一种高维时序数据线性压缩方法

石剑君[1]，陈娟[2]，高玉金[1]，邹天刚[2]，XXX[2]，计卫星[1]

（1.北京理工大学计算机学院，北京 100081；
2.中国中国兵器工业第 201 研究所，北京 XXXXXX）

摘 要：提出了一种高维时序数据线性压缩算法。该算法根据时序数据的维数与每维数据压缩后的数据点数，对输入的时序数据进行线性扫描。在扫描的过程中，采用区间合并的方法对每一分量数据点进行合并。将合并后的所有分量区间的时间标签提取出来，进行全局排序，并根据排序后的时间标签重新生成时序数据，最后将生成的时序数据存入关系数据库中。该算法可应用于各类工业控制系统高维时序数据的压缩存储，较好地保留原数据的特征，计算复杂度低，压缩比高，压缩后的结果可以在关系数据库中存储并使用 SQL 语句进行查询。

关键词：高维数据；数据压缩；时序数据
中图分类号：V211 **文献标识码**：A

A Linear Compression Method for High Dimension Time Series Data

SHI Jian-jun[1], CHEN Juan[2], xxx, GAO Yu-jin, JI Wei-xing
(1. School of Computer Science, Beijing Institute of Technology, Beijing 100081, China; 2. XXXX, Beijing
XXXX, Beijing 100081,China)

Abstract: A Linear Compression and Storage Method For Transmission Device. According to the dimension of time series data and the data points of each dimension, linear scan is performed on the input data. In the process of scanning , the interval merging method merges each component data points. Then, the time labels of all the components are extracted for global ranking, and the time series data are generated by the time labels. Finally, the time sequence data is stored into a relational database. The method proposed in this paper can be applied to all kinds of industrial control system of data storage. It also remain the feature of original data with high real time performance, high compression ratio and strong practicality. The most important of all is all processed data can be stored into a relational database.

Key word: Transmission Device; Linear Compression Storage; Time Series Data

1 引言

时序数据是按时间节点进行记录的数据集合。与非时序数据相比，时序数据的顺序性要

基金项目：XXX 项目资助（XXXXX）（**请 201 提供，是否需要放入相关课题资助**）
作者简介：XXX（1963—），男，教授，工学博士，E-mail: abcd@sina.com

求更高。时序数据通常产生于实际生产生活实践，例如从天气、股票交易和汽车传动装置采集的数据等都属于时序数据[1]。目前随着时序数据数据量的不断增大，海量时序数据在关系数据库的存储和查询面临巨大挑战。由于时序数据的在一段时间内重复数据的比例较高，造成大量存储空间的浪费。高维时序数据是时序数据在空间概念上的体现。高维时序数据通常产生于数学或生产模型中，如传动装置的试验数据，包含了档位、方向、发动机等相关 40 多个参数的试验时序数据。相比于一维时序数据，高维时序数据的复杂性更高，数据量更大。

数据压缩是节约存储空间、提高数据传输效率的有效手段。随着云计算和物联网的不断发展，数据的压缩存储技术成为海量数据研究的重要内容。数据压缩主要分为无损压缩和有损压缩[2-3]。无损压缩是指数据压缩后能恢复到压缩前的最初数据。即整个压缩过程，并没有数据损失。比较典型的例子是计算机磁盘文件的压缩。无损压缩通常可以把数据压缩到原数据的 1/2~1/4。有损压缩是指在不影响数据正常应用的情况下，将原有数据以缩减的方式进行压缩。有损压缩可以以更高的压缩比对数据进行压缩，但有损压缩后的数据并不能恢复到数据的最初状态，因此，有损压缩是一个不可逆过程。有损压缩通常应用于数据要求不高的场合，比如视频和音频的压缩可以采用有损压缩。

相比于一维时序数据的压缩，高维时序数据的压缩更为复杂，然而实际应用中高维时序数据的是非常常见的，因此，研究出一种针对高维数据压缩存储的方法显得尤为重要[4-6]。目前虽然存在多种压缩方法，例如旋转门压缩法、稳态阈值法、线性外插法等[7-12]，但是大多数都是一维数据的有损压缩方法，不能直接对二维、三维或者更多维数据进行压缩；其次，压缩算法复杂性高，大批量数据的处理需要较长的时间和计算资源；更为重要的一点，压缩后的数据无法直接在关系数据库中存储和查询，不利于已有业务系统使用。因此，本文提出了一种高维时序数据压缩存储方法，旨在解决现有压缩方法无法高效对高维数据进行压缩及在已有关系数据库中存储的问题。

