一种高效的混合压缩数据挖掘算法 *

孙志长，冯祖洪，王沛栋
（北方民族大学 计算机科学与工程学院，银川 750021）

摘　要：针对基于垂直数据格式的关联规则挖掘算法在频繁项集查找过程中，需要在内存中保存大量的事务标志列表，有限的内存容量将成为此类算法的最大瓶颈，提出了一种新的混合压缩算法—HC-DM 算法。实验结果表明，将 HC-DM 算法与 dEclat 算法相结合，再加上排序步骤，可以显著减少频繁项集挖掘过程中的内存使用量。

关键词：数据挖掘；关联规则挖掘；频繁项集挖掘；HC-DM 算法
中图分类号：TP311；TP301.6　　**文献标志码**：A　　**文章编号**：1001-3695(2009)10-3738-05
doi:10.3969/j.issn.1001-3695.2009.10.040

Efficient hybrid compression algorithm for data mining

SUN Zhi-chang, FENG Zu-hong, WANG Pei-dong
(*Institute of Computer Science & Engineering, North Nationality University, Yinchuan* 750021, *China*)

Abstract: Vertical mining algorithms need lots of memory to hold the lists of transaction ids when enumerated all frequent itemsets, the limited memory's capability maybe become the most bottlenecks. To solve this problem, presented a new hybrid compressing algorithm-HC-DM algorithm. Performance evaluations indicate that integrating this algorithm with dEclat and increasing the step of sorting can drastically cut down the size of memory required when enumerates all frequent itemsets.

Key words: data mining; association rules mining; mining frequent items; HC-DM algorithm

关联规则挖掘是数据挖掘中的一个重要分支，最初被应用于超市购物篮数据分析，用来辅助市场营销、商品摆放、库存管理和客户关系管理。除了购物篮数据分析外，关联规则挖掘也被广泛地应用于其他领域，如生物信息学、医疗诊断、网页挖掘和科学数据分析等。例如，在地球科学数据分析中，关联模式可以揭示海洋、陆地和大气过程之间的有趣联系。这样的信息能够帮助科学家们更好地理解地球系统中自然力之间的相互作用[1]。

1　关联规则挖掘综述

关联规则挖掘就是找出支持度和置信度分别满足用户给定阈值的所有规则。关联规则挖掘通常分解为两个主要的子任务：a)频繁项集的产生，其目标是发现满足最小支持度阈值的所有项集，如图 1 所示；b)规则的产生，其目标是从上一步发现的频繁项集中提取出所有高置信度的规则。通常，产生频繁项集所需要的计算开销要远大于规则产生所需的计算开销。从概念上讲，数据集可以由一个二维矩阵表示，每行代表一个独立事务，每列代表一个项。现在以购物篮数据库为例，讨论数据集在物理实现上的四种表示形式[2]（图 2）：

a)水平项向量（horizontal item-vector，HIV）。数据库的每一行都有惟一的事务标志符，每列代表一个将要出售的商品。如果在某事务中售出此商品，就在此商品对应的位置上置“1”；否则置“0”。

b)水平项列表（horizontal item-list，HIL）。这种格式与上述 HIV 格式类似，不同的是每行列出的是本次交易商品的标志号，而不是“1”和“0”。

c)垂直事务标志向量（vertical tid-vector，VTV）。此种格式将数据库组织成列的形式。每列存储了商品的标志号和一系列的位向量“1”和“0”。如果商品在某个事务中出现，就在对应的事务标志上置“1”；否则置“0”。

d)垂直事务标志列表（Vertical tid-list，VTL）。与 VTV 格式类似，不同的是每列中存储的是事务标志号。

Distinct database items

Austen	Christie	Doyle	Twain	Wodehouse
A	*C*	*D*	*T*	*W*

database

transaction	items
1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

all frequent itemsets
minimum support=50%

support	itemsets
100%(6)	*C*
83%(5)	*W*,CW
67%(4)	A,D,T,AC,AW,CD,CT,ACW
50%(3)	AT,DW,TW,ACT,ATW,CDW,CTW,ACTW

