2025 年 第 25 卷 第 8 期
2025, 25(8): 03304-12

科 学 技 术 与 工 程
Science Technology and Engineering

ISSN 1671—1815
CN 11—4688/T

DOI:10.12404/j.issn.1671-1815.2401129
引用格式:陈垦, 钟爱平, 文煜轩, 等. 面向桥梁健康监测的数据压缩算法[J]. 科学技术与工程, 2025, 25(8): 3304-3315.
Chen Ken, Zhong Aiping, Wen Yuxuan, et al. Data compression algorithms for bridge health monitoring[J]. Science Technology and Engineering, 2025, 25(8): 3304-3315.

面向桥梁健康监测的数据压缩算法

陈垦[1], 钟爱平[1], 文煜轩[2,3], 杨洋[1], 李伟[1], 曾山[2,3], 王俊[1], 谭屈山[1], 杨柳[2,3*]

(1. 四川数字交通科技股份有限公司, 成都 610095; 2. 西南交通大学综合交通大数据应用技术国家工程实验室, 成都 611756;
3. 西南交通大学信息科学与技术学院, 成都 611756)

摘 要 基于传感器数据采集的桥梁健康监测系统已经成为新建桥梁的标配,在这种场景下所带来的是海量监测数据难以存储的问题。因此,针对桥梁监测数据的时间序列特性,研究桥梁监测数据的压缩方案,该方案研究了基于桥梁监测时间戳数据等差数列性质的差量压缩法和基于监测值数据变化频率不大的浮点数异或(exclusive OR,XOR)压缩法。与 Gorilla 时序数据库的算法相比,增加了异或压缩法的控制位,避免了压缩结果的恶化。通过实验分析得出,两算法对比常用压缩器有不同程度的优势,时间戳序列差量压缩法在压缩率上优于常用压缩器,针对符合等差数列特性的时间戳序列,压缩率 0.015 6,接近压缩极限值,压缩解压速度位居中上,并且对监测类型不敏感。而异或压缩法在变化频率不大的数据集上表现较好,压缩率 0.302 8,在非桥梁数据集上压缩率 0.662 8,表明异或压缩法对监测类型比较敏感。在桥梁监测的实际应用场景中,可以根据桥梁监测数据集的特点选择合适的压缩存储方案。

关键词 桥梁监测; 时序数据; 差量压缩; 异或(XOR)压缩
中图法分类号 TP391; **文献标志码** A

Data Compression Algorithms for Bridge Health Monitoring

CHEN Ken[1], ZHONG Ai-ping[1], WEN Yu-xuan[2,3], YANG Yang[1], LI Wei[1],
ZENG Shan[2,3], WANG Jun[1], TAN Qu-shan[1], YANG Liu[2,3*]

(1. Sichuan Digital Transpotation Tech Co., Ltd., Chengdu 610095, China; 2. National Engineering Laboratory of Integrated Transportation Big Data Application Technology, Southwest Jiaotong University, Chengdu 611756, China;
3. School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China)

[**Abstract**] The bridge health monitoring system based on sensor data acquisition has become standard for new bridge construction. However, this scenario presents challenges due to the massive volume of monitoring data that is difficult to store. Therefore, focusing on the time-series characteristics of bridge monitoring data, compression schemes were explored for bridge monitoring data. Differential compression was investigated based on the arithmetic progression properties of bridge monitoring timestamps and floating-point exclusive OR(XOR) compression based on the low frequency of changes in monitoring value data. Compared to the Gorilla time series database algorithm, the XOR compression method added control bits to avoid degradation of compression results. Experimental analysis reveals that both algorithms exhibit varying degrees of superiority over common compressors. The differential compression of timestamp sequences demonstrates superior compression rates compared to common compressors, achieving a compression rate of 0.015 6 for timestamp sequences that conform to arithmetic progression characteristics, approaching the compression limit value. Compression and decompression speeds are above average, and the method is insensitive to monitoring type. On the other hand, the XOR compression method performs well on datasets with low frequency of change, achieving compression rates of 0.302 8 for bridge data and 0.662 8 for non-bridge data, indicating sensitivity of the XOR compression method to monitoring type. In practical applications of bridge monitoring, suitable compression storage schemes can be selected based on the characteristics of the bridge monitoring dataset.
[**Keywords**] bridge monitoring; time series data; differential compression; exclusive OR(XOR) compression

桥梁是中国工程科技的重要组成部分,大规模建设时期出现了一大批桥梁工程,由于其长寿命,工作重心逐渐转向养护。实时监测对于避免安全事故和减少损失至关重要。传感器数据采集方式确保了

收稿日期: 2024-02-20; **修订日期**: 2024-12-15
基金项目: 四川省科技创新基地(平台)和人才计划(2022JDR0356);四川省科技计划(软科学项目)(2021JDR0101);宜宾市双城市校协议专项科研经费科技项目(SWJTU2021020005);新一代人工智能国家科技重大专项(2022ZD0115600)
第一作者: 陈垦(1986—),男,汉族,四川乐山人,博士,正高级工程师。研究方向:智慧交通与人工智能。E-mail:ck@scsz.com。
* **通信作者**: 杨柳(1978—),女,汉族,四川达州人,博士,高级工程师。研究方向:移动通信与工程信息化。E-mail:yangliu@swjtu.edu.cn。

24 h不间断、精确的数据采集,减少人力成本的同时也带来了新问题。平均一座桥梁的传感器监测数目超过 100 个[1],采集频率高,如振动监测可达 50 Hz,位移、风速、风向等监测至少 1 Hz。假设每个传感器采样数据为一种,每数据点时间戳 8 字节,监测值 4 字节,则 50 个 50 Hz 和 50 个 1 Hz 传感器一天数据量可达千兆(GB)级,一年可达太子节(TB)级。在特大桥健康监测或多桥监测系统中,传感器数目众多和高采样频率导致原始数据体量巨大。

当前,针对时间序列数据的压缩算法根据其压缩后解压数据的失真情况进行分类,通常分为无损压缩[2]算法和有损压缩[3]算法。无损压缩指的是压缩前后是可逆的,从压缩结果可以解压恢复出原始数据。有损压缩指的是压缩前后不可逆。在不同的应用场景下,根据应用场景的数据具体特点,使用的压缩算法是不一样的。例如,对于视频和图像等数据,一般使用有损压缩算法[4],因为数据量比较大,而人眼所能感受的信息是有限的,在不影响人眼体验的情况下,可以舍弃掉部分不重要的信息。而在文本、历史数据等场景下,一般使用无损压缩算法。因此桥梁监测场景下应当使用无损压缩算法。

常用的压缩器一般以字节为处理单位,虽然起到了较好的压缩效果,但是没有抓住时序数据的特点,不能有效地去除时序数据的冗余信息。

目前出现了一些时序数据压缩算法。比较早期的是在时序数据库 Gorilla 中使用的 DoD(delta of deltas)和异或(exclusive OR,XOR)算法,这两个算法考虑了时序数据[5]特性,取得了很好的压缩效果,可将存储空间缩小为原来的 1/10。在此基础上,Wollmer 等[6]研究了跨实体增量编码,避免了预编译的共享字典,实现了高度自定义和高效的压缩。Liakos 等[7]基于 XOR 的压缩和数值尾部零的特点,提出了一种名为 Chimp 的新型流式压缩算法,适用于浮点时间序列数据。Chimp 算法仅在尾部零的数量大于□^{264}时使用,节省了空间,有效提高了存储和分析时间序列数据的效率。大多数无损浮点压缩方法都基于异或运算,但没有充分利用尾随零,Li 等[8]提出了一种基于擦除的无损浮点压缩算法,主要思想是擦除浮点值的最后几位,有着更强大的压缩性能。Chen 等[9]提出了一种自适应的无损浮点数压缩算法,该算法针对时间序列数据库中的浮点数数据进行压缩,通过设计 4 种不同的压缩策略,并根据数据模式动态选择合适的策略,以提高压缩比和效率。Yang 等[10]提出了一种有损压缩器 Machete,采用高效的混合编码器,结合 Huffman 编码和可变长度数量(variable length quantity,VLQ)。自适应编码选择使其在短片段数据压缩比方面表现出色,而简单的框架确保了快速的解压缩速度,还发现了 VLQ 存在的局限性。郭亮亮等[11]提出了一种基于开源时序数据库 Open TSDB,结合两级缓存和扇区压缩的高效时序数据库系统,实现了高并发写入和高效存储。Liakos 等[12]通过对当前值和前一个值进行按位异或操作,得到一组结果位。使用两个控制位来区分 4 种可能的编码格式,以最小化表示每个数据点所需的位数。Naser 等[13]提出了一种新的用于大数据压缩的有损数据压缩方法。将最优奇异值分解(singular value decomposition,SVD)应用到一个矩阵中,达到对发送过程的最优奇异值个数,其他的将被忽略。

