一种基于矩阵的频繁项集更新算法 *

徐嘉莉[1], 陈 佳[2]
(1. 成都大学 电子信息工程学院, 成都 610106; 2. 电子科技大学 计算机科学与工程学院, 成都 610054)

摘 要: 针对相关算法在处理频繁项集更新时所存在的问题,提出了一种基于矩阵的频繁项集更新算法。该算法首先以时间为基准将更新后的数据库分为原数据库和新增数据库,分别将它们转换为 0-1 矩阵,通过矩阵裁剪、位运算产生新增频繁项集,并利用已有频繁项集更新原有频繁项集。实验仿真结果不但证明了该算法的可行性和高效性,而且还证明了它适合大型、稠密性数据库的频繁项集更新。

关键词: 数据挖掘; 关联规则; 频繁项集; 更新
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2010)03-0837-04
doi:10.3969/j.issn.1001-3695.2010.03.008

Updating algorithm based on matrix for mining frequent item sets

XU Jia-li[1], CHEN Jia[2]
(1. *School of Electronic & Information Engineering, Chengdu University, Chengdu* 610106, *China*; 2. *School of Computer Science & Engineering, University of Electronic Science & Technology of China, Chengdu* 610054, *China*)

Abstract: Aiming at updating problems of frequent item sets, this paper proposed an updating algorithm based on matrix (UABM) for mining frequent item sets. Divided the updated database into original database and new one based on time. Converted these databases into matrixes. Got the new frequent sets by matrix cropping and the bit operation, and updated the gotten frequent item sets on gotten ones. The experiments show the algorithm is not only feasible and efficient but also fit to update freguent item sets for a large-scale and dense data base.

Key words: data mining; association rules; frequent item sets; updating

关联规则挖掘是数据挖掘领域中的一个重要研究课题,Agrawal 等人于 1993 年提出挖掘顾客交易数据库中项集间的关联规则问题后,至今已有很多高效的关联规则挖掘算法[1~5],但这些算法大多针对静态数据和固定的最小支持度。而在实际的挖掘过程中,用户往往需要对最小支持度进行不断调整来寻找真正感兴趣的规则;另一方面事务数据库的数据随时间而变化,如在线提供的实时服务、大型商场的购物清单所提供的数据都是动态变化的,这些使得当前已发现的关联规则可能不再有效,也可能还存在新的有效规则有待进一步发现。因此,有必要设计高效的算法来更新维护已挖掘出的关联规则。

关联规则挖掘分两步:产生频繁项集;对每个频繁项集产生所有大于最小置信度的规则。由于第二步相对较易,关联规则挖掘的研究重点放在了第一步,同样地,关联规则的更新重点也就转为频繁项集的更新。频繁项集的更新一般分为三种:a)最小支持度不变、数据库记录数发生变化;b)最小支持度发生变化、数据库记录数不变;c)最小支持度、数据库记录数都发生变化。

目前已有一些典型的频繁项集更新算法出现。FUP[6]、IFUP[7]都是在最小支持度给定的情况下,当数据库记录数发生变化时,在新增事务中寻找频繁项集,然后结合已有频繁项集挖掘出新的频繁项集。虽然它们在挖掘过程中利用已有知识避免原有频繁项集的重复挖掘,在一定程度上提高了算法的效率,但由于它们都是基于 Apriori 框架的,需要多次扫描数据库并产生庞大的候选项集,其效率并没有得到充分的提高。IUA[8]则是针对频繁项集更新的第二种情况,由于它仍然基于 Apriori 框架,仍然需要多次扫描数据库并产生庞大的候选项集。人们对第三种情况的研究很少,基于 FP-tree 的 IM[9]算法虽然可以处理这种情况,但是该算法采用复杂的数据结构 FP-tree,需进行复杂的操作,算法过程需要庞大的存储空间。

本文提出了一种基于矩阵的频繁项集更新算法(UABM)。该算法不但能处理数据库记录数增大而最小支持度不变、数据库记录数不变而最小支持度改变以及数据库记录数增大而最小支持度也改变等条件下的频繁项集更新,而且还避免了对原数据库中已有频繁项集的重复挖掘,并且摆脱了 Apriori 框架的束缚,只需扫描一遍数据库、不产生庞大的候选项集;在挖掘过程中对矩阵进行有效的裁剪还能减少空间和时间的开销。

