄弟叶子节点(“a”和“b”)。

第 2 步:遍历整棵 Huffman 树,找到一个比规定码长小 1 的叶子节点(“d”)。

第 3 步:把第 1 步中兄弟节点的一个放入父节点的位置,裁取另一个叶子(把“a”节点放入“5”节点位置,裁取“b”节点)。

第 4 步:把第 1、2 步中选取的叶子节点(“d”与“b”)组合成一对兄弟节点,它们的父节点的位置就是“d”的原位置。

上述软件校正流程如图 5 所示,需要遍历二叉树找到超长节点位置跟“嫁接位置”。由于二叉树无法直接通过索引寻址,需要搜寻二叉树,通过链表寻址方式找到目标节点。要想用硬件化实现上述软件校正流程,由于硬件很难实现对二叉树搜索,此外整个搜索流程耗时非常大。某些情形下,Huffman 码树上超长的叶子节点可能会存在很多,以上遍历搜索流程不能并行实现处理,这会加剧校正哈夫曼码树的时间消耗,极端情形下,校正 Huffman Tree 的时间消耗会数倍于构建哈夫曼码树的时间消耗。如上所述,二叉树搜索的校正方案校正超长码的时间消耗较大,并且不利于硬件实现,因此十分有必要找到一种易于硬件实现的快速校正方案。



注:1. 找到一对超长的兄弟节点(“a”和“b”);2. 找到嫁接位置“d”;3. 把“a”节点放入“5”所示的位置;4. 将“b”与“d”组成一对兄弟节点。

图 5　超高 Huffman 树校正示意图

2　基于上下文模型的校正方案

2.1　超长 Huffman 码概率统计实验

由二叉树理论可知,Deflate 协议中 Huffman 码长理论上会超过限定的最大值。为了探究超长 Huffman 码的发生概率,利用 Zlib 参考代码设计如下实验:

(1)运用 Zlib 代码压缩参考数据集(Silesia & Canterbury)里面每个文件,统计每个文件分块数目 block-number。

(2)Zlib 代码中包含校正超长 Huffman 码的函数接口,通过统计调用校正接口的次数来统计超长 Huffman 块的数目 ultra-block-number。

(3)超长码的发生概率计算方法如式(2):

$$P_{\text{ultra\_block}} = \frac{\text{ultra - block - number}}{\text{block - number}} \times 100\% \qquad (2)$$

上述超长码概率统计实验数据如表 2 所示,最后计算得超长 Huffman 码的平均发生概率为 35.5%,由此说明超长 Huffman 码的出现频率比较高。所以,有必要找到一种快速校正哈夫曼码树的硬件方案。

表 2　参考数据集超长 Huffman 频率统计

文件名	分块数	超长块数	超长率/%	文件名	分块数	超长块数	超长率/%
alice29_txt	2	0	0.0	dickens	78	68	87.2
asyoulik_txt	1	1	100	mozilla	391	248	63.46
bible_txt	31	14	45.2	mr	77	5	6.58
cp_html	1	0	0.0	nci	256	0	0.0
E_coli	36	0	0.0	ooffice	47	43	91.575
fields_c	1	0	0.0	osdb	77	62	80.575
grammar_lsp	1	0	0.0	reymont	51	14	27.575
kennedy_xls	8	6	75	samba	165	21	12.775
lcet10_txt	4	0	0.0	sao	56	45	80.475
plrabn12_txt	4	3	75	xml	41	1	2.475
ptt5	4	1	25	webster	317	67	21.175
sum	1	0	0.0	x-ray	65	1	1.575
world192_txt	19	16	84.2	总计	1 735	616	35.5
xargs_1	1	0	0.0				

2.2　上下文间 Huffman 码表相似性实验

上下文模型是利用上文与下文之间的相似性构建的算法模型。在数据压缩编码中,上下文模型有广泛的应用:H264、HEVC 等视频编解码标准中采用的 CAVLC、CABAC 两种熵编码方案都是基于上下文模型构建[13-14];LZ77 算法为了搜索重复数据构建的字典也是一种上下文模型。

