

基于压缩矩阵方式的 Apriori 改进算法

付 沙, 廖明华, 宋 丹

(湖南财政经济学院, 湖南 长沙 410205)

摘 要: 针对关联规则中 Apriori 算法的不足之处, 提出两种基于压缩矩阵方式的 Apriori 改进算法. 改进算法充分利用矩阵并对其进行压缩, 以大幅度减少扫描数据库的次数, 并提高频繁项集的生成效率, 从而有效提升算法的运算效率. 同时, 通过实例应用和算法性能分析证明所提出的两种改进算法都是有效的关联规则挖掘方法, 且比 Apriori 算法具有更好的性能.

关键词: 数据挖掘; 关联规则; Apriori 算法; 压缩矩阵; 频繁项集

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2012)06-0028-05

The Improved Apriori Algorithm Based on Compression Matrix Approach

FU Sha, LIAO Ming-hua, SONG Dan

(Hunan University of Finance and Economics, Changsha 410205, China)

Abstract: For the inadequacy of Apriori algorithm in association rules, this paper presents two methods of Apriori algorithm based on compression matrix approach. Improved algorithms make full use of the matrix and give compression on it, to significantly reduce the number of scans the database, and improve the generation efficiency of the frequent itemsets, and then improve the efficiency of the algorithm effectively. Meanwhile, the application example and algorithm performance analysis shows that the proposed two improved algorithms are efficient association rule mining method, and the properties are better than the Apriori algorithm.

Key words: data mining; association rules; Apriori algorithm; compression matrix; frequent itemsets

1 引言

最经典的关联规则挖掘算法是 Apriori 算法, 它由 Agrawal 等人于 1993 年提出, 是一种最有影响的挖掘布尔关联规则频繁项集的算法. Apriori 算法广泛应用于各种领域, 通过对数据的关联性进行分析和挖掘, 挖掘出的这些信息在决策制定过程中具有重要的参考价值^[1]. 然而, 对于大型数据库挖掘, 该算法存在着两个致命的性能瓶颈: 一是多次扫描事务数据库, 需要很大的 I/O 负载; 二是可能产生大量的候选项集, 需耗费大量的时间处理^[2]. 因此, 对 Apriori 算法予以改进具有一定的理论价值与现实意义.

由于数据库的规模通常较大, Apriori 算法从大量的候选项集生成频繁项集的过程耗时大, 严重影响了数据挖掘的效率. 频繁项集的生成作为影响关联规则挖掘的关键问题, 其计算复杂度决定了整个挖掘算法的时间复杂度^[3]. 基于数据挖掘的需求, 本文提出了两种基于压缩矩阵方式的 Apriori 改进算法——CM_Apriori (Compression Matrix Apriori) 算法, 以实现运算效率的提升.

2 关联规则概念和基本原理

关联规则挖掘问题的形式化描述如下: 设 $I = (I_1, I_2, \dots, I_m)$ 是项目的集合, 任务相关的数据 D 是数据库事务的集合, 其中每一个事务 T 都是项的集

收稿日期: 2011-09-29; 修回日期: 2011-11-26

基金项目: 湖南省情与决策咨询研究课题(201011BZZ100)

合,使得 $T \subseteq I$. 关联规则是形如 $A \Rightarrow B$ 的蕴含式,其中 $A \subset I, B \subset I$, 并且 $A \cap B = \emptyset$. 定义支持度 (support) 为 D 中包含 $A \cup B$ 的百分比,则 $\text{support}(A \Rightarrow B) = P(A \cup B)$; 置信度 (confidence) 为 D 中包含 A 的事务同时也包含 B 的百分比,则 $\text{confidence}(A \Rightarrow B) = P(B|A)$.

项的集合称为项集. 包含 k 个项的项集称为 k 一项集. 如果项集的出现频率大于或等于最小支持度与 D 中事务总数的乘积,则该项集满足最小支持度 min_support . 若其满足最小支持度,则称它为频繁项集. 频繁 k 一项集的集合记为项集 L_K .