2 算法描述

2.1 问题描述

为了实现状态监测、故障预警和诊断，车辆传动装置内部安装了大量传感器。在传动装置车载试验过程中，这些传感器采集的数据将持续不断的发送到上位机电脑中。以车辆转向加速试验为例，采集到的数据包括档位及方向、发动机转速、发动机回水温度、风扇转速等多达 40 个参数变量。以每秒 10 次的采样率计算，装置持续运转一个小时会收集到 72000 条数据记录。在装置长时间运转过程中，会生成总量巨大的数据，如果简单的将所有的数据保存下来，将占用大量的物理存储空间，且难以检索和使用。

实际观察采样数据会发现，大量的试验数据中存在较多的冗余数据，这是因为设备在运转过程中，某一部件的局部状态经常处于一个稳定状态。例如在高频采样的数据中，“档位及方向”在一段时间内会出现大量的重复数据。从业务角度考虑，工程人员很难从高维海量的时序数据中观察到有用的信息，业务人员更多希望看到数据所展现出来的趋势，以及系统运行过程中是否存在异常数据。因此实际应用中，采集的数据保存时允许有一定的误差。另外，目前众多的应用系统基于关系数据库构建，考虑到使用习惯、迁移成本等问题，目前还无法在短期内实现从关系数据库到海量存储系统的改造工作。因此，如何基于关系数据库实现海量高维时序数据的压缩存储是目前急需解决的问题之一。

本文提出的高维时序数据线性压缩存储方法是基于高维时序数据顺序性、局部性的特性，将高维时序数据中大量冗余数据剔除，同时又保留数据本身变化趋势、基本特性的数据压缩方法。

2.2 基本思路

首先根据时序数据的维数和每维数据压缩后的数据点数，对输入的时序数据进行线性扫

描。在扫描的过程中，对每一分量分别进行处理，将无明显拐点的相邻数据点合并到同一个区间中，当每个分量的区间总数超出用户设定的区间总数限制时，采用区间合并的方法对每一分量已有区间逐步进行合并。最后，将合并后的所有分量区间的开始时间和结束时间标签提取出来，进行全局排序，并根据排序后的时间标签重新生成时序数据，并将生成的时序数据存入关系数据库中。

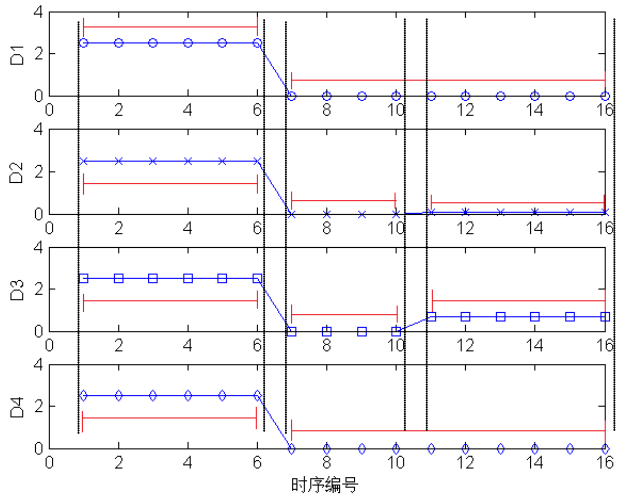


图 1 原始时序数据 D1-D4

如图 1 所示，D_1、D_2、D_3和 D_4为同一系统采集的时序数据，其中 D_1从 t_1到 t_6的值未发生变化，t_7到 t_{16}区间内的值相同，因此将 D_1划分为两个区间[1, 6]和[7, 16]（如红色所示），D_4与 D_1类似；按照同样的方法，D_2和 D_3被划分为 3 个区间[1, 6]、[7, 10]和[11, 16]。将 4 个分量所有区间的所有开始和结束标签提取出来，经过去重和排序之后获得的时序列表为[1, 6, 7, 10, 11, 16]（如图中虚线所示）。重新生成压缩后的数据时，针对时序列表中的 6 个时刻点生成 6 条数据记录，每个时刻某一数据分量的值为对应时刻所属数据区间的平均值。最后生成的数据序列如下表 1 所示：