假设上下文之间的哈夫曼码表具有很高的“相似性”,可以利用上文中的正常哈夫曼码表来对下文的源符号(source symbol)进行编码,从而规避超长的哈夫曼码树,实现“校正”超高哈夫曼码树的功能。

上述校正方案成立的假设条件是上下文之间的哈夫曼码表具有很高的“相似性”,为了测试上下文之间的哈夫曼码表具有很高的相似性,设计如下实验,利用 PSNR(峰值信噪比,Peak Signal to Noise Ratio)指标评价上下文间哈夫曼码表的相似性。

在图像视频领域,PSNR 是一种评价图像质量的客观标准。图像(视频)压缩领域通常为有损压缩,即编解码后的影像会跟原始影像不同,编解码领域一般采用 PSNR 值作为衡量编解码算法的性能指标[12]。PSNR[15]的计算方法如公式(3):

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{(\text{Peak})^2}{\text{MSE}}\right) \qquad (3)$$

式中,Peak 为像素的最大值,Peak 值与像素的 bit 数 n 有关:$\text{Peak} = 2^n - 1$。

MSE 为原始影像与编解码后影像之间的均方差,

MSE 的计算方法如公式(4):

$$\mathrm{MSE} = \frac{1}{N} * \sum_{i=0}^{N} (P_2[i] - P_1[i])^2 \quad (4)$$

式中, N 为图像中像素的数目, P 为像素值。所以, PSNR 也可用公式(5)计算:

$$\mathrm{PSNR} = 10 \times \log_{10}(\frac{(2^n - 1)^2}{\mathrm{MSE}}) \quad (5)$$

PSNR 不仅是一种评价图像质量的指标,也是一种判断图像相似性的指标。事实上从公式的内容看, PSNR 就是通过计算两幅图像(原始图像与处理后图像)之间的差异性来评价图像质量的。HEVC、VP9 等视频编解码标准用 PSNR 作为评价视频中前后帧相似性的指标[13-14],PSNR 值越大,相似性越高。在本实验中,利用 PSNR 指标衡量前后两个数据块的 Huffman 树的相似性计算。

Deflate 中 Huffman 码长用 4bit 数字标识,即公式(5)中 n 取值为 4;公式(5)可以转换成公式(6):

$$\mathrm{PSNR} = 10 \times \log_{10}(\frac{(2^4 - 1)^2}{\mathrm{MSE}}) = 10 \times \log_{10}(\frac{225}{\mathrm{MSE}}) \quad (6)$$

Deflate 协议中 Literal & length 的符号数目为 286, Distance 的符号数目为 30;所以两种 Huffman 码表的均方差公式分别如下:

$$\mathrm{MSE}_{\mathrm{literal}} = \frac{1}{286} * \sum_{i=0}^{286} (\mathrm{literal\_cl}_2[i] - \mathrm{literal\_cl}_1[i])^2 \quad (7)$$

$$\mathrm{MSE}_{\mathrm{dist}} = \frac{1}{30} * \sum_{i=0}^{30} (\mathrm{dist\_cl}_2[i] - \mathrm{dist\_cl}_1[i])^2 \quad (8)$$

式(7)为 Literal & length 码表均方差公式,其中 $\mathrm{literal_cl}_1[\,]$为上文码表,$\mathrm{literal_cl}_2[\,]$为下文码表;

式(8)为 Distance 码表的均方差公式,其中 $\mathrm{dist_cl}_1[\,]$为上文码表,$\mathrm{dist_cl}_2[\,]$为下文码表。

运用公式(6)~(8)统计计算测试集中上下文之间两种 Huffman 码表的相似性(PSNR)数据。图 6 为测试数据集中三种典型文件数据的上下文之间的 PSNR 数值分布图。“bible”、“mr”、“cp. html”分别代表文本、图像、网页三种最常见的格式文件;其中“bible”是一个文本数据[16],“mr”是一张医疗诊断照片[17],“cp. html”是一个 html 格式网页文件[16]。

