文章编号:1000-5862(2015)05-0463-06

一种基于 DHP 的动态链地址关联规则挖掘算法

吴　恒,吴根秀*,毛临川,黄　梅
(江西师范大学数学与信息科学学院,江西 南昌　330022)

摘要: 采用动态链地址法建立 H_k 表,提出 DLDHP 算法,克服 DHP 算法不能给 C_k 中的每个项集单独计数的缺点,在增加 H_k 表的空间时也会删除链表中的结点空间,从而 H_k 表不会占用较大的内存,最终直接由 H_k 表得到频繁 k-项集的集合 L_k,不用再次扫描数据库. 实验结果表明: DLDHP 算法是有效可行的.

关键词: 关联规则; DHP 算法; DLDHP 算法; Hash 表; 动态链地址
中图分类号: TP 311　　**文献标志码:** A　　**DOI:** 10.16357/j.cnki.issn1000-5862.2015.05.05

0　引言

关联规则挖掘[1]是目前数据挖掘领域应用最为广泛和成熟的一个分支,主要目的是从给定的数据库中发现项目之间有趣的关联或相关关系. 查找频繁项目集一直是关联规则挖掘的关键技术和步骤,由此展开大量的相关研究与改进工作. Apriori 算法[2]是一种经典的挖掘频繁项集的算法,但当挖掘强关联模式或阈值设置过低时,会生成大量的候选项集且需要重复扫描数据库,MFIs 算法[3],DHP 算法[4]等是在 Apriori 算法的基础上改进或者衍生出来的.

对 DHP 算法的改进也有较多,卢云彬等[5]提出了一种有序 Hash 表的 DHP 算法,能够较好地节省存储空间,提高挖掘效率. 张江等[6]引入了改进的 2 维哈希函数,在对 Apriori 算法改进的 2 维哈希算法中引入了新的哈希函数. 该哈希函数有效地避免了哈希"冲突",有利于迅速产生频繁 2-项集,从而提高了频繁项集产生的效率. 曾传璜等[7]提出了多项集元素的 hash_tree 快速查找方法,该方法在查找元素是多项集结构方面有独到的应用价值,在数据挖掘中的多维关联规则算法的实现过程中产生了较好的效果. 崔玮等[8]给出了一种基于最小完美哈希函数的关联规则的挖掘算法,这一基于 Apriori 的算法在综合了传统哈希剪枝技术的同时,充分利用了最小完美哈希函数的优点,从而在保证静态数据库关联规则挖掘的同时,使对关联规则的哈希结构数据进行动态调整成为可能. 该算法不仅提高了挖掘效率,而且通过抑制哈希地址冲突提高了算法的稳定性和可用性.

然而,上述文献是通过减少地址冲突从而尽量压缩每个候选项集的集合,任何压缩都不可能达到百分之百,最终还是需要再次扫描数据库. 黄传明[9]考虑了一种可以直接删除非频繁项集的散列函数,虽然能够给集合 C_k 中的每个项集单独计数,但是给定的 Hash 表长为候选项集的大小,会占用较大的内存空间,并且也没有给出具体的实现过程.

本文在 DHP 算法的思想上,采用动态链地址法建立表,提出了 DLDHP 算法,在增加表的空间时也会删除其中的链表空间,从而 H_k 表不会占用较大内存,最终直接由 H_k 表得到频繁 k-项集的集合 L_k,不需再次扫描数据库.

1　关联规则挖据

令 $I=\{i_1,i_2,\cdots,i_m\}$ 为全体项目的集合,$D=\{t_1,t_2,\cdots,t_n\}$ 为全体事务集合,每一个事务 t_j $(1\leqslant j\leqslant n)$ 是包含 I 中一些项目的集合. 设 X 是 I 中一些项目的集合,若 $X\subseteq t_j$,则称事务 t_j 包含 X.

一个关联规则为形如 $X\Rightarrow Y$ 的蕴含式,其中 $X\subseteq I$,$Y\subseteq I$,且 $X\cap Y=\varnothing$. 规则 $X\Rightarrow Y$ 在事务数据库 D 中的支持度是事务集中包含 X 和 Y 的事务数与所有事务数之比,记为 $\sup(X\Rightarrow Y)=P(X\cup Y)$;规则 $X\Rightarrow Y$ 在事务集中的可信度是指包含 X 和 Y 的事务数与包含 X 的事务数之比,记为

收稿日期:2015-06-20
基金项目:江西省教育厅科学技术(GJJ14244)资助项目.
通信作者:吴根秀(1965-),女,江西南丰人,教授,主要从事数据挖掘、不确定性推理和信息融合的研究.

$$Conf(X \Rightarrow Y) = P(X \cup Y)/P(X).$$

关联规则挖掘分为 2 个阶段:

1) 生成频繁集,这些项集必须满足预定义的最小支持度 $sup_{\min}$.