图 1　频繁项集挖掘

水平项向量这种格式在实际应用中较少出现，大部分关联规则挖掘算法都采用 HIL 格式。这种格式的最大优点就是可以略过在上一次迭代中没有出现任何频繁项集的事务；缺点是在完成对数据集的第一次遍历后，还要多次读取非频繁项集。在稠密数据集中，项在大部分事务中频繁出现，此时 VTV 格式对数据集有着良好的压缩比率；不足之处是其无法对稀疏数据集进行压缩。因为在稀疏数据集中，连续含有同一个项的两个

收稿日期：2008-12-15；**修回日期**：2009-02-18　　**基金项目**：宁夏自然科学基金资助项目（NZ0697）；宁夏高等学校科学技术研究项目（2006JY018）

作者简介：孙志长（1982-），男，辽宁沈阳人，硕士研究生，主要研究方向为数据挖掘、计算机网络（sunzhichang@yahoo.com.cn）；冯祖洪（1956-），男，江苏无锡人，教授，主要研究方向为计算机网络、智能计算；王沛栋（1982-），男，山东青岛人，硕士研究生，主要研究方向为智能计算.

事务普遍间隔较大,即列表中会有大量的“0”出现。VTL 的最大优势在于利用此种格式的挖掘算法可以通过交集操作,忽略掉所有非频繁项,即在下次迭代中不会读取任何无用的非频繁项集;缺点是当频繁项集的 VTL 中的重复项较多时,基于交集操作的挖掘算法会产生大量的中间结果,此时有限的内存容量将成为这类算法的最大瓶颈。

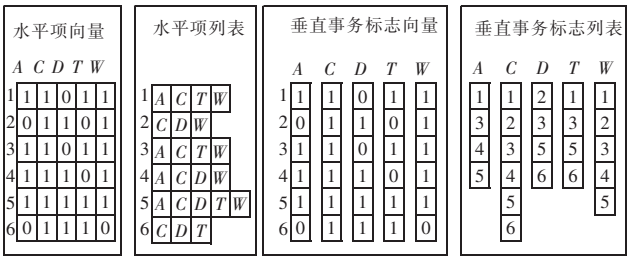

图 2　数据集在物理实现上的四种表示形式

2　Dif-bits 算法介绍

近年来,许多关联规则挖掘算法采用了垂直数据格式进行频繁项集挖掘,包括 Viper[2]、Eclat[3]、dEclat[4]、Charm[5] 和 Partition[6] 等。这些算法的优势在于它们不但可以忽略掉无用的非频繁项,还避免了复杂数据结构的建立,如经典的 Apriori 算法[7] 在利用反单调性搜索子集空间时,需要建立复杂的哈希树结构,而且当子集搜索空间较大时,子集匹配过程将耗费大量的时间。

2.1　内存容量与频繁项集挖掘

在频繁项集挖掘过程中,内存容量扮演着至关重要的角色。其中最大的问题就是在挖掘算法执行过程中,存储中间结果需要大量的内存空间。这里有两个主要原因:a)创建吉比特级的数据结构需要大量的内存;b)递归查找过程也会消耗大量的内存[8]。许多关联规则挖掘算法在面对大型数据集时几乎都会遇到此类问题。

频繁项集挖掘算法的性能通常可以通过两种方式进行改进,包括使用高效的挖掘算法和降低算法的内存空间使用量。然而,算法在执行过程中产生的数据量如果超出了内存的容量,则不管算法的数据结构如何高效,挖掘算法的整体性能将会严重下降,甚至有的算法无法完成挖掘任务。Dif-bits[9] 就是针对此问题所提出的一种数据压缩算法。

2.2　Dif-bits 算法

为了避免创建复杂的数据结构,Dif-bits 算法采用了垂直数据格式。在对数据集的读取过程中,算法首先用垂直事务标志列表中的当前事务标志减去前一个事务标志,得到一个事务标志差 D_i;然后将 D_i 转换为对应二进制格式,再将这些二进制比特插入到比特向量中。垂直事务标志列表中存储的事务标志是一个整数。在常见的计算机架构中,每个整型事务标志在内存中要占用 32 bit,相比之下 Dif-bits 算法节省了大量的存储空间。用当前的事务标志减去前一个事务标志,得到的差值一定小于当前的事务标志,所以存储 Dif-bits 所用的比特数一定小于存储原始事务标志所用的比特数。另外,Dif-bits 存储的是两个事务标志的差值,所以 Dif-bits 的大小与数据集中的事务总数无直接联系。