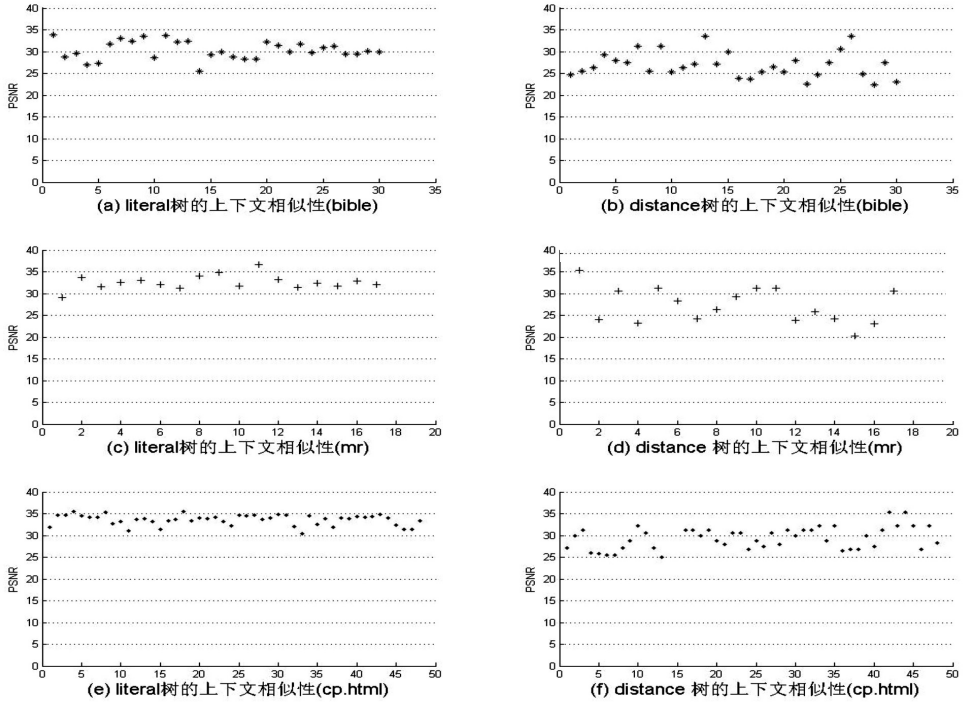


图 6 上下文之间 Huffman 码表相似性(PSNR)散点图

图中第 1 列为上下文之间 Literal-Huffman 码表的相似数据,PSNR 的值总体大于 30。第 2 列为上下文之间 Distance-Huffman 码表的相似数据,PSNR 的值总体大于 30。说明两种 Huffman 树的上下文间的相似性比较高。

统计并计算图 6 每个子图中 PSNR 数据的中值,评估上下文之间总体PSNR取值,计算结果见表3。

表 3 Huffman 码表上下文间 PSNR 统计值(中值)

文件名	中值(literal)	中值(Distance)
bible	29.956 0	26.392 0
mr	32.403 0	26.252 0
cp. html	33.772 0	29.842 0

之所以统计计算中值,而不是均值,是因为块间的 Huffman 码表如果完全一致,其 MSE 的值为 0,PSNR 取值无穷大,均值也无穷大。

由图 6 以及表 3 可以看出,对于 Literal 类型的 Huffman 码表,三个文件的上下文之间的 PSNR 中值分别为 29.956 0(bible)、32.403 0(mr)、33.772 0(cp.html)。可以认为 Literal 类型码表 PSNR 总体取值在 30 以上。对于 Distance 类型的 Huffman 码表,三个文件的上下文之间的 PSNR 中值分别为 26.392 0(bible)、26.252 0(mr)、29.842 0(cp.html)。由此可以认为总体取值在 25 以上。通常采用以下 PSNR 经验阈值来评定相似性[9,13-14]:PSNR 如果大于 35,接近一致;PSNR 如果大于 25,高度一致。PSNR 如果小于 15,相似性较低。因此可以认定,上下文之间的两种哈夫曼码表都具有很高的相似性,即上下文之间哈夫曼码表具有高度相似性的假设成立。

2.3　基于上下文模型快速校正超长 Huffman 码方案

基于上下文模型设计了一种快速校正超长 Huffman 码的硬件方案(如图 7 所示),在常规 Deflate 编码方案中添加一组参考 Huffman 码表(该文采用的上下文模型)。参考 Huffman 码表用来保存最新的常规的 Huffman 码表,当发现哈夫曼树超长时,会直接运用参考 Huffman 码表编码源数据,编码流程中会对参考 Huffman 码表更新。