大型数据库中挖掘关联规则包含两个步骤^[4]: (1) 找出所有频繁项集; (2) 由频繁项集产生强关联规则. 其中,第 (1) 步是核心,决定挖掘关联规则的总体性能. 因此,该研究主要针对第 (1) 步^[5]. 文中涉及关联规则挖掘算法中的两个性质:

性质 1 频繁项集的所有非空子集都必须也是频繁项集.

性质 2 非频繁项集的任一超集必定也是非频繁项集.

3 改进算法 CM_Apriori_1

3.1 CM_Apriori_1 算法的基本思想

(1) 扫描事务数据库 D 以构建布尔矩阵,对相同事务进行计数,使矩阵中的每列压缩存储一条或多条事务信息,且无重复列. 通过建立 AE 数组以存放每列的权值,若存在后续列与当前列的项目完全相同,则该列的权值数增加 1.

(2) 扫描矩阵并计算所有项目的支持度计数,若项目的支持度计数 support_count 小于最小支持度计数,则该项目对生成频繁 2 一项集无作用,可删除该项目对应的行向量.

(3) 对矩阵各行按位采用“与运算”,将该结果按位与 AE 数组中各列对应权值相乘,其加权和即为项集 $\{I_i, I_j\}$ 的支持度计数,若其值大于等于最小支持度计数,则应保存该行向量,否则予以舍弃. 此时,保存下来的行向量所对应的项集即频繁 2 一项集.

(4) 将保存下来的行向量按顺序重新组合得出矩阵 D^1 ,分别对 D^1 中的各列求和,若其和值与 AE 数组中对应权值之积小于最小支持度计数,则删除该列向量. 对剩下的列向量按顺序组合成矩阵 D^2 .

(5) 对 D^2 各行按位采用“与运算”,重复步骤 (3)、(4),生成频繁 k 一项集 $L_k (k \geq 3)$. 直到矩阵行数少于或等于一行时算法终止.

3.2 CM_Apriori_1 算法的实例应用

某事务数据库中有 10 个事务, $D = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}\}$, 对应的项目集 $I = \{I_1, I_2, I_3, I_4, I_5, I_6\}$, 如表 1 所示. 假设最小支持度 $\text{min_support} = 20\%$, 则最小支持度计数为 $\text{min_support} \times |D| = 20\% \times 10 \approx 2$.

表 1 事务数据库

TID	项 ID 的列表	TID	项 ID 的列表
T_1	I_1, I_2, I_4, I_5, I_6	T_6	I_2, I_3, I_5, I_6
T_2	I_2, I_3, I_5, I_6	T_7	I_3
T_3	I_2, I_3	T_8	I_2, I_4, I_5, I_6
T_4	I_3, I_4, I_5	T_9	I_3, I_4, I_5
T_5	I_3, I_5, I_6	T_{10}	I_5, I_6

(1) 扫描事务数据库 D , 生成布尔矩阵及 AE 数组.

$$AE[1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1]$$
$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

因为事务 T_2 和 T_6 以及 T_4 和 T_9 含有相同项目,则第 2、4 列的权值为 2,其余的权值均为 1.

(2) 生成频繁 2 一项集. 对矩阵各行按位采用“与运算”,过程如下: $D_1 \wedge D_2 = 10 \ 000 \ 000$, $\text{support_count}(I_1, I_2) = 1 \times 1 + 0 \times 2 + 0 \times 1 + 0 \times 2 + 0 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 1 = 1$.

由此,项集 $\{I_1, I_2\}$ 的支持度计数小于最小支持度计数,应舍去该行向量. 同理,项集 $\{I_2 I_3, I_2 I_4, I_2 I_5, I_2 I_6, I_3 I_4, I_3 I_5, I_3 I_6, I_4 I_5, I_4 I_6, I_5 I_6\}$ 的支持度计数大于等于最小支持度计数,保存该行向量;而项集 $\{I_1 I_3, I_1 I_4, I_1 I_5, I_1 I_6\}$ 的支持度计数小于最小支持度计数,舍去该行向量.

$$AE[1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1]$$

$$D^1 = \begin{matrix} D_2 \wedge D_3 \\ D_2 \wedge D_4 \\ D_2 \wedge D_5 \\ D_2 \wedge D_6 \\ D_3 \wedge D_4 \\ D_3 \wedge D_5 \\ D_3 \wedge D_6 \\ D_4 \wedge D_5 \\ D_4 \wedge D_6 \\ D_5 \wedge D_6 \end{matrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

所留存的行向量所对应的项集即为频繁 2 项集,按顺序重新组合得出矩阵 D^1 ,生成频繁 2 项集 $L_2 = \{I_2 I_3: 3, I_2 I_4: 2, I_2 I_5: 4, I_2 I_6: 4, I_3 I_4: 2, I_3 I_5: 5, I_3 I_6: 3, I_4 I_5: 4, I_4 I_6: 2, I_5 I_6: 6\}$.