1 相关定义与性质

数据库 D 是事务的集合,即 $D=\{T_1,T_2,\cdots,T_m\}$,每个事务 T 是项的集合,项集合 $I=\{I_1,I_2,\cdots,I_n\}$。定义变量 i、j,其取值范围分别为 $1\leqslant i\leqslant m$,$1\leqslant j\leqslant n$,则有以下定义:由于定义描述中数学符号较多,为了便于读者阅读,定义 1~3 用表格的形式进行描述,如表 1 所示。

收稿日期: 2009-07-28; **修回日期**: 2009-08-29 **基金项目**: 国家"863"计划资助项目(2007AA01Z443);华为软件技术有限公司高校合作资助项目(YBIN2007243)

作者简介:徐嘉莉(1969-),女,四川成都人,讲师,硕士,主要研究方向为数据挖掘、网格计算(lotussunnyx@163.com);陈佳(1980-),女,四川南充人,讲师,博士,主要研究方向为数据挖掘、网格计算.

表 1　定义 1 ~ 3 的描述

定义	项集	向量表示	支持数
定义 1	1-项集 $\{I_j\}$	$A_j=(a_{1j},a_{2j},\cdots,a_{mj})^{\mathrm{T}}$	$\sum_{i=1}^{m}a_{ij}$，当 $I_j\notin T_i$ 时，$a_{ij}=0$；当 $I_j\in T_i$ 时，$a_{ij}=1$
定义 2	2-项集 $\{I_i,I_j\}$	$A_{ij}=A_i\wedge A_j=(a_{1i}\wedge a_{1j},a_{2i}\wedge a_{2j},\cdots,a_{mi}\wedge a_{mj})^{\mathrm{T}}$	$\sum_{k=1}^{m}(a_{ki}\wedge a_{kj})$
定义 3	k-项集 $\{I_{i1},I_{i2},\cdots,I_{ik}\}$	$A_{i1i2\cdots ik}=A_{i1}\wedge A_{i2}\wedge\cdots\wedge A_{ik}$	$\sum_{q=1}^{m}(a_{qi_1}\wedge a_{qi_2}\wedge\cdots\wedge a_{qi_k}$ 其中 $1\leqslant i_k\leqslant n$

定义 4　D 的布尔矩阵记为

$$A=\begin{Bmatrix}0 & 1 & 2 & \cdots & n\\ \begin{matrix}\text{ű}\\ a_{i0}\\ \text{ű}\end{matrix} & A_1 & A_2 & \cdots & A_n\\ 0 & a_{(m+1)1} & a_{(m+1)2} & \cdots & a_{(m+1)n}\end{Bmatrix}$$

其中：A 的下标从 0 开始，$a_{0j}(1\leqslant j\leqslant n)$ 代表项 I_j 的 ID 号 j，$a_{(m+1)j}(1\leqslant j\leqslant n)$ 为项 I_j 的支持数，$a_{i0}(1\leqslant i\leqslant m)$ 为事务 T_i 包含的项数。

性质 1　频繁 k-项集存在的判定方法。如果 k-项集 $\{I_{i1},I_{i2},\cdots,I_{ik}\}$ 的支持数不小于最小支持数，则 $\{I_{i1},I_{i2},\cdots,I_{ik}\}$ 就是频繁 k-项集。

性质 2　设 L_k 为 k-频繁项集，若 $T\subseteq I$，T 的支持数 $|T|=k$，则求 $k+1$ 维支持度时可以从布尔矩阵中删除事务 T。

证明　由布尔“与”运算的性质知，仅当参加“与”运算的元素的值全为 1 时，运算结果才为 1。所以当 T 中包含项的数目小于 $k+1$ 时，T 的元素值至少有一个为 0。因而 k-项集对于生成频繁 $(k+1)$-项集是没有用的，所以在求 $(k+1)$ 项集的支持度时，可以删除布尔矩阵中项的数目等于 k 的行。

性质 3　设 X_k 是 k-项集，如果所有 $(k-1)$-频繁项集 X_{k-1} 中包含 X_k 的 $(k-1)$-项集的个数小于 k，则 X_k 不可能是 k 维最大频繁项集[10]。

设 D 由 n 个子集 $D_1,D_2,\cdots,D_n$ 构成，$D_1,D_2,\cdots,D_n$ 分别对应 n 段连续时间间隔下所挖掘的事务集；$|D|$、$|D_i|$ 分别为 D、$D_i(i=1,2,\cdots,n)$ 中事务的总数；s 为原最小支持度，s' 为新的最小支持度；L_D 和 L_D^K 分别为 s 条件下的所有维频集的集合和 $k(k=1,2,\cdots,n'_1)$ 维频集；$L_{D\text{-new}}$ 和 $L_{D\text{-new}}^k$ 分别为 s' 条件下所有维频集的集合和 $k(k=1,2,\cdots,n'_1)$ 维频集。则有以下性质：