考虑图 1 所示的数据集,项 A 的 VTL 为{1,3,4,5}。如果用整型来存储这些事务标志,共需要 $32 \times 4 = 128$ bit。算法首先计算相邻标志的事务差 D_i,这里规定首个事务差就是第一个事务标志本身;第二个事务标志为 3,这时 D_i 等于 $3 - 1 = 2$,依次类推,然后得到的差值集合为{1,2,1,1};该算法将得到的差值转换为比特格式,并插入到用于保存结果的比特向量中,最后得到项 A 的 Dif-bits 为{11011}。可以看出,以 Dif-bits 格式存储项 A 的事务标志列表仅使用了 5 bit。但这种比特向量格式还无法还原成原始的 D_i,因为无法确定每个 D_i 被转换为 Dif-bits 后的精确比特位数。为了解决此问题,算法在插入 D_i 之前,先插入 n 的填充位。这里的 n 指出了后面二进制位串的长度。在将 Dif-bits 格式的向量还原成原始差值时,算法通过读取 n 的填充位,就可以得到原始差值的准确长度。还是考虑上例,这里假设加入了 3 bit 的填充位。由于第一个 D_i 的二进制格式为 1,其长度为 1,于是算法就使用填充位 001 来表示它的长度;下一个 D_i 的二进制为 10,其长度为 2,对应的填充位为 010。如果填充位的长度为 5,则最多可以表示出长度为 31 的比特位串。31 位二进制所能表示的整数已经超过 21 亿。一个项在如此大的事务间隔才出现一次是十分罕见的,如果数据集中某个项总是在经过 21 亿个事务后才出现一次,那么它的支持度一定小于 $5 \times 10^{-8}\%$,用户几乎不可能定义如此低的支持度[10]。因此,无论数据集有多少条事务,一般只需 5 个填充位即可。

3　HC-DM 算法

Dif-bits 算法所要解决的问题是当用户定义的支持度较低或面对的数据集较大时,挖掘算法产生的大量中间结果与有限的内存容量之间的矛盾。在存储事务差 D_i 时,算法使用二进制比特来代替整型数据,节省了大量的存储空间。在实际应用中,许多数据集都是稠密的,或呈分布不均匀状态,如大型超市中的季节性商品在特定的一段时间内,会在购物篮事务中频繁的出现。这时如果只是单一的使用 Dif-bits 格式,仍会浪费大量的存储空间。本文提出了一种新的混合压缩算法—HC-DM (hybrid compression for data mining)算法。HC-DM 算法要解决的问题是如何分辨出数据集的稠密部分和稀疏部分,然后再对每部分采用不同的存储格式,这样就可以节省更多的内存空间。

3.1　HC-DM 算法的压缩过程

在稠密数据集或分布不均匀的数据集中,项在某些连续的事务中频繁出现。如果将这些间隔较小的事务标志以 VTV 格式存储,就会节省更多的内存空间。例如,示例数据库中项 A 的事务标志之间的间隔较小,{1,3,4,5} 中相邻两个项之间最大的间隔仅为 2。如果将其差值{1,2,1,1}转换为 VTV 格式{10111},只使用了 5 bit;如果使用原始的 Dif-bits 格式,则需要 25 bit。因为无论相邻的事务标志差多大,Dif-bits 算法都需要 5 bit 的填充位来表示后面二进制位串的长度。

HC-DM 算法的主要思想是:当相邻的事务标志差 D_i 较大时,使用 Dif-bits 格式存储 D_i;当 D_i 连续较小,并且在结合使用 VTV 格式可以节省存储空间的情况下,就将这些连续的 D_i 以