(3)生成频繁 3 项集. 分别对 D^1 中的各列求和,若其和值与对应权值之积小于最小支持度计数,则删除该列向量. 由此,删除第 3、6、8 列,然后对剩余列向量按顺序组合成矩阵 D^2 . 限于篇幅,矩阵 D^2 不再罗列.

对 D^2 各行按位采用“与运算”, $D_2 \wedge D_3 \wedge D_4 = 00000$, $\text{support_count}(I_2, I_3, I_4) = 0$. 项集 $\{I_2, I_3, I_4\}$ 的支持度计数小于最小支持度计数,应舍去该行向量. 同理可得,项集 $\{I_2 I_3 I_5, I_2 I_3 I_6, I_2 I_4 I_5, I_2 I_4 I_6, I_2 I_5 I_6, I_3 I_4 I_5, I_3 I_5 I_6, I_4 I_5 I_6\}$ 对应的行向量应当保存,而项集 $\{I_3 I_4 I_6\}$ 对应的行向量予以舍去.

$$D^3 = \begin{matrix} AE[1 \ 2 \ 2 \ 1 \ 1] \\ D_2 \wedge D_3 \wedge D_5 \\ D_2 \wedge D_3 \wedge D_6 \\ D_2 \wedge D_4 \wedge D_5 \\ D_2 \wedge D_4 \wedge D_6 \\ D_2 \wedge D_5 \wedge D_6 \\ D_3 \wedge D_4 \wedge D_5 \\ D_3 \wedge D_5 \wedge D_6 \\ D_4 \wedge D_5 \wedge D_6 \end{matrix} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

此时,保存下来的行向量所对应的项集即频繁 3 项集,按顺序重新组合得矩阵 D^3 ,生成频繁 3 项集 $L_3 = \{I_2 I_3 I_5: 2, I_2 I_3 I_6: 2, I_2 I_4 I_5: 2, I_2 I_4 I_6: 2, I_2 I_5 I_6: 4, I_3 I_4 I_5: 2, I_3 I_5 I_6: 3, I_4 I_5 I_6: 2\}$.

(4)生成频繁 4 项集. 分别对 D^3 中的各列求和,并计算其和值与对应权值之积,执行删除第 4 列后,对剩下的列向量按顺序组合成矩阵 D^4 .

对 D^4 各行按位采用“与运算”, $D_2 \wedge D_3 \wedge D_5 \wedge D_6 = 0100$, $\text{support_count}(I_2, I_3, I_5, I_6) = 2$. 项集

$\{I_2, I_3, I_5, I_6\}$ 的支持度计数大于等于最小支持度计数,保存该行向量. 同理可得,项集 $\{I_2 I_4 I_5 I_6\}$ 对应的行向量应当保存,而项集 $\{I_2 I_3 I_4 I_5, I_3 I_4 I_5 I_6\}$ 对应的行向量予以舍去.

$$D^5 = \begin{matrix} AE[1 \ 2 \ 2 \ 1] \\ D_2 \wedge D_3 \wedge D_5 \wedge D_6 \\ D_2 \wedge D_4 \wedge D_5 \wedge D_6 \end{matrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

此时,保存下来的行向量所对应的项集即频繁 4 项集,按顺序重新组合得出矩阵 D^5 ,生成频繁 4 项集 $L_4 = \{I_2 I_3 I_5 I_6: 2, I_2 I_4 I_5 I_6: 2\}$.