性质 4　项集 I 在 n 个时段都为频繁项集，则它在全时段必为频繁项集。

性质 5　项集 I 只要在其中一个时段不为频繁项集，则它在全时段就不一定为频繁项集。

性质 6　若项集 I 是全局频繁项集，则它至少在其中一个时段局部频繁。

证明　假设 I 在任意时段上都不频繁，即 $|I|_{D1}<s\times|D_1|$，$|I|_{D2}<s\times|D_2|,\cdots,|I|_{Dn}<s\times|D_n|$，则有 $|I|_{D1}+|I|_{D2}+\cdots+|I|_{Dn}<s\times(|D_1|+|D_2|+\cdots+|D_n|)$，即 I 在 D 中也不频繁。这与已知条件项集 I 是全局频繁项集矛盾，所以该假设不成立。此题得证。

性质 7　如果 $s>s'$，则 $L_D^k\subseteq L_{D\text{-new}}^k$，$L_D\subseteq L_{D\text{-new}}$，且 $n_1\leqslant n'_1$；如果 $s<s'$，则 $L_D^k\supseteq L_{D\text{-new}}^k$，$L_D\supseteq L_{D\text{-new}}$，且 $n_1\geqslant n'_1$。

证明　对于任意 $I\in L_D^k$，有 $|I|_D/|D|\geqslant s$。如果 $s>s'$，则 $|I|_D/|D|\geqslant s'$，即 $I\in L_{D\text{-new}}^k$，故有 $L_D^k\supseteq L_{D\text{-new}}^k$ 成立，从而推得 $L_D\subseteq L_{D\text{-new}}$ 和 $n_1\leqslant n'_1$ 成立。同样，对于任意 $I\in L_{D\text{-new}}^k$，有 $|I|_D/|D|\geqslant s'$。如果 $s<s'$，则 $|I|_D/|D|\geqslant s$，即 $I\in L_D^k$，故有 $L_D^k\supseteq L_{D\text{-new}}^k$ 成立，从而推得 $L_D\supseteq L_{D\text{-new}}$ 和 $n_1\geqslant n_1'$ 成立。

2　相关子算法

2.1　矩阵生成算法

设 $D=\{T_1,T_2,\cdots,T_m\}$，项集 $I=\{I_1,I_2,\cdots,I_n\}$，由定义 1、4 可构造布尔矩阵 $A_{(m+2)\times(n+1)}$。其算法核心如下：

```
//矩阵生成算法
a[0][0] = 0; a[m+1][0] = 0;
for (j = 1;j < = n;j + + )
  {a[0][j] = j;// 填充各项的 ID 号
    a[m+1][j] = 0;}
for(i = 1;i < = m;i + + )
    {a[i][0] = 0;
    for(j = 1;j < = n;j + + )
    { //如果 Ij 在 Ti 中存在,则 a[i][j] 为 1,否则为 0
      if (Ij in Ti) { a[i][j] = 1;
      a[i][0] + + ;//统计事务 Ti 包含项的个数
      a[m+1][j] + + ;//统计项 Ij 的支持记数 }
      else a[i][j] = 0;} }
```

2.2　矩阵压缩算法

已知 s，矩阵 $A_{(m+2)\times(n+1)}$。其中 m 代表事务数，n 代表项数，则矩阵压缩算法的核心如下：

```
min_sup_num = s * m;//最小支持记数 min_sup_num
do {// 删除不符合条件的列
   for(j = 1;j < = n;j + + )
   {if a[m+1][j] < min_sup_num
   { for(i = 1;i < = m;i + + )
     if(Ij in Ti) a[i][0] - - ;// 修改第 0 列中各元素值
       delete 第 j 列 //由性质 1、3;
     }}
   for all Ij ∈ Lk-1   // Lk-1 为频繁(k-1)-项集
     if Ij 在频繁(k-1)-项集中出现的次数小于(k-1)
     { for(i = 1;i < = m;i + + )
       if(Ij in Ti) a[i][0] - - ;
       delete j 列;//由性质 3
     }
   //删除不符合条件的行
   for(i = 1;i < = m;i + + )
   if a[i][0] < k
     { for(j = 1;j < = n;j + + )
       if(Ij in Ti) a[m+1][j] - - ;// 修改第 m+1 行各元素值
       delete 第 i 行 //由性质 2,删除第 0 列中值小于 k 的元素所对应的行;}
   }while 无行、列可删除。
```