2) 产生关联规则,这些规则必须同时满足最小支持 $sup_{\min}$ 和最小置信度 $conf_{\min}$.

2 DHP 算法

DHP 是利用散列修剪技术的算法来减少数据的传输量. DHP 算法可以较好地解决 Apriori 算法剪枝效率不高的问题,它通过给定一个散列函数将候选项集映射到一张散列表中,利用散列表地址对应的计数值删除不是频繁项集的候选项集,这样压缩了要考察的候选项集,特别是解决了生成频繁 2-项目集时的性能瓶颈问题.

DHP 算法设置了散列桶,将产生冲突的候选项集放入桶中并增加对应的桶计数,当项目个数较多或者事务集较大时,所有的桶计数都将会达到预设的最小支持度计数,压缩的效果不够好,并且由于不同的项集会共用同一个地址,当桶计数大于预设的最小支持度计数时,为了判断这些桶中的候选集是否为频繁项集还需再次扫描数据库.

本文主要针对 DHP 算法的以上缺点,采用动态链地址的方法,直接从建立完成的 Hash 表得到频繁项集,不需再次扫描数据库.

3 DLDHP 算法

本文目的是给每个候选项集单独计数,若给定的 Hash 表长为候选项集大小,当然能够使每个候选项集唯一对应一个散列地址,但是可能会占用大量的内存空间,如候选 2-项集的集合 C_2 大小为 C_n^2,其中 n 为频繁 1-项集的集合 L_1 大小. 因此,采用动态链地址法,既能够有效处理地址冲突,又不会占用较大内存空间.

链地址法: 对 Hash 表中每个 Hash 值建立一个冲突表,即将冲突的几个记录以表的形式存储在其中.

动态链地址法: 在链地址法的基础上增加了 S 空间,用来存储判定为频繁项集的元素. 同时也增加了链表中元素的计数空间. 在存储元素时,先与 S 比较,判断是否需要放入链表中,并且当链表中计数达到最小支持度计数时将元素放入对应的 S 中并删除该元素及其计数空间,结构如图 1 所示,其中 S_h 为存放地址为 h 且判定为频繁项集的元素集合,L_h 为存放地址为 h 且暂时还不是频繁项集的链表,最初的 H_k 表是不存在的,也即未知大小.

地址	S	L(链表)					
0	集合	元素	计数	…	元素	计数	…
1	集合	元素	计数	…	元素	计数	…
2	集合	元素	计数	…	元素	计数	…
…	…	…	…	…	…	…	…

图 1 动态链地址法的 H_k 表

构造过程如下:

(i) 得到元素 X 的 Hash 值为 h: 若 h 不存在,则建立 h 及对应的空间和 L_h 链表,把元素和计数放入 L_h 中; 反之,转(ii);

(ii) 在 S_h 中查找 X 是否存在,若存在,则转到(i); 反之,转(iii);

(iii) 查找 L_h 中是否存放了 X,若是,则加入其中并增加计数; 反之,在 L_h 中新增一个结点,把 X 放入其中并计数;

(iv) 当 L_h 中元素的计数等于最小支持度计数,则将该元素放入 S_h 中,并删除该元素及其计数空间;

(v) 扫描完数据库,此时直接得到频繁项集的集合为 $S_0 \cup S_1 \cup \cdots$

例如给定最小支持度计数为 2,在数据库中顺序扫描到的 3-项集为{ABC}、{CDF},{BEH} 和{ABC},且设它们的值分别为 0,2,2 和 0. 首先地址 0 不在 H_3 表中,建立地址 0 及对应的 S_0 空间和 L_0 链表,并将{ABC} 和计数 1 放入 L_0 中. 接着地址 2 也不在 H_3 表中,建立地址 2 及对应的 S_2 空间和 L_2 链表,并将{CDF} 和计数 1 放入 L_2 中. 之后又是地址 2,但地址 2 存在并且 S_2 中没有{BEH},但 L_2 中也没有该元素,则在 L_2 中新增一个结点,把{BEH} 和计数 1 放入其中. 最后地址又是 0,但是地址 0 存在并且 S_0 中没有{ABC},但在 L_0 中已经存放了{ABC},只需增加计数,计数变为 2 等于最小支持度计数,将{ABC} 放入 S_0 中,并把{ABC} 和计数空间都删除.