(5)生成频繁 5 项集. 分别对 D^5 中的各列求和,计算其和值与对应权值之积,执行删除第 1、3、4 列后,对剩余列向量按顺序组合成矩阵 D^6 .

$$D^6 = \begin{matrix} AE[2] \\ D_2 \wedge D_3 \wedge D_5 \wedge D_6 \\ D_2 \wedge D_4 \wedge D_5 \wedge D_6 \end{matrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

对 D^6 各行按位采用“与运算”, $D_2 \wedge D_3 \wedge D_4 \wedge D_5 \wedge D_6 = 0$, $\text{support_count}(I_2, I_3, I_4, I_5, I_6) = 0$. 项集 $\{I_2, I_3, I_4, I_5, I_6\}$ 的支持度计数小于最小支持度计数,舍去该行向量,此时行向量为空,结束频繁项集的生成.

4 改进算法 CM_Apriori_2

4.1 CM_Apriori_2 算法的基本思想

(1)扫描事务数据库 D ,生成相对应的布尔矩阵. 矩阵利用行表头标记项目,列表头标记事务. 根据提供的最小支持度计数化简矩阵,若某个项目出现在所有事务中的次数总和小于最小支持度计数,则删除该项目对应的列.

(2)经过上述处理的矩阵称为初始矩阵,对初始矩阵的所有事务中寻找包含最多项的频繁 k 项集^[6]. 寻找所有事务中包含项目数最多的事务,项目数记为 k .

①若包含项目数为 k 的事务个数小于最小支持度计数,则说明包含最多项的频繁项集不可能包含 k 个项目,所以在初始矩阵中再次寻找包含项目数仅次于 k 的事务(仍标记为 k),直到符合要求的事务个数大于等于最小支持度计数为止.

②若包含项目数为 k 的事务个数大于等于最小支持度计数,则根据 k 个项目的要求对矩阵进行化简. 对于所有事务中包含项目数少于 k 的可从矩阵中删除该事务对应的行. 对行的处理会引起列的变化,对列进行处理的又可能会对行产生影响,所以重复步骤(1)、(2),直到矩阵已无变化则停止处理.

(3)结合得出的矩阵,选择矩阵中的某个事务 T ,接着在事务 T 中选取包含 k 个项目组成一个项目集,通过“与运算”等操作计算其支持度计数.若支持度计数不小于最小支持度计数,则此项目集是频繁 k —项集的一个子集.同理,对事务 T 中的其他可能组成包含 k 个项目的项目集进行计算,直到计算完毕事务 T 中所有组成包含 k 个项目的项目集,此时可以删除该事务对应的行.重复进行步骤(3),继续处理其他的各个事务,对于已经处理过的项目集无需再次计算,直到矩阵为空矩阵时该步骤结束^[7].

(4)结合初始矩阵根据性质 1 生成自频繁 2—项集到频繁 $k-1$ 项集的所有频繁项集.

4.2 CM_Apriori_2 算法的实例应用

实例同上文 3.2 节.

(1)扫描事务数据库 D ,生成相对应的布尔矩阵.因为最小支持度计数为 2,而项目 I_1 出现在所有事务中的次数总和小于最小支持度计数,可删除 I_1 对应的列以简化矩阵,如表 2 所示.

表 2 初始矩阵

	I_2	I_3	I_4	I_5	I_6
T_1	1	0	1	1	1
T_2	1	1	0	1	1
T_3	1	1	0	0	0
T_4	0	1	1	1	0
T_5	0	1	0	1	1
T_6	1	1	0	1	1
T_7	0	1	0	0	0
T_8	1	0	1	1	1
T_9	0	1	1	1	0
T_{10}	0	0	0	1	1
共持度计数	5	7	4	8	6

(2)对表 2 进行化简,可以得出包含最多项的频繁项目集.其具体操作步骤如下:

①从表 2 中找出所有事务中包含项目数最多的事务,包含项目数为 4 的事务为 T_1 、 T_2 、 T_6 、 T_8 共 4 项,其个数大于等于最小支持度计数 2,可根据要求对矩阵进行化简.

②对于所有事务中包含项目数少于 4 的可删除该事务对应的行,如表 3 所示.