VTV 格式存储。这里的问题是在对最终结果进行解压缩时，如何分辨出哪些比特是原始的 Dif-bits 格式，哪些是 VTV 格式。在 Dif-bits 算法中，前 5 bit 是填充位，用来标志后面二进制位串的长度。HC-DM 算法在这五个填充位的前面添加了一个标志位，用来区分两种数据格式。当标志位等于 0 时，代表后面的比特为 Dif-bits 格式；当等于 1 时，代表 VTV 格式。Dif-bits 中的五个填充位最多只能表示长度为 31 的比特向量，即当填充位等于 11111 时，这个长度对于 VTV 格式来说显然是不够用的。在 HC-DM 算法中，当标志位等于 1 时，这 5 位代表了后面二进制位串中 1 的个数。此种格式最多能表示出 VTV 中的 31 个事务标志差。算法根据不同的标志位和其后面的五个比特，就可以准确得到原始的事务标志差。

接下来 HC-DM 算法要解决的问题就是确定何时用 VTV 格式来存储 D_i。通过分析发现，10 是事务差的一个临界值。因为 10 的二进制格式为 1010，加上前面 6 位，刚好等于 10。如果 $D_i > 10$，那么它的 VTV 格式就一定多于 10 位，此时 VTV 格式相对于 Dif-bits 格式来说总是要占用更多的内存空间。当 $D_i < 10$ 时，就有可能通过和后面的 VTV 进行累积而达到节省存储空间的目的。这里 HC-DM 算法定义了一个增益函数 Gain，Gain 等于 Dif-bits 和 VTV 两种格式长度的比值，用来判断使用 VTV 格式存储 D_i 是否会节省存储空间。具体判断方法如下：如果当前事务差小于 10，算法将事务差 D_i 分别以 Dif-bits 和 VTV 两种格式存储在临时向量中；否则就直接以 Dif-bits 格式存储当前事务差。如果存在连续多个小于 10 的 D_i，且 D_i 的个数小于 31，算法就将这些 D_i 分别以两种格式进行累计存储，并计算 gain 值。如果 gain > 1，就说明此时使用 VTV 格式可以节省更多的存储空间。当用来保存两种数据格式的临时向量非空时，HC-DM 算法分四种情况将临时向量中的值插入到最终结果：

a) 当 gain > 1，并且小于 10 的事务差累计到 31 的时候，将临时向量中的 VTV 插入结果集，清空临时向量；

b) 当前事务差大于等于 10，且 gain > 1，将临时向量中的 VTV 插入结果集，再将当前 D_i 以 Dif-bits 格式插入结果集，清空临时向量。

c) 当前事务差大于等于 10，而 gain < 1，将临时向量中的 dif-bits 插入结果集，再将当前 D_i 以 Dif-bits 格式插入结果集，清空临时向量。