基于桥梁监测系统中所存在的数据存储[14]问题,针对桥梁监测数据的特点,研究梁监测数据的压缩方案,提出相关的解决思路。首先,实现两种针对桥梁监测数据特点的无损压缩算法:一是桥梁监测数据的时间戳数据具有类似等差数列的特点,将时间戳做二次差分运算后编码,有效降低时间戳数据的存储开销;二是桥梁监测数据的监测值为浮点数序列,对于某一些监测因素的采集数据,它们在一段时间内变化量很小,对其做异或差分运算后编码,可以取得显著的压缩效果。这两种算法的思想来源于 Gorilla 时序数据库,在具体实现上对原有算法进行优化,并将两算法与常用压缩器进行性能测试比较,得出最优的压缩方案选型。该研究通过针对桥梁监测数据特点提出创新的压缩方案,不仅能有效减少存储空间,提升数据传输效率,还能为大规模桥梁健康监测系统中的数据处理提供高效的解决方案,从而支持更精准的桥梁结构健康监测和管理决策。

1　桥梁监测数据的压缩方案

所研究的桥梁为一座单跨钢桁梁悬索桥[15],该桥梁已铺设约 200 个传感器,监测内容为钢桁梁应力、风速风向、环境温湿度、索力、位移以及振动共 8 个类型,传感器按照监测内容分布如表 1 所示。

表 1　传感器类型分布

Table 1　Sensor type distribution

序号	监测内容编码	监测指标	数量/个	采样频率/Hz
1	MonConCode_1	应力	60 ~ 70	25 或 1
2	MonConCode_2	风速	5 ~ 10	1
3	MonConCode_3	风向	5 ~ 10	1
4	MonConCode_4	湿度	5 ~ 10	1
5	MonConCode_5	索力	40 ~ 45	< 1
6	MonConCode_6	温度	5 ~ 10	50 或 1
7	MonConCode_7	位移	5 ~ 10	1
8	MonConCode_8	振动	30 ~ 40	50 或 20

1.1 时间戳序列差量压缩

1.1.1 压缩算法

桥梁监测数据的时间戳数据精度达到毫秒级别,在 UNIX 时间表示法下,时间戳就是一个 64 位的 long(长整型)数值。目前,在压缩该类数据序列时,多采用差值法。因为传感器采集的数据一般是按照指定频率进行的。只有在理想情况下,相邻采集值之间的时间间隔相同,从某一个时间点开始记录,符合标准等差数列的特点,才只需存储第一个值、差值和数据点个数,即可还原原始时间戳序列。

但是在实际的工程应用场景下,传感器采集数据时,会出现抖动、漏点和多采集数据点等异常情况。这些问题会导致时间戳序列并不完全按照等差数列增长,如果需要精确记录下这一情况,则需要采用差值法记录每一个差值的情况,如表 2 所示。

这是一个 5 个长度的时间戳序列,时间戳精度为毫秒级别,D_n为时间戳序列的差值计算结果,计算公式为

$$D_n = T_n - T_{n-1} \tag{1}$$

式(1)中:D_n为第 n 个时间戳的差值;T_n为第 n 个时间戳;T_{n-1}为第 $n-1$ 个时间戳。

第一个值是初始值,不做压缩,直接保存。而后的每个值则可用 32 位 int(整型)变量保存,此时已经起到了压缩的效果,原始 5 个时间戳需要 5 × 64 = 320 bits 存储,现在仅需 64 + 4 × 32 = 192 bits 存储。

如果采用变长编码的方式,对 D_n再细划分,注意 D_n一定为正整数,不可能采集到已经采集过的时间之前的值,则可省去符号位。即用5位表示[0,31],用 6 位表示[0,63]等。此时上表中 40 和 39 的差值可表示为'0b101000'和'0b100111'仅需6 位即可保存,编码总长度为 64 + 6 × 4 = 88 bits 存储,进一步压缩了数据大小。

从计算所得的 D_n序列还可以得到另一个特点,即便传感器采样出现抖动,也是在采样频率上下小幅度变化,在 D_n序列除了第一个值外,其他值的相对差值较小,于是延伸出了时间戳二次差分法,如表 3 所示。

在差值法的基础上对 D_n序列从第 2 个值开始再做一次差分运算,得到 ΔD_n序列。ΔD_n序列中第一个值同 D_n序列,不做压缩,直接保存。而后的每个值使用变长编码,直接存储其二进制表示的最小长度,注意此时需要保留符号位,因为会出现负值。则 ΔD_n序列的第一个值编码结果为64 位,第 2 值 40 编码结果为'0b0101000'7 位,后续的 3 个值,2 个 0 编码结果为 1 位,-1 为 2 位。合计 64 + 7 + 1 × 2 + 2 = 75 bits,比使用差值法 + 变长编码的结果要更小。编码结果以第 1 个值 64 bits 为主,增加总序列的长度,可以看到二次差分法的压缩效果有更好的效果。在这一步骤中,并不能直接将编码结果保存到文件中。因为采用序列的概念,将每个编码值隔开,即将所有的二进制串编码结果连接在一起,以发挥最佳的压缩效果。然而,由于变长编码无法确定每个编码区域的长度,导致解压缩变的异常复杂。因此,需要引入控制位来辅助解码。控制位编码规则如表 4 所示。

控制位采用前缀编码的思想,还有另一种思路是指定控制位为 3 位,因为表 4 只列出了 5 种情况,3 bits 可以表示所有状态。但是在实行效果上不如

表 2 差值法示例

Table 2 Example of difference method

T_n	时间标签	UNIX 时间戳	D_n
T_1	2021/1/2 00:00:00.000	1609516800000	1609516800000
T_2	2021/1/2 00:00:00.040	1609516800040	40
T_3	2021/1/2 00:00:00.080	1609516800080	40
T_4	2021/1/2 00:00:00.120	1609516800120	40
T_5	2021/1/200:00:00.159	1609516800159	39

注:T_n为第 n 个时间戳;D_n为第 n 个时间戳的差值。

表 4 二次差分法控制位编码规则

Table 4 Coding rules for second-order finite difference method

ΔD_n	控制位	编码后长度/bit
0	0	1
[-63,64]	10	2 + 7 = 9
[-255,256]	110	3 + 9 = 12
[-2 047,2 048]	1 110	4 + 12 = 16
>2 048	1 111	4 + 64 = 68