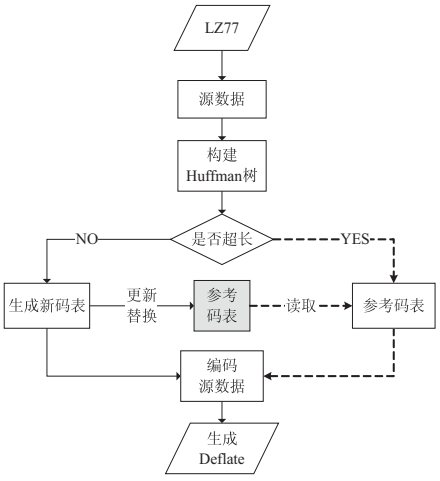


图 7　基于上下文模型校正超长 Huffman 码流程

第 1 步:Deflate 模块对 LZ77 输出的源数据进行统计排序,并构建 Huffman 树。

第 2 步:Huffman 树构建完成后,判断 Huffman 树的最大节点深度是否超出限定值。

如果原树的高度不超过限定值,此时不需要校正 Huffman 树,执行常规编码流程(第 3 步)。

如果原树高度超出限定值,此时需要执行校正流程(第 4 步)。

第 3 步:常规流程,如前文所述。利用 Huffman 树生成常规 Huffman 码表,并利用 Huffman 码表编码源数据。同时会利用参考 Huffman 码表记录常规 Huffman 码表,即用本次生成的码表刷新(写)参考码表的数值,伪代码算法 1。参考码表会参与下文中超长 Huffman 码的校正。

算法 1:Huffman 码表参考算法。

```
for index = 0 : max_symbol
    Ref - Huff - table [ index ]. length ← Huff - table [ index ]. length
    Ref - Huff - table [ index ]. code ← Huff - table [ index ]. code
end
```

第 4 步:校正流程(图),利用参考 Huffman 码表实现校正功能:直接用参考 Huffman 码表编码源数据,即当前块数据采用与上一个块相同的 Huffman 码表进行数据编码。校正流程中无需更新参考码表。

如上所述,在校正流程中,直接利用上文中已生成的常规 Huffman 码表编码下文的源数据,因此该校正方案会将校正 Huffman 树的时间消耗降为零,可以提高 Deflate 的压缩效率。

3　上下文模型校正方案的测试实验

上文中提出一种基于上下文模型的校正方案,该方案具有校正效率快的优点,同时还需要获取该方案的数据压缩性能(压缩比,Ratio)。为此,实验中选用常见的两个压缩测试数据集 Canterbury 和 Silesia[16-18] 测试评估上下文校正方案的压缩性能。这两组测试数据集既有文本、图像、网页等传统类型的数据文件,也包含数据库、多媒体、程序文件等新型数据文件。运用两种方案(上下文模型与二叉树搜索方案)压缩数据集内的所有文件(表 2 中所示的不存超长块的文件无需再测试),通过对比两种方案的压缩比,评估上下文模型校正方案的性能。表 4 为测试实验的数据统计。

表 4 对比两种校正方案的压缩性能。第 1 列(文件名)与第 2 列(Size)表示测试集内文件名及其文件大小;第 3 列为运用软件算法压缩文件得到的压缩比(Ratio);第 4 列用上下文模型方案压缩得到各个测试文件的压缩比;第 5 列(Delta)代表两种方案之间的压缩比差值(第 4 列数字减去第 2 列数字);第 6 列(Delta-rate)的数据含义为该方案与软件方案相比,压缩比的增长率($Delta/Ratio_{软件}$);最后一行的数据为总体数据的均值。

由第 5 列、第 6 列的数据可以直观看出,与软件方案相比,上下文校正方案对于压缩性影响非常小:压缩比仅仅下降 0.009 4,下降的百分率为 0.372%。如果从压缩比较评估,设计的基于上下文模型的校正方案对于压缩性能的影响几乎忽略不计。在基于上下文模

型的校正方案中,规避掉超长 Huffman 码,直接使用参考码表参与 Deflate 编码,与软件校正方案相比,上下文校正方案省去“搜索”、“嫁接”等环节,因此该方案的校正时间为零。此外软件方案“搜索”、“嫁接”等子流程,不易用硬件实现,而基于上下文模型的校正方案易于硬件化实现。