d) 当对最后一个 D_i 判断结束后，再次检查临时向量和增益标志，并按上述方法将剩余的部分插入结果集。

下面举一个简单的例子来详细说明一下算法的执行过程。假定某一垂直事务标志列表为{2,3,5,7,8,100,109,200}，算法首先计算当前 D_i。因为第一个 $D_i = 2 < 10$，算法就将 2 分别以两种数据格式存储。2 的 VTV 格式为 01，VTV 格式的标志位为 1，当前只有一个元素，所以填充位为 100001，临时 VTV 向量中的二进制位串为 10000101，长度为 8。2 的二进制格式为 10，Dif-bits 格式的标志位为 0，当前 D_i 的长度为 2，所以填充位为 000010，临时 Dif-bits 向量中的二进制位串为 00001010，长度为 8。这时得到的 gain 值为 8/8 = 1。第二个 $D_i = 1$，VTV 为 1，Dif-bits 为 0000011，将两种格式分别进行累计，得到 VTV = 100010011，Dif-bits = 000010100000011，gain 值为 15/9，依此类推。当进行到事务标志 100 时，当前 $D_i = 92$，大于 10，再判断 gain 值，得到当前增益大于 1，于是将累计在临时向量中的 VTV 插入到结果集，在以 dif-bits 格式插入当前事务差 D_i。下一个事务差为 9，gain 值为 10/15，此时算法继续对下一个事务差进行判断。得到当前 $D_i = 91$，大于 10，经过判断，gain < 1，即没有得到增益，于是算法就将存储在临时向量中的 Dif-bits 插入到结果集；再以 Dif-bits 格式插入当前事务差 91；最后得到一个长度为 50 的比特向量。与 Dif-bits 算法相比，HC-DM 算法仅增加了一些简单的判断和插入操作，所以两种算法的时间复杂度为同阶。HC-DM 算法的伪代码如下：

```
input:dataSet(D),items(I),minimum support(s)
output:set of bit vectors
  for all t∈T in D
    for all i∈I in t
    计算当前事务差 Di;
  if Di<10
    计算累计计数 count;
    将 Di 转换为 VTV 格式;
    存储到对应的临时向量中;
    将 Di 转换为 Dif-bits 格式;
    存储到对应的临时向量中;
    计算增益函数 gain;
    if gain>1
      if count<31,继续累计;
      if count=31,将临时向量中的 VTV 插入到结果集;
  else if Di≥10,并且得到增益
    将临时向量中的 VTV 插入到结果集;
    再以 Dif-bits 格式插入当前 Di;
  else if Di≥10,但没有得到增益
    将临时向量中的 Dif-bits 插入到结果集;
    再以 Dif-bits 格式插入当前 Di;
    else
    直接以 Dif-bits 格式插入当前 Di;
  end for
end for
addHybVector(tmpVTVVectror || tmpBitVector) //循环结束后,插入最后剩余的部分
```

3.2　HC-DM 算法的解压缩过程

相对于压缩过程，解压缩过程较为简单。算法首先对标志位进行判断，如果等于 1，则后面的 5 位比特就代表了 VTV 格式中 1 的个数。这时根据 1 的个数，可以直接在比特向量中连续提取出 D_i。如果标志位为 0，则后 5 位比特代表二进制比特的位数。根据这个长度就可以提取出当前的事务差。将得到的事务差与前面相邻的事务标志相加，就可以得出原始的事务标志。解压缩的伪代码如下：

```
input:A bit vector(V)
output:set of tid(T)
t=0;
for all e∈V
  flag=readFlagBit(e);
  if flag=0;
    p=readPaddingBits(0);
    d=readDiffBits(p);
    n=convertToInteger(d);
    t=t+n;
  else
    r=readPaddingBits(1);
    v=readVTVBits(r)
     i=convertToInteger(v);
```

```
        t = t + n //可连续提取出 r 个 D_i
    end for
```

4　结合 dEclat 算法挖掘频繁项集

dEclat 算法使用 Diffsets 数据格式,通过对当前项的事务标志列表和前一个项的事务标志列表进行差集运算,得到新的候选项集。接着算法判断得到的候选项集是否满足用户定义的最小支持度;然后将满足条件的频繁 k-项集保存到中间结果集中,当中间结果集非空时,算法就进行递归调用,直到产生所有的频繁项集。有关 dEclat 算法的详细介绍请参考文献[4]。

本章将 HC-DM 算法与 dEclat 算法结合在一起进行频繁项集挖掘。首先用 HC-DM 数据格式存储原始的 Diffsets,然后在频繁项集挖掘过程中,使用 HC-DM 格式来保存中间结果。在原始 dEclat 算法的基础上,增加了排序步骤。在进行差集运算之前,首先对当前层的候选项集按照 VTL 的势进行递减排序,排序后再进行差集操作会显著减少下一层候选项集的生成量。例如,$|M| = 5\ 000$,$|N| = 100\ 000$,对 M 和 N 进行差集运算。$|M - N|$在最好的情况下等于 95 000,最坏的情况下等于 100 000;$|N - M|$的最坏情况仅为 5 000,而最好的情况等于 0。其余的步骤与 dEclat 算法相同,这里不再重复。

dEclat 算法在执行过程中,一共产生 22 个事务标志,数据量为 $22 \times 32 = 704$ bit。下面计算结合 Dif-bits 格式的 dEclat 算法、结合 HC-DM 格式的 dEclat 算法和结合 HC-DM 格式并排序后的 dEclat 算法各自的内存使用量,如图 3 ~ 5 所示。结合 dif-bits 格式的 dEclat 算法产生了 160 bit;结合 HC-DM 格式的 dEclat 算法产生了 141 bit;结合 HC-DM 格式并排序后的 dEclat 算法仅产生了 86 bit。dEclat 算法通过与 HC-DM 算法相结合,尤其经过排序步骤后,显著降低了内存的使用量,这意味着结合后的算法具有更大的伸缩性。