③在事务 T_1 中,包含 4 个项目的项集只有一种组合,即 $\{I_2, I_4, I_5, I_6\}$,所以将 I_2 、 I_4 、 I_5 、 I_6 这 4 列进行“与运算”,将其结果相加,得出项集 $\{I_2, I_4,$

$I_5, I_6\}$ 的支持度计数为 2.所以,该项集是包含最多项的频繁项集之一,即:频繁 4—项集的一个子集.

表 3 化简后的矩阵

	I_2	I_3	I_4	I_5	I_6
T_1	1	0	1	1	1
T_2	1	1	0	1	1
T_6	1	1	0	1	1
T_8	1	0	1	1	1
支持度计数	4	2	2	4	4

因为事务 T_1 中不再拥有其他的包含 4 个项目的项集,所以删除事务 T_1 对应的行.同理,在事务 T_2 中,包含 4 个项目的项集也只有一种组合,即 $\{I_2, I_3, I_5, I_6\}$,经处理后可得该项集也是包含最多项的频繁项集之一.然后,删除事务 T_2 对应的行后对矩阵进行化简,对已经处理过的项集无需再行计算,得到的是空矩阵.综上,说明包含最多项的频繁项集为 $\{I_2, I_4, I_5, I_6\}$ 和 $\{I_2, I_3, I_5, I_6\}$.

(3)已经得出包含最多项的频繁项集为 $\{I_2, I_4, I_5, I_6\}$ 和 $\{I_2, I_3, I_5, I_6\}$,根据文中性质 1,可得部分频繁项集:频繁 2—项集 $\{I_2 I_4, I_2 I_5, I_2 I_6, I_4 I_5, I_4 I_6, I_5 I_6, I_2 I_3, I_3 I_5, I_3 I_6\}$,频繁 3—项集 $\{I_2 I_4 I_5, I_2 I_4 I_6, I_2 I_5 I_6, I_4 I_5 I_6, I_2 I_3 I_5, I_2 I_3 I_6, I_3 I_5 I_6\}$.所以不用再计算其支持度计数,也大幅度减少了连接和剪枝的操作.

①生成频繁 2—项集 L_2 :根据表 2 和已得出的部分频繁 2—项集,可得候选 2—项集为 $\{I_3 I_4\}$,仅为 1 个项集,而按照 Apriori 算法则要计算 10 个项集的支持度.结合表 2,对项集 $\{I_3 I_4\}$ 采用“与运算”计算得出支持度计数为 2.由此,最终生成频繁 2—项集 $L_2 = \{I_2 I_4, I_2 I_5, I_2 I_6, I_4 I_5, I_4 I_6, I_5 I_6, I_2 I_3, I_3 I_5, I_3 I_6, I_3 I_4\}$.

②生成频繁 3—项集 L_3 :化简表 2,删除项目数少于 3 的事务的对应行,如表 4 所示.

根据表 2 和已得出的部分频繁 3—项集,可得候选 3—项集为 $\{I_2 I_3 I_4, I_3 I_4 I_5, I_3 I_4 I_6\}$,仅有 3 个项集,而按照 Apriori 算法则要计算 10 个项集的支持度.结合表 4,对项集 $\{I_2 I_3 I_4, I_3 I_4 I_5, I_3 I_4 I_6\}$ 采用“与运算”计算其支持度,得出大于等于最小支持度计数的项集为 $\{I_3 I_4 I_5\}$.所以,生成频繁 3—项集 $L_3 = \{I_2 I_4 I_5, I_2 I_4 I_6, I_2 I_5 I_6, I_4 I_5 I_6, I_2 I_3 I_5, I_2 I_3 I_6, I_3 I_5 I_6, I_3 I_4 I_5\}$.

表 4 删除部分行与列的矩阵

	I_2	I_3	I_4	I_5	I_6
T_1	1	0	1	1	1
T_2	1	1	0	1	1
T_3	0	1	1	1	0
T_5	0	1	0	1	1
T_6	1	1	0	1	1
T_8	1	0	1	1	1
T_9	0	1	1	1	0
支持度计数	4	5	4	7	5

5 算法实验与分析

为了检验改进算法的性能,将两种改进算法与 Apriori 算法进行比较,对最小支持度为 20%,但在不同数据库规模下各算法运行时间进行对比.实验环境:CPU 为 Intel Pentium 42.66 GHz、内存 1 GB 和 Windows XP 专业版操作系统.程序采用 Delphi7.0 实现,测试数据库为 Microsoft SQL Server 2005 所带的 foodmart.mdb 数据库,样本数据包括 1 000~6 000 条记录且逐次递增 1 000 条共 6 种情况^[8].实验结果如图 1 所示.