表 3 二次差分法示例

Table 3 Example of the second difference method

T_n	时间标签	UNIX 时间戳	D_n	ΔD_n
T_1	2021/1/2 00:00:00.000	1609516800000	1609516800000	1609516800000
T_2	2021/1/2 00:00:00.040	1609516800040	40	40
T_3	2021/1/2 00:00:00.080	1609516800080	40	0
T_4	2021/1/2 00:00:00.120	1609516800120	40	0
T_5	2021/1/2 00:00:00.159	1609516800159	39	-1

使用前缀编码,原因如下,传感器采集的大部分时间戳数据其间隔是稳定的,则在二次差分法中会出现很多0,即0占ΔD_n序列的大部分,应当优先考虑它的编码结果。如果使用3 bits控制位,则每个0可以被编码为3 bits,而使用前缀编码控制位,每个0可以被编码为1 bit,就直接使用一位控制位0表示0的编码结果即可。假如有90 000个0,使用前缀编码控制位可以少编码90 000×2=180 000 bits,其效果是明显优于定长控制位方案的。

“编码后长度(bit)”的计算结果如第二行“2+7=9”举例,2为控制位长度,7为“1位符号位+ΔD_n二进制表示”,因为ΔD_n处于[-63,64],7位二进制数足够表示除0外-63到64共127个值,需要注意编码结果'0b101000000'在该压缩算法中没有表示任何值,因为0被单独处理了。若为ΔD_n为负数则将其绝对值化后表示为二进制,然后在最前面添加符号位1,再添加控制位10。例如,-40表示为'0b101101000'。若为ΔD_n为正数则将其减1后表示为二进制,这是因为64没法用6位二进制表示,同时'0b100000000'这种情况下不需要用来表示0,所以可以把正数整体减1表示,在解码时再加1即可。然后在最前面添加符号位0,再添加控制位10,如40表示为'0b100100111'。

所使用的二次差分法数据压缩算法,算法核心是对时间戳序列进行二次差分运算,取得的压缩效果十分显著,算法流程图如图1所示。

二次差分法数据压缩算法的核心内容包括数据预处理、数据编码和BitStream处理,具体步骤如下。

步骤1　进行数据预处理,对时间戳序列进行二次差分运算。

步骤2　数据编码,首先将差分结果根据控制位编码规则进行逐个编码。依次将编码结果存储到BitStream中。检查是否为最后一次编码:如果是,触发BitStream的最终保存操作。在最终保存期间,除了一般的保存操作外,额外存储totalLen,即编码结果二进制串的总长度,作为long(长整型)型变量,存入文件的最后8个字节,以备解码需要。

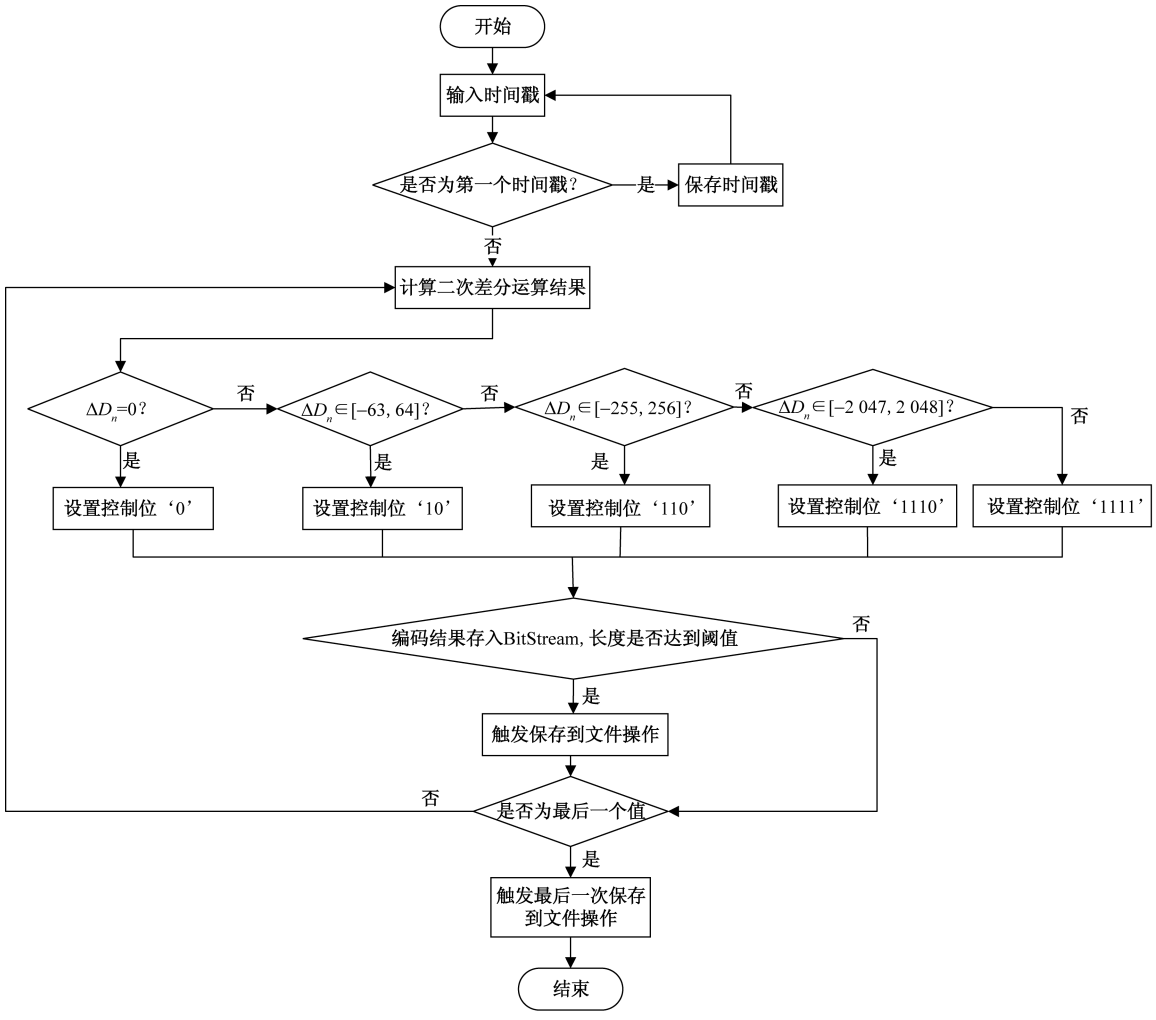

图1　二次差分法数据压缩流程图

Fig. 1　Second-difference method data compression workflow diagram

步骤 3 BitStream 处理,首先将每次接收到编码结果后,将其加到 BitStream 的末尾。然后执行阈值判断,如果比特流达到预定义的阈值,则触发保存到文件的操作。保存到文件的同时,从比特流中删除已保存的部分。如果未达到阈值,则不进行其他操作。

回到上述案例,经过所提的二次差分数据压缩算法处理后,编码结果如表 5 所示。编码长度合计 64 +9 ×2 +1 ×2 =84 bits,压缩效果十分显著。

表 5 二次差分法编码结果

Table 5 Encoding results of the second-order finite difference method

T_n	UNIX 时间戳	D_n	ΔD_n	编码结果
T_1	1609516800000	1609516800000	1609516800000	1609516800000
T_2	1609516800040	40	40	0b100101000
T_3	1609516800080	40	0	0b0
T_4	1609516800120	40	0	0b0
T_5	1609516800159	39	-1	0b101000001

1.1.2 解压算法

介绍时间戳序列差量压缩算法相对应的解压缩算法,该算法实现了对压缩算法的逆运算。通过数据初始化、数值准备、前缀编码分析、控制位编码和二次解差分运算实现压缩算法的执行逆过程,详细步骤如下。

步骤 1 进行数据初始化,对 BitStreamForRead 进行初始化,从需要解压的文件末尾读出压缩二进制串的总长度并赋值给 totalLen,单位为 bit。