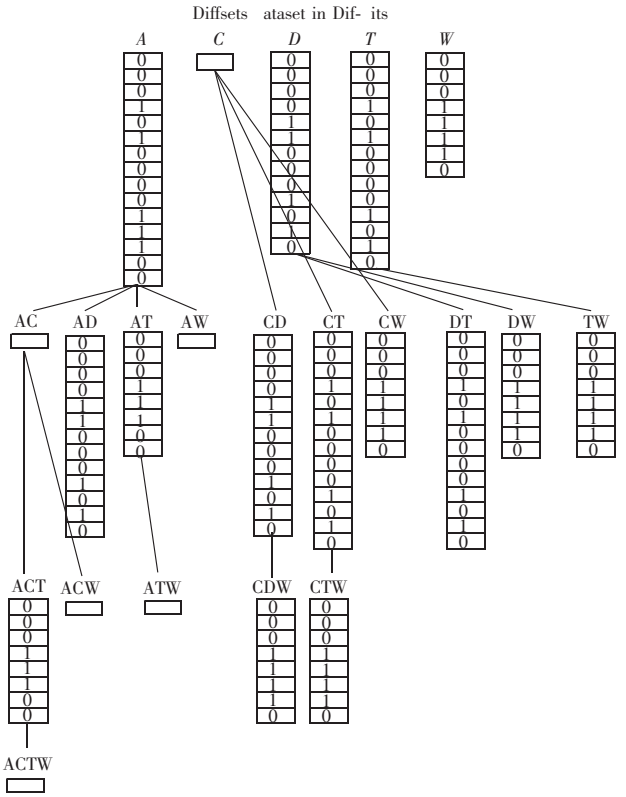

图 3　结合 Dif-bits 格式的 dEclat 算法

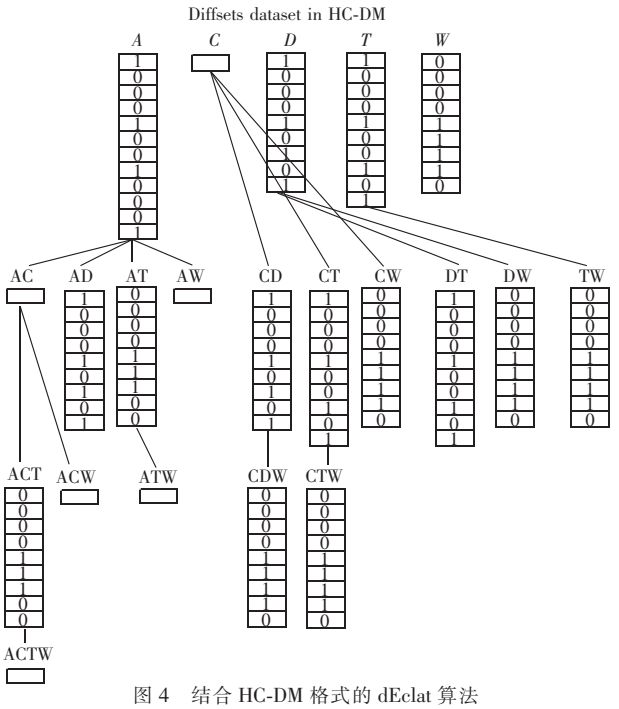

图 4　结合 HC-DM 格式的 dEclat 算法

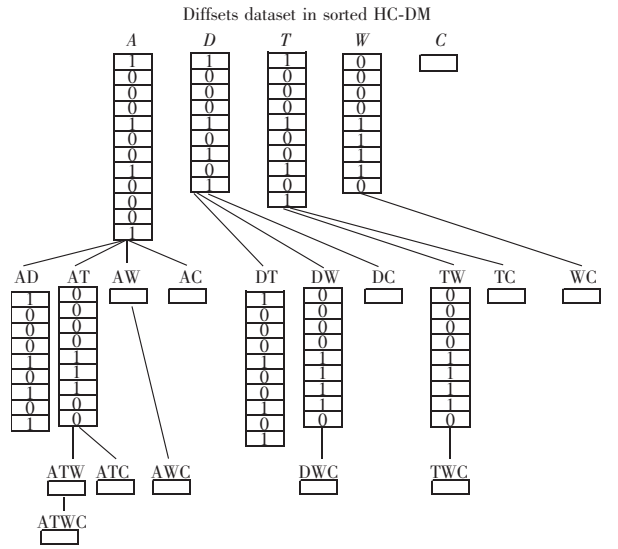

图 5　结合 HC-DM 格式并排序后的 dEclat 算法

5　实验结果和实验分析

5.1　实验结果

实验用的数据集可以在 http://fimi.cs.helsinki.fi/data/上得到。Mushroom 数据集包含了多种蘑菇的特征;connect 和 chess 数据集来源于各自的游戏步骤;T10I4D100K 数据集是一个人造数据集;pumsb_star 数据集中包含的是人口普查数据。基于垂直数据格式的频繁项集挖掘算法主要分为两个步骤:首先算法在内存中存储候选项集的事务标志列表;然后利用交集操作得出所有的频繁项集[10]。由于本实验的目的是为了比较 VTL、Dif-bits 和 HC-DM 算法各自的内存使用量,这里只关心上述过程的第一步。将整个测试数据集的所有 1-项集的事务标志列表分别以 VTL、Dif-bits 和 HC-DM 三种格式全部装入内存,然后再对三种格式的内存使用量进行比较。测试数据集的特征和基于三种数据格式的内存使用量的比较结果如表 1 所示。

表 1　测试数据集的特征和三种格式的内存使用量

数据集	项数	记录数	VTL	Dif-bits	HC-DM
mushroom	120	8 124	729	151	58
connect	130	67 557	11 347	2 217	619
chess	76	3 196	461	88	23
T10I4D100K	1 000	100 000	3 946	1 310	1 419
pumsb_star	7 117	49 046	9 671	2 211	1 251

5.2　实验分析