表 1 压缩后时序数据 D_1-D_4

T	D_1	D_2	D_3	D_4
1	2.5	2.5	2.5	2.5
6	2.5	2.5	2.5	2.5
7	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0
11	0.0	0.1	0.7	0.0
16	0.0	0.1	0.7	0.0

2.3 数据区间及相关操作

设 D 维时序数据每一分量 D_i 压缩后数据点数为 $CP(D_i)$，并将压缩梯度 $Grad(D_i)$设置为 $CP(D_i)\times 2^k (k=1, 2, 3,...)$。定义 $Max(D_i)$为 D_i 已经扫描数据的最大值，$Min(D_i)$为 D_i 已经扫描数据的最小值，$IL(D_i)$用于存储 D_i 已经扫描过的数据的区间列表，列表的每一个区间是一个元组 T，

$$T= <t_s, t_e, v_{max}, v_{min}, v_{avg}, n> \quad (1)$$

其中，t_s 和 t_e 分别表示区间的开始时间标签和结束时间标签，v_{max}，v_{min} 和 v_{avg} 分别表示区间内数据点的最大值、最小值和平均值，n 表示区间内数据点的个数。

每个区间是 1 个以上的原始数据点合并之后的结果，同一个区间覆盖的原始数据点波动

较小，相邻区间的结束点和开始点应该是原始数据中的明显拐点。压缩过程实际上是在用户指定的输入条件 $CP(D_i)$的约束下，不断对 D_i 创建区间、扩展区间和合并区间的过程。

新建区间 CI：高维时序数据压缩须保持原数据总体变化趋势等基本特性，在数据处理的过程中，当遇到较大的数据拐点且不在同一个压缩梯度等级时，应创建新的区间段，新的数据区间取决于拐点数据，即：

$$I_n=<t, t, d_{it}, d_{it}, d_{it}, 1> \quad (2)$$

其中，d_{it} 为拐点数据，即当前时刻 t 第 i 维数据的值。

扩展区间 EI：在扫描数据序列的过程中，若当前数据点 d_{it} 属于当前的数据区间，则应对当前区间进行更新，区间扩展方法为：

$$I_{1ast}=<t_s, t, MAX(v_{max}, d_{it}), MIN(v_{min}, d_{it}), (v_{avg}*n+d_{it})/(n+1), n+1> \quad (3)$$

其中，I_{last} 为 $IL(D_i)$区间的最后一个元组，d_{it} 为当前时刻 t 关于 D_i 的分量。

区间合并 MI：区间合并发生在已有区间总数超出了设定的数据点数 $CP(D_i)$时进行，区间合并是指将原来的两个相邻的区间合并为一个区间。设 $I_i=<t_s, t_e, v_{max}, v_{min}, v_{avg}, n>$和 $I_{i+1}=<t'_s, t'_e, v'_{max}, v'_{min}, v'_{avg}, n'>$为数据区间列表 $IL(D_i)$中的相邻元组，则合并后的数据区间 I_{lr} 定义为：

$$I_{lr}=<MIN(t_s, t'_s), MAX(t_e, t'_e), MAX(v_{max}, v'_{max}), MIN(v_{min}, v'_{min}), (v_{avg}\times n+ v'_{avg}\times n')/(n+n'), n+n'> \quad (4)$$

2.4 数据压缩算法

在对整个数据序列进行压缩的过程中，顺序扫描每一行数据，分别对每一分量进行区间压缩(CompressInterval 算法)，得到所有不同分量的区间列表序列 $IL=[IL(D_1), IL(D_2), \ldots, IL(D_N)]$。在区间创建和扩展过程中，对每个分量 D_i 判断 $IL(D_i)$中区间个数与 $CP(D_i)$的大小，如果$|IL(D_i)|>CP(D_i)$，则按照区间合并操作 MI 选择性地对相邻区间进行合并，这里的选择性是指选取那些数据分布较为接近和相似的相邻区间进行合并。最后，取 IL 中所有 $IL(D_i)$中的元组时间标签，并按时间先后排序后，放入一时间标签队列 TL 中，最终重新生成分量数据。算法具体的描述如下所示：