步骤 2 数值准备,利用 supply( )函数确保 value 的长度超过 64 位。

步骤 3 前缀编码解析,从 value 中提取前缀编码中的控制位。

步骤 4 控制位解码,根据控制位编码规则,从 value 中提取指定长度的二进制串进行解码,并将解码结果保存到解码序列。检查是否已达到最后一个值,是则执行二次差分运算,否则返回第二步。

步骤 5 二次解差分运算,对解码得到的序列执行二次解差分运算,得到最终的解压结果。

具体的时间戳序列差量解压算法描述如下。

算法 1:时间戳序列差量解压算法

输入:ΔDC_n,ΔDC_n为压缩后的时间戳;
输出:T_n,T_n为第 n 个时间戳;
N 为时间戳总数;
n 为当前压缩时间戳索引,初始化为 1;
ΔD_n为当第 n 个时间戳二次异或结果;
for $n \leqslant N$ do
  if $n = 1$ do
    $T_1 = \Delta DC_1$;
    $n + +$;
    continue;
  end if
  if ΔDC_n的控制位设置为'0' do
    $\Delta D_n = 0$,并计算 T_n;
    保存解压后的时间戳 T_n;
  end if
  if ΔDC_n的控制位设置为'10' do
    ΔDC_n后 7 位为 bit 为 ΔD_n的二进制,
    转为十进制后得到 ΔD_n,并计算 T_n;
    保存解压后的时间戳 T_n;
  end if
  if ΔDC_n的控制位设置为'110' do
    ΔDC_n后 9 位为 bit 为 ΔD_n的二进制,
  转为十进制后得到 ΔD_n,并计算 T_n;
  保存解压后的时间戳 T_n;
  end if
  if ΔDC_n的控制位设置为'1110' do
    ΔDC_n后 12 位为 bit 为 ΔD_n的二进制,
    转为十进制后得到 ΔD_n,并计算 T_n;
    保存解压后的时间戳 T_n;
  end if
  if ΔDC_n的控制位设置为'1111' do
    ΔDC_n后 64 位为 bit 为 ΔD_n的二进制,
    转为十进制后得到 ΔD_n,并计算 T_n;
    保存解压后的时间戳 T_n;
  end if
  $n + +$;
end for

1.2 浮点数异或差分压缩

1.2.1 压缩算法

桥梁监测的数据值为浮点数,采用 float(浮点型)类型数值进行存储。浮点数在 IEEE 标准下的二进制表示分为 1 位符号位、8 位指数位和 23 位有效位,一个 float 类型变量占用 32 位空间与 int 类型变量一致。

异或运算是一种基于二进制的位运算,对提高程序的可读性和运行效率具有重要应用价值,通常记为^或⊕,采用^表示,其运算规则如式(2)所示。

$$0 \wedge 0 = 0;\ 1 \wedge 1 = 0;\ 0 \wedge 1 = 1;\ 1 \wedge 0 = 1 \tag{2}$$

异或运算具有交换律、结合律、自反律和配对律等常见性质,可以广泛的应用在奇偶性判别,纠错检验,网络编码,信息加密,图像处理等应用中。

通过分析桥梁监测数据,如表 6 为位移传感器 MonConCode_7_11_01 采集的部分数据,可以看出,连续的 5 个监测值,在二进制表示法下相似度较高,这是因为原本数据值就相差不大,运用时间戳序列差量压缩的思想,只保留数据间不同的部分,则可以对桥梁监测的数据值序进行压缩,在二进制中,

对两个二进制串做异或运算,实际就保留了两二进制串的不同部分,相同部分则清零。对上述案例的每一个数据值与前一个数据值做异或运算得到异或结果,如表 7 所示。可以看出,如果两个相邻值相同,则后面那个值的异或结果为 0,如果两个值相隔比较近,则异或结果将有很多前导零和后导零,这得益于浮点数的 IEEE 表示法。两浮点数在该表示法下,如果原始数据整数部分相同,则二进制的符号位和指数位也会相同,则异或运算结果至少能让前 9 位清零。

使用统计的方法查看 MonConCode_7_11_01 传感器该日某一小时的数据情况,采样频率为 1 Hz,最小值为 2 442.375 0,最大值为 2 452.125 0,二进制表示及异或结果如表 8 所示。

由表 8 可知,该传感器下相差最大的两个数的异或结果仍然有较多的前导零和后导零,对异或结果做编码则可以达到压缩效果。

编码形式的基本要求为:控制位 + 前导零个数(5 bits) + 有意义区域长度(5 bits) + 有意义区域,如表 9 所示。

控制位采取同时间戳序列差量压缩法相似的前缀编码形式,为了实现最佳的压缩效果,需要确保大多数的 0 被编码为最短的编码。即使 0 的数量不是最多的,但由于将 32 位压缩到 1 位,也是最佳压缩效果的情况之一。

控制位为 0 和 10 的情况更易理解,因为异或结果为 0 时,则可以直接用 1 位'0b0'来表示编码结果,而 10 的编码结果则是还原出原始值的必要条件。有意义区域指的是异或结果去除掉前后导零剩下的部分,该部分和前导零的最大值都是 32 位,故都用 5 bits 来存储。而控制位为 110 则是来源于一种情况,如果相邻的两个异或结果的前导零个数相同,则可以在解压时多维护一个变量,该变量为上一个解压值的前导零个数。该种情况下,在压缩时,可以省去前导零个数这一栏,在解压时通过使用前一个值的前导零个数即可还原原始值。故控制位为 110 时,编码后长度为"8 + 有意义区域长度"。

控制位为 1110 这种情况,是为了防止产生恶化的压缩结果而设定的。因为按照基本编码要求,只要是控制位为 10 情况,则至少需要添加 12 位信息用于解码,而有意义区域可能为 30 位,此时编码结果为 42 位,比不压缩的 32 位值还要多 10 位,则不如不使用压缩。要避免这种情况,需要在控制位 10 和 110 这两种情况前添加一个有关前导零和后导零的和的判断,判断表述如下。

情况 1　当前后零个数与前一个相同时,若前后导零个数小于等于 4 则控制位设为 1110,后面紧跟 32 位原始异或结果值,否则,使用控制位 110 这种情况编码。

情况2　当前后零个数与前一个不同时,若前

表 6　位移传感器部分数据二进制表示

Table 6　Binary representation of partial data from displacement sensor

数据值	二进制表示
2 442.656 2	01000101 00011000 10101010 10000000
2 442.687 5	01000101 00011000 10101011 00000000
2 442.687 5	01000101 00011000 10101011 00000000
2 442.656 2	01000101 00011000 10101010 10000000
2 442.625 0	01000101 00011000 10101010 00000000

表 9　浮点数压缩控制位编码规则

Table 9　Floating-point number compression control bit encoding rules

控制位	编码规则	编码后长度/bit
0	表明异或结果为 0	1
10	按照基本要求编码	12 + 有意义区域长度
110	省去前导零个数	8 + 有意义区域长度
1110	这类数据保持原值	4 + 32

表 7　案例异或运算结果

Table 7　XOR operation result of the case

数据值	二进制表示	异或结果
2 442.656 2	01000101 00011000 10101010 10000000	01000101 00011000 10101010 10000000
2 442.687 5	01000101 00011000 10101011 00000000	00000000 00000000 00000001 10000000
2 442.687 5	01000101 0001100 010101011 00000000	00000000 00000000 00000000 00000000
2 442.656 2	01000101 00011000 10101010 10000000	00000000 00000000 00000001 10000000
2 442.625 0	01000101 00011000 10101010 00000000	00000000 00000000 00000000 10000000