下面讨论 Hash 函数的确定问题,首先修剪数据库 D,将全为 0 的事务行和全为 0 的项目列删除得到 D_1,扫描一次数据库 D_1,得到 C_1 及各项对应的计

数,与最小支持度计数比较得到 L_1,并对 L_1 中的项排列编号,修剪数据库 D_1 得到数据库 D_2.

对于 2-项集: 将数据库 D_2 中每个事务 t 分解为 2-项集的集合,并用如下给定的 Hash 函数来计算 D_2 中每个项集的地址:

$$H(x,y) = ((n-3)*order(x) + order(y)) \bmod (prime(\sqrt{C_n^2})), \quad (1)$$

其中 n 为 L_1 中项的个数,$prime(t)$ 为取不大于 t 的最大素数,$1 \leqslant order(x) < order(y) \leqslant n$.

由于动态链地址的 Hash 表既有纵向长度也有横向长度,为了使得 Hash 表结构紧凑,选择除数为 $prime(\sqrt{C_n^2})$,使得横向和纵向长度近似相等.

$order(x)$ 的系数: 设 $order(x)$ 的系数为 a,对于 $order(x)$ 从 1 到 $n-1$,可以得到 $n-1$ 个区间为 $[a+2, a+n]$, $[2a+3, 2a+n]\cdots$, $[(n-2)a+(n-1), (n-2)a+n]$, $[(n-1)a+n, (n-1)a+n]$,每个区间内的值是相连的,让各区间也能相连,这样项集平均分配到各地址,得到如下不等式方程:

$$\begin{cases} 2a+3 \geqslant a+n, \\ 3a+4 \geqslant 2a+n, \\ \cdots \\ a \geqslant 1, n \geqslant 1. \end{cases} \Rightarrow a \geqslant n-3 \ (n \geqslant 4),$$

因此选定 $order(x)$ 的系数为 $n-3$,这样能够进一步保证 Hash 表结构紧凑.

对于 k-项集($k \geqslant 3$): 将数据库 D_k 中每个事务 t 分解为 k-项集的集合,给定的 Hash 函数与(1)式相同,只是这里的 x 为项集首项,Y 为后面所有的项.

$$H(x,Y) = ((n-3)*order(x) + order(Y)) \bmod (prime(\sqrt{C_n^2})), \quad (2)$$

$1 \leqslant order(x) < order(Y) \leqslant n$.

如计算 3-项集$\{ABC\}$ 的 Hash 值,其中 A、B、C 编号为 1、2、3,则有

$$H(A,BC) = ((n-3)*1 + (2+3)) \bmod (prime(\sqrt{C_n^2})).$$

算法的主要步骤如下:

1) 扫描事务数据库 D_1 中的每个事务,产生候选 1-项集的集合 C_1,根据最小支持度 $sup_{\min}$,由 C_1 产生频繁 1-项集的集合 L_1,得到 L_1 的大小,从而可以确定 Hash 函数.

2) 利用 Apriori 算法的连接方法得到候选 k-项集的集合 C_k,并且对数据库 D_{k-1} 进行修剪得到 D_k,再将 D_k 中的每个事务拆分为 k-项集的集合,计算每个 k-项集的 Hash 值,利用动态链地址法存储项集,存储过程如前所述. 当扫描完 D_k,S_h 中存储地址为 h 的频繁 k-项集,$\bigcup_{h=0}^{\cdots} S_h$ 即为频繁 k-项集的集合 L_k.

3) 对于 D_{k-1} 的每个事务: 当其长度小于 k,则不可能生成频繁 k-项集,将其删除; 当其不包含任何频繁$(k-1)$-项集,则肯定不包含频繁 k-项集,将其删除. 这样就修剪了数据库 D_{k-1} 得到数据库 D_k.