算法 1: 高维时序数据压缩算法 CompressInterval

输入：　分量压缩点数 $CP(D_i)$；梯度系数 k；原始数据序列 D

输出：　压缩后的数据序列

```
1   FOR i =1 to N DO
2       Max(Di) = 0; Min(Di) = 0; IL(Di) = [];
3       Grad(Di) = CP(Di)*k;
4   END FOR
5   WHILE (未到达输入 D 结束位置且下一条数据记录为 Dt) DO
6       FOR i =1 to N DO
7           Max(Di) = MAX( Max(Di), dit);
8           Min(Di) = MIN(Min(Di), dit );
9           Ilast(Di) = IL(Di)的最后一个区间;
10          IF( dit >= Ilast(Di).vmin AND dit <= Ilast(Di).vmax)
11              Ilast = EI(Ilast, dit);
12          ELSE    IL(Di).push(EC(dit));
13          END IF
14          IF(|IL(Di)| > CP(Di))
15              IL(Di) =MergeInterval (IL(Di), CP(Di), Grad(Di));
```

```
16          END IF
17      END FOR
18  END WHILE
19  IL=[IL(D1), IL(D2), ..., IL(Dn)];
20  RETURN   GenData(IL);
```

压缩算法首先对每一分量的相关数据进行初始化，然后顺序读入每条数据记录，对每个分量数据，更新最大值和最小值。接着判断当前的数据 d_{it} 是否在最后一个区间，如果不属于最后一个区间，则创建一个区间；否则直接对最后一个数据区间进行扩展。处理完每条记录之后，都需要判断当前的区间总数是否超出了用户设定的 $CP(D_i)$，如果超出则调用 *MergeInterval* 对相邻区间进行合并。算法最后根据 IL 调用 *GenData* 重新生成数据。

算法 2:区间合并算法 IntervalMerge

输入：　*合并前的区间列表 $IL(D_i)$，$CP(D_i)$, $Grad(D_i)$*

输出：　合并后的区间列表 $IL(D_i)$

```
1   R(Di) = (Max(Di)-Min(Di))/Grad(Di);
2   FOR i =1 to |IL(Di)| - 1 DO
3       IF (∃ x <Grad(Di) AND   Ii.vmin >= Min(Di) + x*R(Di) AND Ii.vmax <
    Min(Di) +( x+1)*R(Di)   AND Ii+1.vmin >= Min(Di) + x*R(Di) AND Ii+1.vmax <
    Min(Di) +( x+1)*R(Di) )
4           Ii = MI(Ii, Ii+1);  将 Ii+1 从 IL(Di) 中删除; i--;
5       END IF
6   END FOR
7   WHILE(|IL(Di)| > CP(Di)) DO
8       Grad(Di) = Grad(Di) / 2;
9       R(Di)=(Max(Di)-Min(Di))/Grad(Di);
10      FOR i =1 to |IL(Di)| - 1 DO
11          IF (∃ x <Grad(Di) AND   Ii.vmin >= Min(Di) + x*R(Di) AND Ii.vmax <
12  Min(Di) +( x+1)*R(Di)   AND Ii+1.vmin >= Min(Di) + x*R(Di) AND Ii+1.vmax <
13  Min(Di) +( x+1)*R(Di))
              Ii=MI(Ii, Ii+1);将 Ii+1 从 RL(Di) 中删除;   i--;
            END IF
14      END FOR
15  END WHILE
16  RETURN IL(Di);
```

区间合并算法首先判断在现有的梯度等级划分之下，是否可以对 D_i 的已有数据区间进行合并，如果合并完成 $IL(D_i)$中的区间总数仍然是大于 $CP(D_i))$，则降低梯度等级数量，从而将更多的数据区间进行合并，循环该过程直到数据区间的总数满足条件。

算法 3:数据生成算法 GenData

输入：　*合并后的区间列表序列 IL*

输出：　压缩后的数据

```
1   TL=[ ];
2   FOR i = 1 to N
3       FOR j =1 to |IL(Di)| DO
4           TL.push(IL(Di)j.ts); TL.push(IL(Di)j.te);
```

```
    END FOR
END FOR
对 TL 中的时间标签从小到大排序
删除 TL 中相邻冗余的时间标签
FOR i = 1 to |TL|
    FOR j =1 to |IL(Di)| DO
        输出 TLi 所属的 IL(Di)区间的平均值;
    END FOR
    输出换行符;
END FOR
```

数据重新生成算法首先将所有分量的所有区间的时间标签放入到一个列表中，经过排序和去重之后，再遍历 *IL* 生成新的数据。值得注意的是，新生成的数据是区间内所有数据点的平均值。