表 8　最小值和最大值二进制表示及异或结果

Table 8　Min and max binary representations and XOR result

数据值	二进制表示	异或结果
2 442.375 0	01000101 00011000 10100110 00000000	01000101 00011000 10100110 00000000
2 452.125 0	01000101 00011001 01000010 00000000	00000000 00000001 11100100 00000000

后导零个数小于等于 8 则控制位设为 1110,后面紧跟 32 位原始异或结果值,否则,使用控制位 10 这种情况编码。

所使用的序列做异或差分运算算法,算法核心是对监测值浮点数序列进行异或差分运算,取得的压缩效果十分显著,算法流程图如图 2 所示。

浮点数异或差分压缩算法的核心内容包括数据预处理、数据编码和 BitStream 处理,具体步骤如下。

步骤 1 进行数据预处理,对监测值浮点数序列进行序列做异或差分运算。

步骤 2 数据编码,首先将差分结果根据控制位编码规则进行逐个编码。依次将编码结果存储到 BitStream 中。检查是否为最后一次编码:如果是,触发 BitStream 的最终保存操作。在最终保存期间,除了一般的保存操作外,额外存储 totalLen,即编码结果二进制串的总长度,作为 long(长整型)型变量,存入文件的最后 8 个字节,以备解码需要。

步骤 3 BitStream 处理,首先将每次接收到编码结果后,将其加到 BitStream 的末尾。然后执行阈值判断,如果比特流达到预定义的阈值,则触发保存到文件的操作。保存到文件的同时,从比特流中删除已保存的部分。如果未达到阈值,则不进行其他操作。

回到上述案例,经过本文浮点数异或差分压缩算法处理后,编码结果如表 10 所示。编码长度合计 $36+14\times2+1+13=78$ bits,相比原始长度 $32\times5=160$ bits,压缩效果得到了进一步提升。

1.2.2 解压算法

介绍基于浮点数序列异或压缩算法相对应的

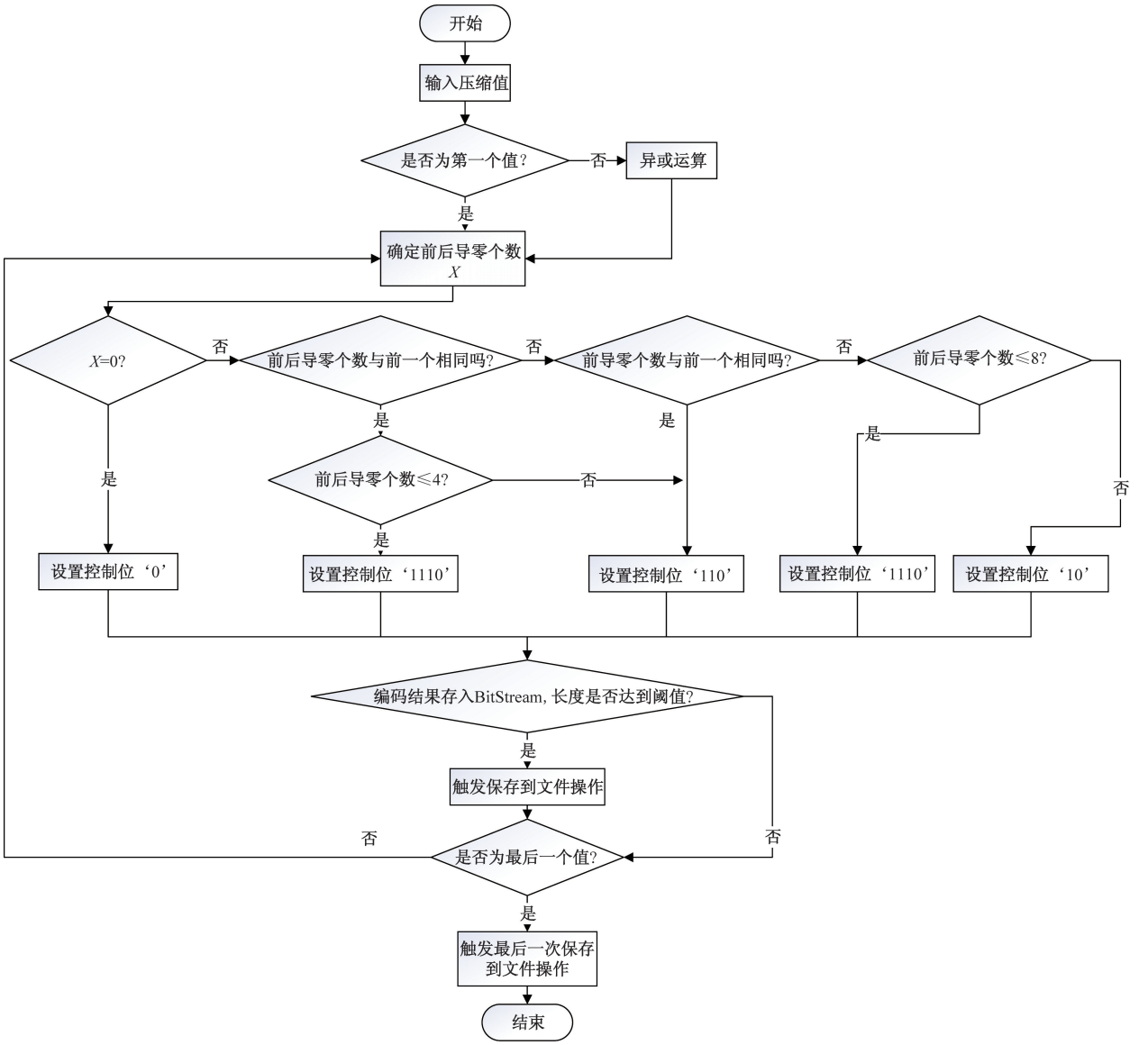

图 2 浮点数异或差分压缩流程图

Fig. 2 Floating-point XOR-difference compression workflow diagram

表 10　浮点数异或差分压缩结果

Table 10　Floating-point XOR-differential compression result

数据值	异或结果	编码结果
2 442.656 2	01000101 00011000 10101010 10000000	0b1110 + 异或结果
2 442.687 5	00000000 00000000 00000001 10000000	0b10101110001011
2 442.687 5	00000000 00000000 00000000 00000000	0b0
2 442.656 2	00000000 00000000 00000001 10000000	0b10101110001011
2 442.625 0	00000000 00000000 00000000 10000000	0b1011000000011

解压缩算法,该算法实现了对压缩算法的逆运算。通过数据初始化、数值准备、前缀编码分析、控制位编码和序列做异或解差分运算实现压缩算法的执行逆过程,详细步骤如下。

步骤 1　进行数据初始化。对 BitStreamForRead 进行初始化,从需要解压的文件末尾读出压缩二进制串的总长度并赋值给 totalLen,单位为 bit。

步骤 2　数值准备。利用 supply( )函数确保 value 的长度超过 64 位。

步骤 3　前缀编码解析。从 value 中提取前缀编码中的控制位。

步骤 4　控制位解码。根据控制位编码规则,从 value 中提取指定长度的二进制串进行解码,并将解码结果保存到解码序列。检查是否已达到最后一个值,是则执行异或解差分运算,否则返回第二步。