实验中所用的四个真实数据集相对较为稠密,在这些数据集上 HC-DM 算法表现出了良好的压缩比率。由实验结果可以看出,在真实数据集上,HC-DM 算法的内存使用量为 Dif-bits 算法的 1/3 左右。与 VTL 相比,HC-DM 算法的内存使用量仅为它的 1/15 左右。T10I4D100K 数据集极为稀疏,项很少在连续的事务中频繁出现。算法多用了 1 bit 的标志位并且只得到了很少的增益,但从实验结果可以看出,HC-DM 算法的内存使用量仅比原算法多 10% 左右。由此可以看出,即使在最坏的情况下,HC-DM 算法的性能也接近于 Dif-bits 算法。

6　结束语

关联规则挖掘算法在频繁项集挖掘过程中需要大量内存空间,如果内存容量不足,挖掘算法的整体性能会严重下降。本文提出了一种基于垂直数据格式的高效混合压缩算法。实验结果表明,在大多数情况下,算法具有良好的压缩比率。

参考文献:

[1] TAN Pang-ning, STEINBACH M, KUMAR V, *et al*. 数据挖掘导论[M]. 范明,范宏建,译. 北京:人民邮电出版社,2006:203-204.

[2] SHENOY P, HARITSA J R, SUDARSHAN S, *et al*. Turbo-Charging vertical mining of large databases[C]//Proc of the ACM SIGMOD International Conference on Management of Data, Te. New York: ACM Press, 2000:22-33.

[3] ZAKI M J. Scalable algorithms for association mining[J]. IEEE Trans on Knowledge and Data Engineering, 2000, 12(3):372-390.

[4] ZAKI M J, GOUDA K. Fast vertical mining using diffsets[C]//Proc of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2003:326-335.