算法伪代码:

```
(i) // 主函数
function(D, sup_min)
L_1 = find(D, sup_min) // 找到所有的频繁 1-项集
for(k = 2; L_k = ∅; k ++)
C_k = apriori_gen(L_{k-1}) // 生成候选 k-项集的集合
D_k = reduce(D_{k-1}) // 修剪数据库
L_k = DL_Hash(D_k, S) // 由动态链地址生成频繁 k-
                      // 项集的集合
until L_k = ∅ // 结束查找
end
result = ∪ L_k // 得到所有的频繁项集.
(ii) // 子函数,修剪数据库
functionreduce(D_{k-1})
for k ⩾ 2
for t ∈ D_{k-1}
if |t| ⩽ k
delete t // 删除长度小于 k 的事务
end
C_tk = subset(t) // 将事务 t 拆分为 k-项集的集合
if(c_{tk_i} ∈ C_tk ∩ c_{tk_i} ∉ C_k)
    delete c_{tk_i} // 如果事务 t 中的某个 k-项集不在候
                    // 选 k-项集的集合中就删掉
end
end.
functionDL_Hash(D_k, S) // 生成 Hash 表并得到
                        // 频繁项集
n = length(L_1) // 频繁 1-项集的集合大小
for t ∈ D_k
h = H(c_{tk_i}) // 由公式可以计算事务 t 中每个 k-项集
                // 的地址
if h = null // 地址 h 不存在
create(h, S_h, L_h) // 创建地址、集合及链表
save(c_{tk_i}, L_h, count) // 存储 k-项集到对应链表
else
if c_{tk_i} ∈ S_h // 该 k-项集已确定为频繁 k-项集
continue
else
if c_{tk_i} ∉ L_h
```

$add(L_h\_node)$ // 在 L_h 中增加一个结点
$save(c_{tk_i}, L_h, count)$
else
$count\_c_{tk_i}$ ++ // 增加该 k-项集的计数
end
end
end
if $count\_c_{tk_i} == N * sup_{\min}$ // 计数等于最小支持
// 度计数
$save(c_{tk_i}, S_h)$ // 将该 k-项集放到对应地址的 S 中
delete(L_h_node) // 删除该结点
end
end
$L_h = \bigcup_{h=0}^{\cdots} S_h$ // 得到频繁 k-项集的集合.

4 数据实例

给定数据库 D 如表 1,修剪 D 知 $D_1 = D$,设最小支持度计数为 4,扫描 D 得到频繁 1-项集的集合为 $L_1 = \{\{A\},\{B\},\{D\},\{F\},\{G\}\}$,对 A、B、D、F、G 编号为 1、2、3、4、5,修剪数据库 D 便得到 D_2 如表 2 所示,在表 2 中非频繁项 C 和 E 被删除,删除 C 和 E 后事务长度小于 2 的 t_3 和 t_5 也被删除,最后分解数据库 D_2 如表 3 所示.

利用 Apriori 算法的连接原理,就得到候选 2-项集的集合 $C_2 = \{\{AB\},\{AD\},\{AF\},\{AF\},\{AG\},\{BD\},\{BF\},\{BG\},\{DF\},\{DG\},\{FG\}\}$ 文献[4]中 DHP 算法给定 Hash 函数为

$H(x,y) = (10 * order(x) + order(y)) \bmod 7$ 对表 3 中的所有 2-项集进行计算,得到地址从 0 到 6 的桶计数向量为 〈7,3,4,10,4,4,3〉,与最小支持度计数 4 比较,小于 4 的取 0,反之取 1,得到结果位向量为 〈1,0,1,1,1,1,0〉,可以看到地址 1 和 6 的取值为 0,从而将它们中的项集{AG},{AD} 和{DF} 删除,得到筛选后的候选 2-项集的集合为 $C_2^* = \{\{AB\},\{AF\},\{BD\},\{BF\},\{BG\},\{DG\},\{FG\}\}$,这些都需要再次扫描 D_2 确定是否为频繁项集.

但是,通过 DLDHP 算法得到 Hash 函数

$H(x,y) = (2 * order(x) + order(y)) \bmod 3$.

对表 3 中的所有 2-项集进行计算,建立的 H_2 表如图 2 所示. 只需扫描一次 D_2,当建立完 H_2 表就直接得到频繁 2-项集的集合为 $L_2 = S_0 \cup S_1 \cup S_2 = \{\{AF\},\{BG\},\{FG\},\{AB\},\{BD\},\{BF\}$ 不需要再次扫描数据库.

在图 2 中可以看到,地址 0 中元素全都在 S_0 中,即全为频繁项集,因而 L_0 为空. 对于地址 1 和地址 2,S_1,L_1 和 S_2,L_2 中均含有元素,即这 2 个地址中既有频繁项集又有非频繁项集. 最终,建立完成的 H_2 表结构比较紧凑并且最后占用的内存空间很小.