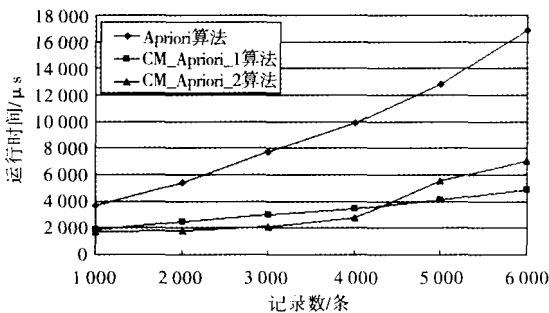


图 1 不同数据库规模下三种算法的性能比较

由图中可以看出,当数据库规模较小时,三种算法的运行时间差距不大,但随着数据库规模的增大,Apriori 算法的运行时间基本呈现线性增加;CM_Apriori_1 算法的运行时间明显大幅低于 Apriori 算法,且性能稳定;而 CM_Apriori_2 算法的运行时间增长趋势在样本数据为 4 000 条记录之前更加缓慢,但随着样本数据的增加,其呈现小幅上升趋势.

算法的运行时间与其执行过程有着密切的关系:Apriori 算法每生成一次频繁项集就需要扫描一次数据库,加之其在处理时会产生大量的候选项集,若数据库规模越大,则其时间消耗量也就越大.改进

算法 CM_Apriori_1 与传统的 Apriori 算法相比,只需扫描一次数据库并生成布尔矩阵,通过对相同事务的计数来压缩数据库规模,并且对矩阵的行向量进行“与运算”直接生成频繁项集,省去 Apriori 算法中的连接和剪枝步骤,无需生成候选项集.另外,通过对矩阵的有效裁剪,大大减少了求解高次频繁项集的运行时间,提高了算法的运算效率.改进算法 CM_Apriori_2 同样采用矩阵方式表示事务数据库,对矩阵进行处理找出包含最多项的频繁 k -项集,然后利用矩阵生成自频繁 2-项集到频繁 $k-1$ 项集的所有频繁项集.由频繁 k -项集可直接得到部分频繁 $k-a$ 项集($a < k$),将降低候选项集计算支持度计数、连接和剪枝步骤的开销,从而节省大量时间.此外,在求解频繁项集时,该算法充分利用矩阵,采用“与运算”的方法取代到数据库中查找的过程,提高了计算效率.

6 结束语

本文对关联规则挖掘中的 Apriori 算法进行了分析,提出两种基于压缩矩阵方式的 Apriori 改进算法,并对其性能进行了分析和对比.实验结果表明,所提出的两种改进算法有效地压缩了搜索空间,减少了扫描数据库的次数,提高了频繁项集的生成效率,其性能较传统的 Apriori 算法有显著提高.

参考文献:

[1] Stephen Haag, Maeve Cummings, Donald J brey. 信息时代的管理信息系统[M]. 严建援,译. 北京:机械工业出版社,2005: 95-99.

[2] 崔贯勋,李梁,王柯柯,等. 关联规则挖掘中 Apriori 算法的研究与改进[J]. 计算机应用, 2010,30(11):2952-2955.

[3] 李晓虹,尚晋. 一种改进的新 Apriori 算法[J]. 计算机科学,2007,34(4):196-198.

[4] 吴绍函,余昭平. 基于矩阵的关联规则挖掘算法[J]. 计算机工程,2008,34(23):31-33.

[5] 涂明,张公让,程业媛. 一种高效的关联规则增量式更新算法[J]. 微电子学与计算机,2010, 27(9):56-60.

[6] 王锋,李勇华,毋国庆. 基于矩阵的改进的 Apriori 算法[J]. 计算机工程与设计,2009, 30(10):2435-2438.