以图 1 中数据作为输入数据，采用上述高维数据压缩存储算法进行数据压缩。首先设置 4 维时序数据压缩点数和压缩梯度如下表 2 所示：

表 2 设置数据压缩点数和压缩梯度

参数设置	D_1	D_2	D_3	D_4
$CP(D_i)$	2	4	4	4
$Grad(D_i)$	4	8	8	8

压缩点数的设置与用户的经验相关，如果用户已知某个分量的数据稳定性比较好，且对应用的使用产生的影响较小，则可以减少压缩点数，反之增加压缩点数。压缩点数的设置会影响最后压缩后生成的时序数据总量。初始时，将 D_i 的最大值 $Max(D_i)$和最小值 $Min(D_i)$分别设置为第一个数据点的值；例如 $Max(D_2)=Min(D_2)=0.25$。为每一分量 D_i 初始化一个区间列表 $IL(D_i)$，例如初始 $IL(D_2)=\{\}$。顺序读入时序数据，对任意时刻 t 采样到的 4 维时序数据，分别对每一分量 d_{it} 进行处理，采用区段合并的方法将每个分量数据压缩为一个区段列表 $IL(D_i)$。以表 1 中分量 D_2 为例，$Grad(DC1)=8$。开始时 $R(D_2)=Max(D_2)-Min(D_2)=0$，将 $R(D_2)$ 分为 8 个区间，每个区间 R_j 的跨度大小为 0。顺序读入数据，按照算法执行过程，如下表 3 所示：

表 3 高维数据线性压缩算法执行过程

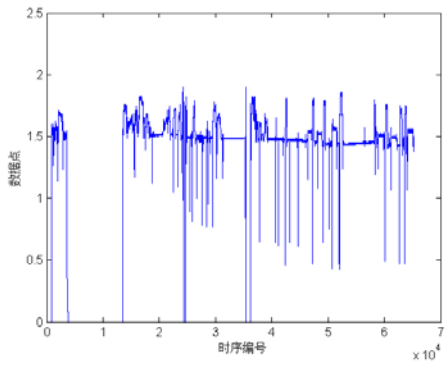
时序	$Max(D_2)$	$Min(D_2)$	$R(D_2)$	$IL(D_2)$
1	0.25	0.25	0.00	[<1, 1, 0.25, 0.25, 0.25, 1>]
…	…	…	…	…
6	0.25	0.25	0.00	[<1, 6, 0.25, 0.25, 0.25, 6>]
7	0.25	0.00	0.03125	[<1, 6, 0.25, 0.25, 0.25, 6>, <7, 7, 0, 0, 0, 1>]
…	…	…	…	…
10	0.25	0.00	0.03125	[<1, 6, 0.25, 0.25, 0.25, 6>, <7, 10, 0, 0, 0, 4>]
11	0.25	0.00	0.03125	[<1, 6, 0.25, 0.25, 0.25, 6>, <7, 10, 0, 0, 0, 4>, <11, 11, 0.1, 0.1, 0.1, 1>]
…	…	…	…	…
16	0.25	0.00	0.03125	[<1, 6, 0.25, 0.25, 0.25, 6>, <7, 10, 0, 0, 0, 4>, <11, 16, 0.1, 0.1, 0.1, 6>]