同理修剪数据库 D_2 得到数据库 D_3,最后分解数据库 D_3 并建立 3-项集的 H_3 表,得到的集合 S_0,S_1 和 S_2 都为空,即没有频繁 3-项集,从而结束查找频繁项集. 合并所有的频繁项集,得到

$L = \{\{A\},\{B\},\{D\},\{F\},\{G\},\{AF\},\{BG\},\{FG\},\{AB\},\{BD\},\{BF\}\}$.

5 实验及结果分析

实验环境如下: 操作系统 Windows 7,编程语言 Matlab 下分别变化项目个数和最小支持度,通过 DHP 和 DLDHP 算法时间复杂度比较进行分析:

1) 随机生成数据库 D,D 中事务个数为 1 000,给定的最小支持度为 0.2,比较 DHP 和 DLDHP 算法随 D 中项目个数变化在查找频繁 2-项集时所用时间. 比较结果如图 2 所示.

表 1 数据库 D

事务	项集合
t_1	$ACEFG$
t_2	CFG
t_3	EG
t_4	$ABDFG$
t_5	CG
t_6	$BDFG$
t_7	BDG
t_8	BD
t_9	FG
t_{10}	AB
t_{11}	ABF
t_{12}	$ABFG$

表 2 数据库 D_2

事务	项集合
t_1	AFG
t_2	FG
t_4	$ABDFG$
t_6	$BDFG$
t_7	BDG
t_8	BD
t_9	FG
t_{10}	AB
t_{11}	ABF
t_{12}	$ABFG$

表 3 分解后的 D_2

事务	项集合
t_1	{AF},{AG},{FG}
t_2	{FG}
t_4	{AB},{AD},{AF},{AG},{BD} {BF},{BG},{DF},{DG},{FG}
t_6	{AB},{BF},{BG},{DF},{DC},{FG}
t_7	{BD},{BG},{DG}
t_8	{BD}
t_9	{FG}
t_{10}	{AB}
t_{11}	{AB},{AF},{BF}
t_{12}	{AB},{AF},{AG},{BF},{BG},{FG}

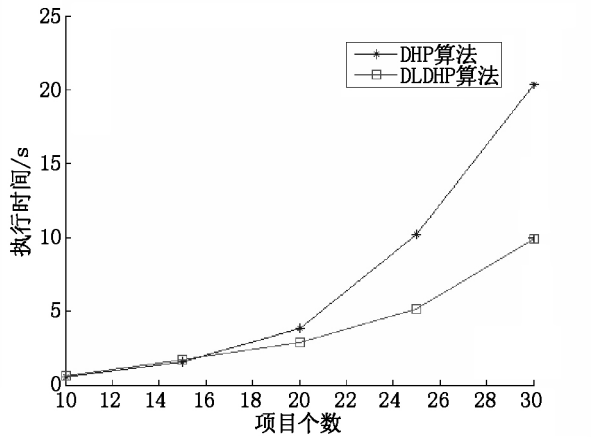

图 2　项目个数变化时的时间对比

通过图 2 可以看出,随着 D 中项目集个数的增加,2 种算法挖掘频繁 2-项集的时间逐渐增大,并且趋势基本保持一致. 项目个数在 15 以下时,2 种算法所用时间基本相同,随着项目个数的增加,DLDHP 算法所用的时间明显小于 DHP 算法,并且趋势越来越大. 因此,在挖掘大数据集时,DLDHP 算法的性能较 DHP 算法有很大提升.

2) 随机生成数据库 D,中事务个数为 1 000,给定的项目个数为 20,比较 DHP 和 DLDHP 算法随最小支持度变化在查找频繁 2-项集时所用时间. 比较结果如图 3 所示.

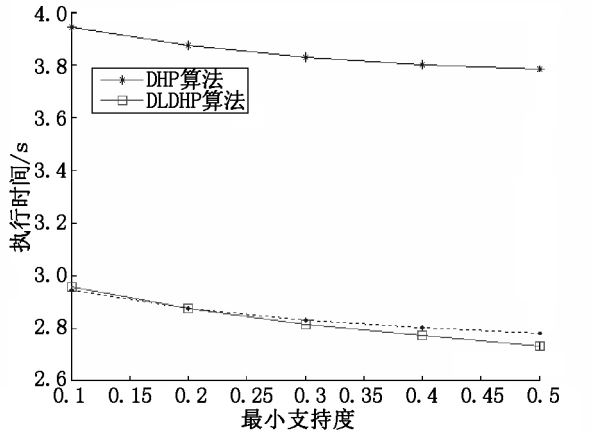

图 3　最小支持度变化时的时间对比

图 3 中的虚线是 DHP 算法的曲线向下平移后的图形,通过图 3 可以看出,随着最小支持度的变大,2 种算法挖掘频繁 2-项集的时间逐渐减小,并且随着最小支持度的不断变大,在最小支持度大于0. 2 时,DLDHP 算法所用时间的下降趋势大于 DHP 算法.