步骤 5　序列做异或解差分运算。对解码得到的序列执行序列做异或解差分运算,得到最终的解压结果。

具体的浮点数序列异或解压算法描述如下。

算法 2:浮点数序列异或解压算法

输入:DC_n,DC_n为压缩后的数据值;
输出:D_n,D_n为第 n 个解压后的数据值;
1. N 为数据值总数;
2. n 为当前压缩数据值索引,初始化为 1;
3. DB_n为当第 n 个数据点的异或结果;
4. PreNum 为前一个解压值的前导零个数;
5. for $n \leqslant N$ do
6. 　if $n = 1$ do
7. 　$D_n = DC_n$;
8. 　$n++$;
9. 　continue;
10. 　end if
11. 　if DC_n的控制位设置为'0' do
12. 　$D_n = D_{n-1}$;
13. 　$n++$;
14. 　continue;
15. 　end if
16. 　if DC_n的控制位设置为'10' do
17. 　取 DC_n第 3 ~ 7 位为前导零个数,
18. 　第 8 ~ 12 位为有意义区域长度;
19. 　提取有意义区域,与前导零组合成 DB_n;
20. 　更新 PreNum 为新的前导零个数;
21. 　end if
22. 　if DC_n的控制位设置为'110' do
23. 　取 DC_n第 3 ~ 7 位为有意义区域长度,
24. 　PreNum 作为前导零个数;
25. 　提取有意义区域,与前导零组合成 DB_n;
26. 　end if
27. 　if DC_n的控制位设置为'1110' do
28. 　DC_n后 32 位即为 DB_n;
29. 　end if
30. 　$D_n = D_{n-1} \oplus DB_n$;
31. 　$n++$;
32. end for

2　实验结果和分析

2.1　实验评估指标

算法性能测试分为 3 个部分:一是使用不同的监测内容的传感器数据进行测试,观察两算法在不同桥梁监测数据集上的性能表现;二是将两算法与常用压缩器进行性能比较;三是使用大气压数据集进行测试,观察两算法在非桥梁数据集上的性能效果。

算法性能测试主要使用 3 个指标:压缩率、压缩时间和解压时间。

压缩率的计算公式为

$$c = \frac{b'}{b} \tag{3}$$

式(3)中:b 为压缩前数据大小;b' 为压缩后数据大小,单位相同。

测试环境如表 11 所示。

表 11　测试环境

Table 11　Test environment

硬件环境	软件环境
CPU:1.6 GHz 八核 Intel Core i5-8250U	操作系统:Window 11
内存:8 GB 2 400 MHz DDR4	实验运行环境:Java 1.8

2.2　不同监测内容测试

测试数据集来源为应力、温度和位移传感器某 5 个小时的采集值,每类监测内容把 5 h 平均划分为 5 个 1 h 的测试集,按照“监测内容编号_01 ~ 05”命名。每个采集值分为时间戳和数据值两部分。因为两算法针对的是不同的数据,时

间戳序列差量压缩法负责压缩时间戳,浮点数序列异或压缩法负责压缩数据值,所以在对同一数据集压缩时,两算法压缩的是数据集的不同部分,如表 12 所示。

选择监测内容的基准涉及到本文介绍的浮点数序列异或压缩方法,该方法对数据特征极为敏感,这一特性在分析算法原理时显著凸显。相比之下,时间戳序列差量压缩方法对此并不敏感,因为桥梁监测数据属于时序数据,时序数据的时间戳在不同采样内容下变化基本一致。

从图 3 可以看出,应力 MonConCode_1_05_01 和温度 MonConCode_6_05_01 这两类数据有很多毛刺,表明变化频率大,即数据间前后的变化比较大,这对于浮点数序列异或压缩法在该数据集上的表现是可以预见的。而位移 MonConCode_7_03_01,虽然从整个时间跨度上看也有较大的起伏,但单看一段时间内,毛刺较少,表明变化频率不大,即数据间前后变化不大,大多数值基本在 2 368 ~2 374。

根据表 13 的测试结果,用 MonConCode_7_01_t 表示该数据集的时间戳序列部分,用 MonConCode_7_01_v 表示采样数据值部分,数据点数相同。时间戳序列差量压缩法在不同数据集上表现基本相同,压缩率非常接近,压缩耗时虽然不同,但是考虑到数据集的点数从上到下是依次递增的,表明该压缩法的压缩耗时和解压耗时与数据集点数呈正相关。

根据分析得出,在压缩比的表现上,异或压缩法在变化频率比较小的位移 MonConCode_7 监测数据值上效果较优,压缩率在 0.2 以下,压缩效果明显。而在温度 MonConCode_6 上只有 0.4,在应力 MonConCode_1 上仅有0.9,压缩效果不理想。整体

表 13 时间戳序列差量压缩测试结果

Table 13 Test results of timestamp sequence delta compression

数据集	压缩率	压缩耗时/ms	解压耗时/ms
MonConCode_7_01_t	0.016 2	13	124
MonConCode_1_01_t	0.015 6	118	218
MonConCode_6_01_t	0.015 6	199	335

表 12 位移、应力和温度监测数据集

Table 12 Dataset for displacement, stress and temperature monitoring

数据集	数据点数	数据集	数据点数	数据集	数据点数
MonConCode_7_01	3 600	MonConCode_1_01	89 993	MonConCode_6_01	179 986
MonConCode_7_02	3 600	MonConCode_1_02	89 993	MonConCode_6_02	179 986
MonConCode_7_03	3 600	MonConCode_1_03	89 993	MonConCode_6_03	179 986
MonConCode_7_04	3 600	MonConCode_1_04	89 993	MonConCode_6_04	179 986
MonConCode_7_05	3 600	MonConCode_1_05	89 993	MonConCode_6_05	179 986

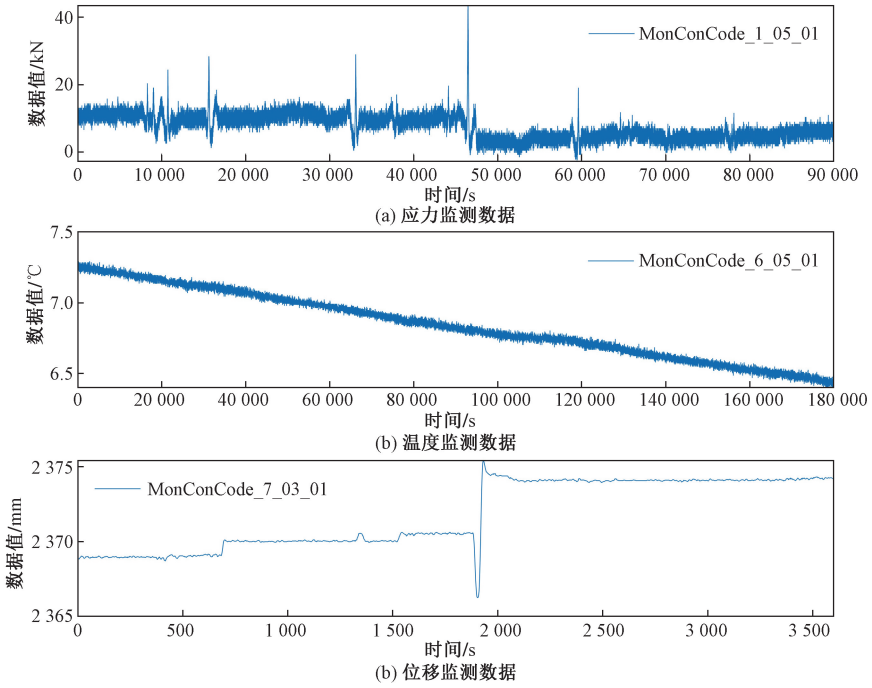

图 3 位移、应力和温度时域特征

Fig. 3 Time-domain features of displacement, stress and temperature

在数据集 01 ~05 上表现平稳,压缩率变化平缓。这表明异或压缩法只适用于相邻值之间变化缓慢,整数部分有较大的值的数据集。另外,当数据集的正负变化也比较频繁,异或压缩法的效果也不理想,这是因为浮点数二进制表示符号位在 0 ~1 交替变换,则异或结果的第一位为 1 的概率较高,导致数据中没有可以压缩的前导零。