经过压缩后的数据量只占原有数据量的 20%左右，去掉了过多的冗余数据，同时保留了高维数据的分量特性、变化趋势等基本特性。

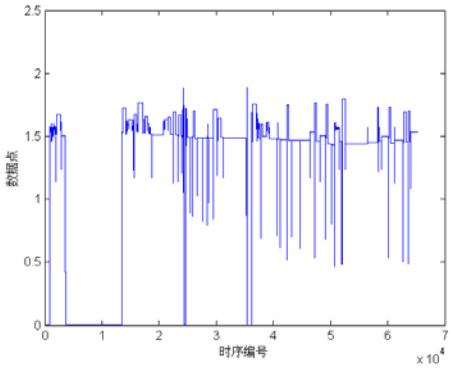
3 性能分析

3.1 压缩效果分析

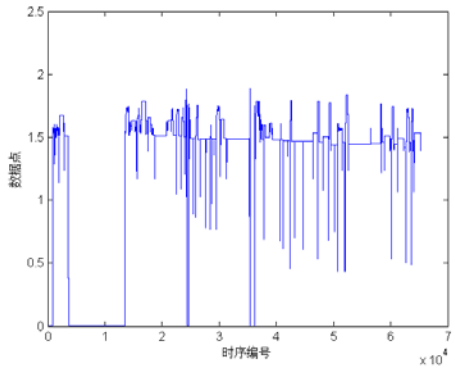
为了对比设置不同压缩点数对压缩性能的影响，图 2 中(b)、(c)和(d)分别显示了压缩点数设为 1024、2048 和 4096 下数据列 *D(CHL)*的不同数据压缩效果。从图中可以看出，压缩前后数据的变化趋势相符，但数据量却大大减少。初始时，原有数据集有 65255 条，设置 *CP(CHL)*=1024 进行数据压缩后，数据记录数减少至 8174 条。同理，*CP(CHL)*设为 2048 和 4096 时，压缩后数据记录条数分别减至 12681 和 23428。