[7] 刘培奇,卢麟,廖福燕,等. 基于一次性数据库访问策略的关联规则挖掘算法的研究[J]. 微电子学与计算机,2010,27(12):22-25.

[8] 洪梓璇,边馥苓. 基于支持度矩阵的 Apriori 改进算法[J]. 武汉大学学报·信息科学版, 2008,33(12):1246-1249.

(下转第 36 页)

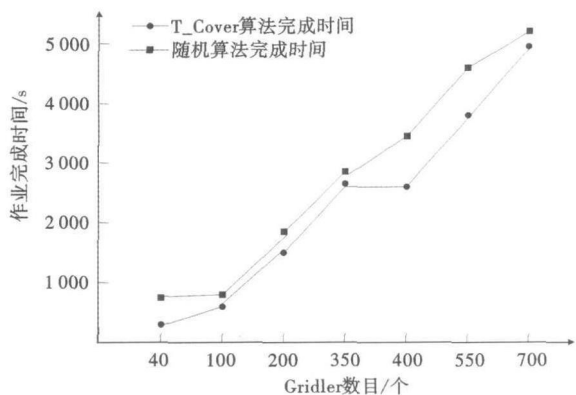


图 2 两种算法实际花费的时间

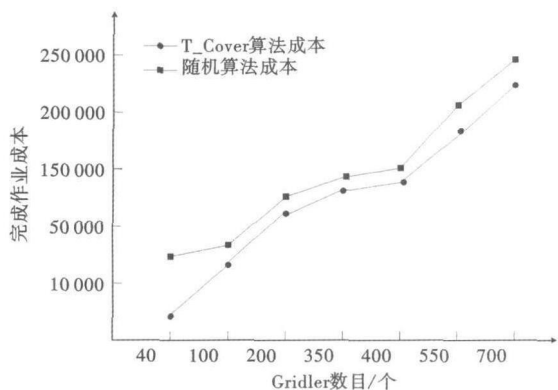


图 3 两种算法实际花费的成本

5 结束语

本文提出的基于最优投资决策模型的资源分配算法,综合考虑作业执行费用和执行时间两方面的因素,根据资源各性能参数值归一化为决策矩阵,从而得到可以减小作业执行时间及成本的分配策略.通过与随机算法的对比结果看出,该算法具有较高的效率和良好的稳定性.

参考文献:

- [1] 王元卓,林闯,杨扬,等. 网格服务可管理性模型及策略研究[J]. 计算机学报, 2008,31(10):1716-1726.
- [2] 丁箭,陈国良,顾钧. 计算网格环境下一个统一的资源映射策略[J]. 软件学报, 2002,13(7):1303-1308.
- [3] 元雪冬,李霞,梁鸿. 计算网格中动态负载平衡策略研究[J]. 微电子学与计算机, 2009,26(5):21-24.
- [4] 钟伯成,杜兆芳,胡靖. 基于 Agent 的计算网格资源分配[J]. 微电子学与计算机, 2006,23(4):166-167.
- [5] 何军,王颀. 多线程处理器资源分配策略[J]. 计算机工程, 2008,34(15):283-285.
- [6] 丛珊. 基于 GridSim 的网格资源模拟技术的研究[D]. 哈尔滨:哈尔滨工业大学, 2006.

作者简介:



郑晓薇 女,(1957—),教授,CCF 高级会员(E20-0008655S). 研究方向为网格系统、并行计算、计算机决策支持系统.



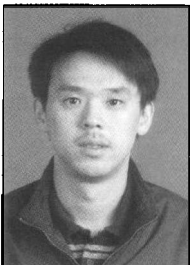
刘海君 女,(1986—),硕士研究生,CCF 学生会员(E200018408G). 研究方向为网格计算、计算机决策支持系统.



张建强 男,(1981—),硕士研究生,助教,CCF 学生会员(E200018401G). 研究方向为并行计算.

(上接第 32 页)

作者简介:



付沙 男,(1980—),硕士,讲师. 研究方向为数据挖掘、网络与信息安全.

廖明华 男,(1977—),硕士,讲师. 研究方向为计算机网络.
宋丹 男,(1976—),博士研究生,讲师. 研究方向为人工智能、免疫算法.