如表 14 所示,数据集的点数从上到下是依次递增的,表明该压缩法的压缩耗时和解压耗时与数据集点数呈正相关。

表 14　浮点数序列异或压缩测试结果

Table 14　Test results of floating-point sequence XOR compression

数据集	平均压缩耗时/ms	平均解压耗时/ms	平均压缩率
MonConCode_7_01-05_v	55.0	27.8	0.175 2
MonConCode_1_01-05_v	464.0	432.8	0.910 8
MonConCode_6_01-05_v	582.4	551.4	0.427 9

2.3　不同压缩器性能比较

zlib 是一个封装的函式库,里面有提供数据压缩的接口,因为其改编自 Gzip,所以在使用 zlib 与 Gzip 时,默认是同一种算法,即 DEFLATE[16] 算法。该算法是基于 LZ77(Lempel-Ziv 77)与哈弗曼编码的,LZ77 是经典的字典算法,而哈夫曼编码[17]是经典的基于概率统计模型的编码方式。DEFLATE 的压缩流程为先对文件进行 LZ77 编码,然后把 LZ77 的编码结果再使用哈夫曼编码再压缩一遍得到最终结果,解压则为逆过程。该算法与基于 Burrows-Wheeler变换的 bz2[18] 相比,压缩文件体积上不如 bz2,但压缩解压速度明显优于 bz2,因为 bz2 追求更好的压缩效果,为达到此效果,在压缩时做了更多前期工作。

DEFLATE 在 zlib 中的具体实现使用的是流式编码,即每一个压缩的基本单元为一个字节,这使得它的通用性大大提高,基本可以用于各式各样的文件压缩。

Zstd 是近些年来 Facebook 推出的开源压缩算法,由 LZ77、哈弗曼编码和有限状态熵编码(finite state entropy,FSE)组成[19]。在网络上对于其理论方面的介绍较少,但从效果来看 Zstd 在速度上普遍优于 zlib、snappy 和 bz2 等常用压缩器,并且也能取得不错压缩效果。从其源码中能够大致看出,Zstd 在 LZ77 压缩模块设置了多种不同的压缩方案,是按照压缩率和压缩速度来从高到低划分,而且每次压缩并不都使用上述 3 种算法,而是根据情况来进行搭配。这意味着可以根据具体使用情况的不同,选用不同的压缩模式,如果时间要求不高,则可以选择压缩率表现比较好的模式,如果需要进行快速的压缩解压操作,则可以选择压缩率表现稍差的模式。这点相比其他的常用压缩器是它的优势。此外,该压缩器是基于块来进行的压缩,可以用于对二进制文件压缩[20]。

选择常用压缩器 Gzip、Zstd 和 snappy 用来性能比较。下面的测试结果中,时间戳序列差量压缩法用 DoD 缩写表示,浮点数异或压缩法用 XOR 表示,此外,还多对比了一种先使用本文 XOR 压缩,然后把压缩结果再通过 Gzip 压缩一遍的混合方法,此方法名称缩写为 MIX。

数据集选用位移采集数据,数据点数 69 825 个,如表 15 所示,同样的,每个数据点分为时间戳和数据值。例如,用 MonConCode_7_t 表示位移时间戳序列,MonConCode_7_v 表示位移数据值序列。

如表 16 所示,时间戳差量压缩法(DoD)在压缩率上与其他压缩器相比有很大的优势,因为数据集中的抖动和漏点情况比较少,则大部分的数值都被压缩为一个比特。理想情况假若所有值都被压缩为一个比特,数据量大时,可忽略第一个值不压缩的影响,则压缩率的极限值为 1/64 = 0.015 625,即时间戳原需要 64 bit 存储,压缩后为 1 bit。测试结果已经非常接近极限值,表明数据集中的时间戳序列非常接近等差数列。而在压缩耗时和解压耗时上,此压缩法在 4 个压缩器中也位居中上游。这表明在压缩时间戳序列数据时,差量压缩法是最好的选择。

如表 17 所示,在压缩率上,所实现的 XOR 算法明显优于 snappy,与 Gzip 和 Zstd 相当,并略优于它们。在压缩耗时上,Gzip 耗时最短,压缩速度最快,XOR 仅低于 Gzip,排行第二。在解压耗时上,XOR 仅优于snappy,解压速度较慢。而混合MIX方法的

表 15　数据集情况

Table 15　Dataset overview

数据集	数据点数	备注
MonConCode_7_t	69 825	位移时间戳序列
MonConCode_7_v	69 825	位移数据值序列

表 16　时间戳差值法在桥梁数据集上的性能比较

Table 16　Performance comparison of timestamp delta method on bridge dataset

测试指标	Gzip	snappy	Zstd	DoD
压缩率	0.349 4	0.561 7	0.232 5	**0.015 6**
压缩耗时/ms	65	313	491	**64**
解压耗时/ms	4	873	**3**	70

注:加粗文字为每组测试结果的最优值。

表 17 浮点数异或法在桥梁数据集上的性能比较

Table 17 Performance comparison of floating-point XOR method on bridge dataset

测试指标	Gzip	snappy	Zstd	XOR	MIX
压缩率	0.327 3	0.526 1	0.319 2	0.302 8	**0.202 7**
压缩耗时/ms	**53**	253	592	136	148
解压耗时/ms	12	388	**5**	238	246

注:加粗文字为每组测试结果的最优值。

压缩率,比单使用 Gzip 和单使用 XOR 都要好,而且非常明显,降低了 10 个百分点。理论上压缩解压耗时应为两个算法单独使用的和,实际测试结果比预期要耗时更短,仅略高于单使用 XOR。

对于变化量不大的位移数据,XOR 在压缩率上优于常用压缩器,压缩速度也比较快,但在解压耗时上表现不佳。而混合 MIX 方法的测试结果表明,本文所实现的 XOR 算法与 Gzip 组合可以取得明显更好的压缩率表现。

2.4 大气压数据集实验

非桥梁数据集采用的是大气压数据集,数据集中的数据点在 2020 年每 10 min 记录一次,总共 52 697条监测数据。

从图 4 可以看出,大气压数据值从整个时间范围上观察有较大的起伏。但整体数值基本在 980 ~ 1 020,表明数据值变化频率较小,相邻时间的数据值变化不大。

每组测试结果的最优值已加粗显示,如表 18 所示。在大气压数据集上,时间戳差量压缩法(DoD)在压缩率上与其他压缩器相比仍然有很大的优势。但在压缩耗时上,Gzip 的耗时更短。解压耗时上,Gzip 和 Zstd 的性能表现要优于 DoD。DoD 的压缩率达到 0.044 8,远优于其他压缩器的压缩率。

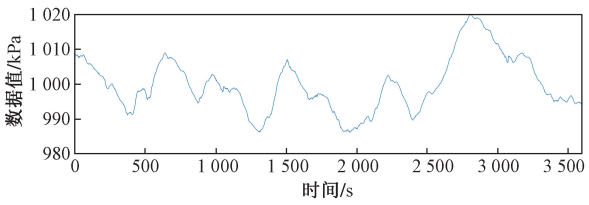

图 4 大气压数据集特征

Fig. 4 Characteristics of atmospheric pressure dataset

表 18 时间戳差值法在非桥梁数据集上的性能比较

Table 18 Performance comparison of timestamp delta method on non-bridge dataset

测试指标	Gzip	snappy	Zstd	DoD
压缩率	0.293 0	0.617 1	0.071 2	**0.044 8**
压缩耗时/ms	**22**	187	1 190	65
解压耗时/ms	**3**	1 098	4	55