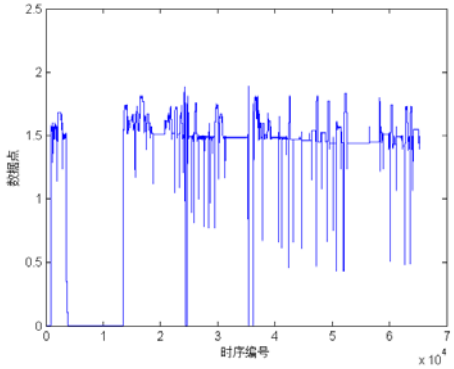
(a)原始数据



(b)压缩点数设为 1024 的压缩结果



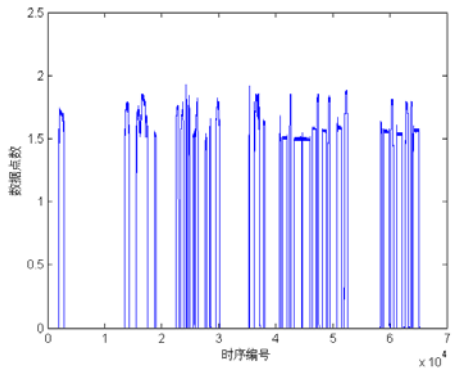
(c)压缩点数设为 2048 的压缩结果

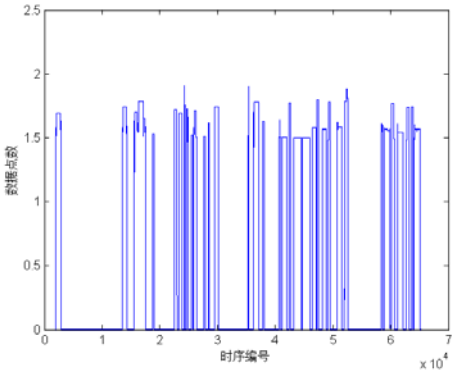


(d)压缩点数设为 4096 的压缩结果

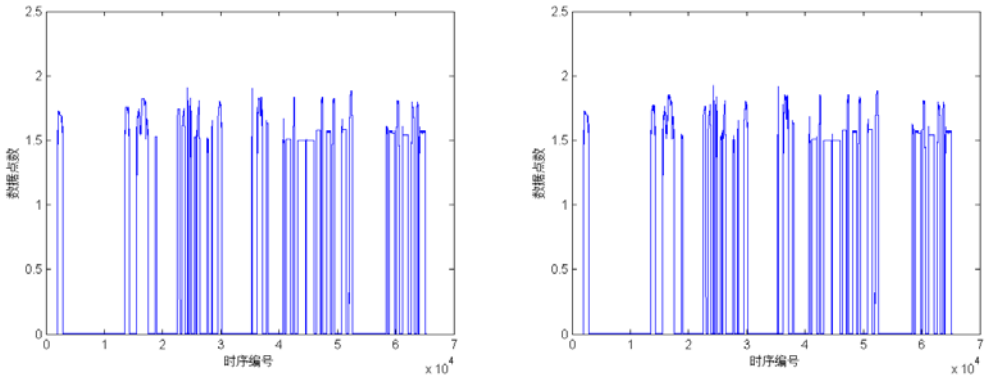
图 2 数据列 *D(CHL)*在不同压缩点数设置下的压缩结果

图 3 中(b) 、(c)和(d)则给出了数据列 D(CL)在压缩点数为 1024、2048 和 4096 下的压缩效果。





(a)原始数据　　(b)压缩点数设为 1024 的压缩结果



(c)压缩点数设为 2048 的压缩结果　　(d)压缩点数设为 4096 的压缩结果

图 3　数据列 D(CL)在不同压缩点数设置下的压缩结果

从图中可以看出，本文提出的高维时序数据线性压缩算法，很好地解决了高维时序数据压缩的问题。并可通过用户设置适当的压缩点数，达到较好的压缩效果。

3.2 压缩比分析

压缩比定义为：

$$R=P/M$$

其中，P 表示压缩之前的数据点数，M 表示压缩之后的数据点数。

表 4 给出了不同压缩点数下压缩数据量和相应的压缩比，从表中可以看出，压缩比受压缩点数设置的影响，设置的压缩点数越高，压缩比会下降。这是由于设置的压缩点数越高，在压缩过程中进行区间合并的可能性越小，数据保存原始数据量越大，因此压缩比越低。

表 4　不同压缩点数下压缩比

压缩点数CP	原始数据点数	压缩后数据点数	压缩比
1024	65255	8174	7.98
2048	65255	12681	5.15
4096	65255	23428	2.79

同时，压缩比会受到高维时序数据维数的影响，维数越大，压缩比越高。图 4 给出了不同数据维度 D 下，相同原始数据压缩比的变化趋势。从图中可以看出，对于相同原始数据，增加数据的维数，数据的压缩比会下降。

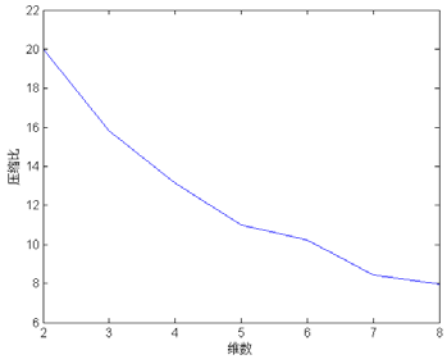


图 4　不同数据维数下的压缩比

4 总结

本文提出了一种针对海量高维时序数据的高效线性压缩存储方法，实现了高维时序数据的线性压缩与存储。高维时序数据的线性压缩存储方法不仅可以应用于传动装置试验数的压缩，还可以广泛应用于各类工业控制系统时序数据的压缩存储，尤其适用于数据流量大、采样种类多的情况，可较好地保留原数据的特征，实时性好，压缩比高，实用性强，压缩后的结果可以在关系数据库中存储和查询。

参考文献：

[1] 关云鸿,杨静.高维时序数据的相似搜索[J].贵州大学学报（自然科学版）,2006,23(1):44-50.DOI:10.3969/j.issn.1000-5269.2006.01.010.

[2]杨永军, 徐江, 许帅,等. 实时数据库有损压缩算法的研究[J]. 计算机技术与发展, 2012, 22(09):5-8.

[3]李雷定, 马铁华, 尤文斌. 常用数据无损压缩算法分析[J]. 电子设计工程, 2009, 17(1):49-50.

[4]游文杰 ， 吉国力 ， 袁明顺. 高维少样本数据的特征压缩[J]. 计算机工程与应用, 2006.

[5]田凯. 高维时序数据可视化系统[D]. 浙江大学, 2013.

[6]孟熠, 刘玉葆, 李启睿. 一种基于压缩策略的高维空间子空间 skyline 查询算法[J]. 计算机研究与发展, 2013(S1):101-108.

[7]段培永,张玫,汤同奎等.SDT 算法及其在局域控制网络中压缩过程数据的应用[J].信息与控制,2002,31(2):132-135.DOI:10.3969/j.issn.1002-0411.2002.02.008.

[8]黄勇, 吕涛. 基于线性分形插值函数的遥测数据压缩[J]. 计算机工程与应用, 2004, 38(16):40-42.

[9]王成山, 王继东. 基于能量阈值和自适应算术编码的数据压缩方法[J]. 电力系统自动化, 2004, 28(24):56-60.

[10]Mah R S H , et al.Process trending with piecewise linear smoothing. Comput Chem Engng , 1995 , 19(2):129-137

[11]Gray R M.Vector quantization .IEEE ASAP Magazine , Apr.1984 , 4-29

[12]Macgregor J F.Statisti cal process control of multivariate process.IFAC ADCHEM, Kyoto, Japan , 1994