[5] ZAKI M, HSIAO C. CHARM: an efficient algorithm for closed itemset mining[C]//Proc of the 2nd SIAM International Conference on Data Mining. Virginia: [s. n.], 2002:457-473.

[6] OMIECINSKY E, SARASERE A, NAVATHE S. An efficient algorithm for mining association rules in large databases[C]//Proc of the 21th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc, 1995:432-444.

[7] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[C]//Proc of the 20th International Conference on Very Large Data Bases. San Jose: Morgan Kaufmann, 1994:487-499.

[8] MOHAMMAD E, OSMAR R. Inverted matrix: efficient discovery of frequent items in large datasets in the context of interactive mining [C]//Proc of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2003: 109-118.

[9] ASHRAFI M Z, TANIAR D, SMITH K. An efficient compression technique for vertical mining methods[C]//Proc of Research and Trends in Data Mining Technologies and Applications in the United Kingdom by Idea Group Publishing. London: David Taniar, 2007:143-173.

(上接第 3730 页)诊断器的物理不可实现问题。最后通过仿真实例证实了本文提出的故障诊断方法及故障可诊断性判据的可行性和有效性。

参考文献:

[1] CASAVOLA A, FAMULARO D, FRANZE G. Robust fault detection of uncertain linear systems via quasi-LMIs[J]. Automatica, 2008, 44(1):289-295.

[2] KHOSROWJERDI M J, NIKOUKHAH R, SAFARI-SHAD N. Fault detection in a mixed setting[J]. IEEE Trans on Automatic Control, 2005, 50(7):1063-1068.

[3] FONG K F, LOH A P, TAN W W. A frequency domain approach for fault detection[J]. International Journal of Control, 2008, 81(2): 264-276.

[4] 李娟,叶若红,李保华. 基于内模原理的变结构最优容错控制[J]. 系统工程与电子技术,2008,30(4):727-730.

[5] ZHONG Mai-ying, DING S X, LAM J, *et al*. Fault detection filter design for LTI system with time delays[C]//Proc of IEEE Conference on Decision and Control. Maui, HI, United States: Institute of Electrical and Electronics Engineers Inc, 2003:1467-1472.

[6] BAI Lei-shi, TIAN Zuo-hua, SHI Song-jiao. Robust fault detection for a class of nonlinear time-delay systems[J]. Journal of the Franklin Institute, 2007, 344(6):873-888.

[7] 白雷石,田作华,施颂椒. 线性状态多时滞系统鲁棒故障诊断滤波器设计[J]. 系统工程与电子技术,2005,27(10):1781-1784.

[8] LI Tao, GUO Lei, WU Ling-yao. Observer-based optimal fault detection using PDFs for time-delay stochastic systems[J]. Nonlinear Analysis: Real World Applications, 2008, 9(5):2337-2349.

[9] YANG Han-long, SAIF M. Observer design and fault diagnosis for state-retarded dynamical systems[J]. Automatica, 1998, 34(2): 217-227.

[10] 尤富强,田作华,施颂椒. 线性时滞系统的鲁棒故障诊断[J]. 系统工程与电子技术,2005,27(9):1608-1613.

[11] JIANG Cang-hua, ZHOU Dong-hua. Fault detection and identification for uncertain linear time-delay systems[J]. Computers and Chemical Engineering, 2005, 30(2):228-242.

[12] MAO Ze-hui, JIANG Bin. Fault estimation and accommodation for networked control systems with transfer delay[J]. Acta Automatica Sinica, 2007, 33(7):738-743.

[13] 李娟,叶若红,唐功友. 含控制时滞系统的实时故障诊断和最优容错控制[J]. 控制与决策,2008,23(4):439-444.

[14] ZHENG Ying, FANG Hua-jing, WANG Hua, *et al*. Observer-based FDI design of networked control system with output transfer delay[J]. Control Theory and Applications, 2003, 20(5):653-663.

[15] FIAGBEDZI Y, PEARSON A. Feedback stabilization of linear autonomous time lag systems[J]. IEEE Trans on Automatic Control, 1986, 31(9):847-855.