注:加粗文字为每组测试结果的最优值。

实验结果表明,即使不是桥梁监测数据的时间戳序列,但只要符合等差数列特点的时间戳数据集,差量压缩法都有着很好的压缩效果。

如表 19 所示,在压缩率上,Zstd 算法的压缩率明显优于其他算法,压缩效果最好。在压缩耗时上,Gzip 耗时最短,压缩速度最快,但本文 XOR 算法的压缩耗时是 Gzip 的 1.78 倍,效果一般。在解压耗时上,Gzip 的解压耗时最短,解压效果最好,而 XOR 的解压耗时达 130 ms,解压效果一般。而混合 MIX 方法的压缩率,比单使用 XOR 要好,而且非常明显,降低了 12 个百分点。但是不如 Gzip 的压缩率。理论上压缩解压耗时应为两个算法单独使用的和,实际测试结果比预期要耗时更短,仅略高于单使用 XOR。

表 19 浮点数异或法在非桥梁监测数据上的性能比较结果

Table 19 Performance comparison of floating-point XOR method on non-bridge dataset

测试指标	Gzip	snappy	Zstd	XOR	MIX
压缩率	0.456 7	0.706 8	**0.319 2**	0.662 8	0.546 1
压缩耗时/ms	**32**	127	592	65	57
解压耗时/ms	**4**	522	5	130	143

注:加粗文字为每组测试结果的最优值。

在非桥梁监测数据上,XOR 的压缩率、压缩耗时和解压缩耗时不理想。而混合后的 MIX 在压缩率和压缩耗时上相比于 XOR 有着较小的提升,但 XOR 在解压耗时上有着更优的性能。这表明所实现的算法在大气压数据集上的效果相较于桥梁监测数据集表现一般,说明所提出浮点数异或压缩算法更适合于桥梁监测类型数据。

3 结论

针对桥梁海量监测数据的存储问题,基于桥梁监测数据的时间戳数据具有类似等差数列的特点,提出了时间戳序列差量压缩法。基于桥梁监测数据的监测值为浮点数序列,提出浮点数序列异或压缩算法。得出如下结论。

(1)实验表明,两算法对比常用压缩器有不同程度的优势,时间戳序列差量压缩法在压缩率上优于常用压缩器,在符合等差数列特性的时间戳序列数据集上,压缩率 0.0 156,接近压缩极限值,压缩解压速度位居中上。DoD 在大气压数据集上的压缩率为 0.044 8,表明 DoD 对数据的监测类型不敏感。而异或压缩法在变化频率不大的桥梁监测类型数据集上表现比较好,压缩率为 0.302 8。但 XOR 在非桥梁数据集上的压缩率为 0.662 8,表明 XOR 对

数据监测类型敏感。两算法均采用差量压缩的思想,以消除相邻数据间的相同部分和保留差异部分达到压缩效果。

(2)所实现的桥梁监测数据压缩方案中,对于监测值浮点数的压缩算法在使用场景上具有局限性,不能应对数据值剧烈变化的场景,并且在解压速度上相比常用压缩器比较慢。后续尝试在这方面继续深入探究,提高算法的综合性能。

参 考 文 献

[1] 郭凡, 伍衡山. 桥梁传感器优化布置的研究现状与发展趋势[J]. 科技创新与应用, 2024, 14(7): 107-110.
Guo Fan, Wu Hengshan. Research status and development trend of optimized layout of bridge sensors[J]. Science and Technology Innovation and Application, 2024, 14 (7): 107-110.

[2] lakkiya S, Thivya K S. Comprehensive review on lossy and lossless compression techniques[J]. Journal of the Institution of Engineers (India): Series B, 2021, 103(3): 1-10.

[3] Yang Y, Mandt S, Theis L. An introduction to neural data compression[J]. Foundations and Trends in Computer Graphics and Vision, 2023, 15(2): 113-200.

[4] Naaman W D. Image compression technique based on fractal image compression using neural network: a review[J]. Asian Journal of Research in Computer Science, 2021, 22: 47-57.

[5] Hao C, Hao R, Zhao H, et al. Identification and validation of sepsis subphenotypes using time-series data[J]. Heliyon, 2024, 10 (7): DOI: 10.1016/j. heliyon. 2024. e28520.

[6] Wollmer B, Wingerath W, Ferrlein S, et al. The case for cross-entity delta encoding in web compression[C]//International Conference on Web Engineering. Cham: Springer International Publishing, 2022: 177-185.

[7] Liakos P, Papakonstantinopoulou K, Kotidis Y. CHimp: efficient lossless compression of floating point time series data[J]. Journal of Visualization, 2023, 18: 147-157.

[8] Li R, Li Z, Wu Y, et al. ELF: erasing-based lossless floating-point compression[J]. Proceedings of the VLDB Endowment, 2023, 16(7): 1763-1776.

[9] Chen H, Liu L, Meng J, et al. AFC: an adaptive lossless floating-point compression algorithm in time series database[J]. Information Sciences, Elsevier BV, 2024, 654: 119847.

[10] Yang S, Zou X, Chen X, et al. Machete: an efficient lossy floating-point compressor designed for time series databases[C]//2024 Data Compression Conference (DCC). Snowbird, UT: IEEE, 2024: DOI: 10.1109/DCC58796.2024.00061.

[11] 郭亮亮, 靳燕, 杨博, 等. 一种基于两级缓存的高效时序数据库系统[J]. 测试技术学报, 2022, 36(2): 147-152.
Guo Liangliang, Jin Yan, Yang Bo, et al. An efficient temporal database system based on two-level caching[J]. Journal of Test & Measurement Technology, 2022, 36(2): 147-152.

[12] Liakos P, Papakonstantinopoulou K, Bruineman T, et al. How to make your duck fly: advanced floating point compression to the rescue[J]. Springer-Verlag, 2024, 11: DOI: 10.1007/11760191_195.

[13] Naser H, Jamshid A, Abdollah F K, et al. Optimal singular value decomposition based big data compression approach in smart grids [J]. IEEE Transactions on Industry Applications, 2021, 57(4): 3296-3305.

[14] 辜晗光, 刘洋, 刘振丘, 等. 基于TDengine的桥梁群监测大数据平台研究[J]. 公路, 2023, 68(2): 396-401.
Gu Hanguang, Liu Yang, Liu Zhenqiu, et al. Research on TDengine based big data platform for bridge group monitoring[J]. Highway, 2023, 68(2): 396-401.

[15] 朱嘉煜, 乔文开, 王康宇. 超大跨度单跨悬索桥钢桁梁制作与安装研究[J]. 建材与装饰, 2023, 19(8): 129-131.
Zhu Jiayu, Qiao Wenkai, Wang Kangyu. Research on fabrication and installation of steel truss girder for extra-large span single-span cable-stayed bridge [J]. Building Materials and Decoration, 2023, 19(8): 129-131.

[16] 李文清, 高平, 李光松. 基于数据分析的DEFLATE算法特征研究[J]. 信息工程大学学报, 2021, 22(1): 74-80.
Li Wenqing, Gao Ping, Li Guangsong. Research on features of the DEFLATE algorithm based on data analysis[J]. Journal of Information Engineering University, 2021, 22(1): 74-80.

[17] Klein S T, Saadia S, Shapira D. Forward looking huffman coding [J]. Theory of Computing Systems, 2021, 65(3): 593-612.

[18] Begum M B, Deepa N, Uddin M, et al. An efficient and secure compression technique fordata protection using burrows-wheeler transform algorithm[J]. Heliyon, 2023, 9(6): DOI: 10.1016/j. heliyon. 2023. e17602.

[19] Chen W, Elliott L T. Compression for population genetic data through finite-state entropy[J]. Journal of Bioinformatics and Computational Biology, 2021, 19(5): 2150026.

[20] Hardi S M, Zarlis M, Lubis D R P, et al. Text file compression using hybrid run length encoding (RLE) algorithm with even rodeh code (ERC) and variable length binary encoding (VLBE) to save storage space[J]. Journal of Physics: Conference Series. IOP Publishing, 2021, 1830(1): 012022.