综合以上 2 个方面,本文提出的 DLDHP 算法效率确实优于 DHP 算法,这是因为 DLDHP 算法只需扫描一次数据库,通过适当增加空间内存,减少扫描数据库的次数,从而达到提高查找频繁项集的效率.

6　结束语

首先,本文概述了关联规则挖掘的相关知识及 DHP 算法的基本思想; 其次,引入 DLDHP 算法即动态链地址的关联规则挖掘算法,并介绍了其实现过程; 最后,实例表明了 DLDHP 算法比 DHP 算法效果更好.

DLDHP 算法处理的是既定的数据库,一旦有新的项目或者事务添加到数据库,都需要重新挖掘整个数据库,如何有效地处理动态的数据库是今后需要研究的一个方向.

7　参考文献

[1] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large database [EB/OL]. [2014-10-16]. http: // dx. doi. org/10. 1145/170035. 170072.

[2] Agrawal R, Imielinski T, Swami A. Database mining: a performance perspective [J]. Knowledge anddata engineering, IEEE Transactions, 1993, 5(6): 914-925.

[3] Gouda K, Zaki M J. Efficiently mining maximal frequent itemsets [EB/OL]. [2014-10-16]. http: // en. cnki. com. cn/Article_en/CJFDTOTAL-WXYJ200508006. htm.

[4] Park J S, Chen M S, Yu P S. An effective hash based algorithm for mining association rules [EB/OL]. [2014-10-22]. http: //dl. acm. org/citation. cfm? id = 223813.

[5] 卢云彬,曹汉强. 基于 Hash 表的关联规则挖掘算法的改进 [J]. 计算机技术与展,2007,17(6): 12-14.

[6] 张江,傅鹤岗. 基于关联规则的二维哈希算法的改进 [J]. 计算机工程与设计,2005,26(8): 2178-2179.

[7] 曾传璜,郑剑,邵利平. 多项集元素的 hash_tree 快速查找方法 [J]. 南方冶金学院学报,2003,24(5): 41-43.

[8] 崔玮,刘建伟,张其善. 基于最小完美哈希函数的数据挖掘算法 [J]. 华中科技大学学报: 自然科学版,2007,35(2): 47-49.

[9] 黄传明. 一种基于散列技术和事务压缩的关联规则挖掘算法 [J]. 计算机工程,2003,29(22): 117-118.

[10] 马如林. 一种哈希表快速查找的改进方法 [J]. 计算机工程与科学,2008,30(9): 66-68.

[11] 张继周. 基于哈希表的关联规则挖掘算法研究 [J]. 软件导刊,2013,12(7): 69-71.

[12] 张素兰. 一种基于事务压缩的关联规则优化算法 [J]. 计算机工程与设计,2006,27(18): 3450-3453.

The Algorithm of Dynamic Chain Address for Mining Association Rules Based on DHP

WU Heng, WU Genxiu*, MAO Linchuan, HUANG Mei
(School of Mathematics and Information Science, Jiangxi Normal University, Nanchang Jiangxi 330027, China)

Abstract: DLDHP algorithm solves DHP algorithm's problem that can't separate count for each candidate itemsets by using dynamic chain address to create Hash table. Increasing the table space and also deleting the nodes make Hash table won't occupy large space. Finally, frequent itemsets directly by the Hash table without scanning the database once again has been got. Examples show that DLDHP algorithm is effective and feasible.
Key words: DHP algorithm; DLDHP algorithm; Hash table; dynamic chain address

(责任编辑: 冉小晓)

(上接第 462 页)

The Metrics Method of Object-Oriented Class Complexity

QIN Huaibin, GUO Li
(College of Information Science and Technology, Shihezi University, Shihezi Xinjiang 832003, China)

Abstract: The metrics methods of within class from attribute, method, between attributes, between methods, between methods and attributes, the metrics methods of class structure from node strength, clustering coefficient, average path length, finally with the specific system class diagram as an example to measure empirically complexity class structure have been given. The result shows that the metrics method can measure better object-oriented class complexity.
Key words: object-oriented; class; complexity; metrics

(责任编辑: 冉小晓)