表 4 两种校正方案的压缩对比

文件名	Size(B)	Ratio(软件)	Ratio(上下文)	Delta	Delta-rate/%
asyoulik_txt	125 179	2.070 4	2.067 4	-0.003 0	-0.14
bible_txt	4 047 392	2.509 2	2.476 0	-0.033 2	-1.32
kennedy_xls	1 029 744	3.518 4	3.516 1	-0.002 3	-0.07
plrabn12_txt	481 861	1.991 1	1.987 8	-0.003 4	-0.17
ptt5	513 216	6.520 9	6.519 6	-0.001 3	-0.02
world192_txt	2 473 400	2.030 5	2.026 7	-0.003 8	-0.19
dickens	10 192 446	2.068 2	2.064 7	-0.003 5	-0.17
mozilla	51 220 480	2.149 5	2.129 8	-0.019 7	-0.92
mr	9 970 564	2.454 2	2.437 7	-0.016 5	-0.67
office	6 152 192	1.627 1	1.613 3	-0.013 8	-0.85
osdb	10 085 684	1.470 0	1.468 9	-0.001 1	-0.07
reymont	6 627 202	2.426 4	2.406 2	-0.020 3	-0.84
samba	21 606 400	2.884 5	2.872 7	-0.011 7	-0.41
sao	7 251 944	1.210 7	1.199 2	-0.011 5	-0.95
xml	5 345 280	4.237 1	4.232 2	-0.004 8	-0.11
webster	41 458 703	2.538 2	2.534 6	-0.003 6	-0.14
x-ray	8 474 240	1.263 0	1.256 7	-0.006 3	-0.50
Mean		2.527 6	2.518 2	-0.009 4	-0.372

4 结束语

提出一种基于上下文模型校正超长 Huffman 码的方案,方案会利用参考 Huffman 码表保存上文中生成的 Huffman 码表,参考 Huffman 码表会用于编码下文的存在超长 Huffman 码的数据块。论文中所述基于上下文模型的校正方案可以快速实现对超长 Huffman 码的校正,并且易于通过硬件电路实现。与此同时,相对 Zlib 等软件代码中校正方案,将该校正方案用于数据压缩,测试数据的整体压缩比下降仅为 0.009 4,下降率仅仅为 0.372%,由此可以认为数据的压缩效果几乎没有区别。综上,基于上下文模型校正超长 Huffman 码的方案是一种快速的硬件校正方案。

参考文献:

[1] BASSIOUNI M A. Data compression in scientific and statistical databases[J]. IEEE Transactions on Software Engineering,1985,11(10):1047-1058.

[2] DEUTSCH L P. DEFLATE compressed data format specification[EB/OL]. 2020. ftp://ftp. uu. net/pub/archiving / zip/doc/. 2020,9.

[3] ZIV J,LEMPEL A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 1977,23(3):337-343.

[4] ZIV J,LEMPEL A. Compression of individual sequences via variable-rate coding[J]. IEEE Transactions on Information Theory,1978,24(5):530-536.

[5] HUFFMAN D A. A method for the construction of minimum redundancy codes[J]. Proceedings of the Institute of Radio Engineers,1952,40(9):1098-1101.

[6] HUffMAN D A. A method for the construction of minimum-redundancy codes[J]. Resonance,2006,11(2):91-99.

[7] WALLACE G K. The JPEG still picture compression standard[J]. IEEE Transactions on Consumer Electronics,1992, 38(1):18-34.

[8] BARDA J F. JPEG 2000,the next millennium compression standard for still images[C]//Proceedings IEEE international conference on multimedia computing and systems. Florence:IEEE,1999:1126-1127.

[9] MORIMOTO C,BURLINA P,CHELLAPPA R. Video coding using hybrid motion compensation[C]//Proceedings of international conference on image processing. Santa Barbara: IEEE,1997:89-92.

[10] DEUTSCH L P. GZIP compressed data format specification [EB/OL]. 2020. ftp://ftp. uu. net/pub/archiving/zip/ doc/.

(下转第 104 页)