2.3　频繁 k-项集生成算法

当 $k>1$ 时，若矩阵经裁剪后为 $A_{r\times p}(k<p)$，对 A 的 1 ~ $p-1$ 列对应的 ID 号进行 k 维组合，则有 $s=C_{p-1}^k$ 个组合对。用数组 $B_{s\times(k+1)}$ 来记录各个组合对（下标从 0 开始，第 k 列代表各组合对的支持数）。对每个组合对中的前 k 个 ID 号对应的列向量求 k 维支持数，由性质 1，如果其 k 维支持数不小于 min_sup_num，则该组合对所对应的项集为 k-频繁项集。其算法核心如下：

```
Lk = ∅; min_sup_num = s * m;//m 为 D 中的事务数
for( x = 0;x < s;x + + )
{ M = ∅;
  sup_num = 0;// k-项集支持数 sup_num 初值为 0
  for( i = 1;i < r - 1;i + + )
     { w = 1;
```

```
    for(j=0;j<k;j++)
    { for(b=1;b<p;b++)
      if B[x][j]=a[0][b] break;
      w=w*a[i][b];//"与"运算
      if (w==0) break; }
    sup_num= sup_num +w;// 计算 k-项集支持数 }
    if sup_num >=min_sup_num //求 k-频繁项集
  {for(j=0;j<k;j++)
  {t=B[x][j]; M=M∪{it}; }
  Lk= Lk∪M;}}
```

2.4 频繁 *k*-项集更新算法

当 $k>1$ 时,若矩阵经裁剪后为 $A_{r\times p}(k<p)$,对 A 的第 1 ~ $p-1$ 列对应的 ID 号进行 k 维组合,则有 $s=C_{p-1}^{k}$ 个组合对。用数组 $B_{s\times(k+1)}$ 来存放各个组合对(下标从 0 开始,第 k 列代表各组合对的支持数)。从 $B_{s\times(k+1)}$ 中删除 D 中原有频繁 k-项集所对应的组合对,对余下的每个组合对求 k 维支持数,k 维支持数不小于 min_sup_num 的组合对所对应的项集则为新增的 k-频繁项集。其算法核心如下:

```
min_sup_num=s*m;
从 Bs×(k+1) 中删除原频繁 k-项集所对应的组合对,得到 Bf×(k+1);
for(x=0;x<f;x++)
{M=∅;
 sup_num=0;// k-项集支持数初值为 0
 for(i=1;i<r-1;i++)
    { w=1;
      for(j=0;j<k;j++)
      { for(b=1;b<p;b++)
        if B[x][j]=a[0][b] break;
        w=w*a[i][b];//"与"运算
        if (w==0) break; }
      sup_num= sup_num +w;// 计算 k-项集支持数 }
  if sup_num >=min_sup_num //求 k-频繁项集
    { for(j=0;j<k;j++)
      { t= B[x][j]; M=M∪{it}; }
    Lk= Lk∪M;}}
```

3 UABM 算法思想

UABM 将更新后的数据库从物理上分为两个完全独立的数据库分别进行处理[6,7,11],而不是将数据库从逻辑上划分成互不相交的块。这主要是因为后者虽然具有只需把所处理的分块放入主存、减轻内存需求并为算法的并行处理提供可能的优点,但由于这些分割的块仍然属于同一个数据库,算法的并行程度不如前者高,算法的效率也较前者低。

UABM 以时间为基准划分数据库,比如在 2009 年 3 月 10 日 10 点设置时间间隔为 15 天,则 2009 年 3 月 10 日 10 点以前的数据库记为原数据库 DB;在 2009 年 3 月 10 日 10 点后的 15 天内新增的事务构成新增数据库 db。显然 DB 和 db 的数据库模式是同构的。为了满足实际挖掘环境中用户对响应时间的需求,当用户设置的时间间隔一满足或最小支持度一旦发生变化,UABM 就会自动运行,以更新维护频繁项集。

记 DB 的事务总数为 $|DB|$,频繁 k-项集为 L_{DB}^{k};db 的事务总数为 $|db|$,频繁 k-项集为 L_{db}^{k};s 为原最小支持度,s'为新最小支持度;在 s'条件下,DB 中的频繁 k-项集为 $L_{DB\text{-}new}^{k}$;更新后的数据库记为 DB + db,其事务总数为 $|DB+db|$,频繁 k-项集为 L_{DB+db}^{k},则 UABM 算法核心描述如下:

```
输入:DB,L^k_DB,db,s,s'
输出:L^k_DB+db
a)生成 db 的 L^k_db。
if |db|==0   L^k_db=∅ //数据库记录数不变的情况
else {
   调用矩阵生成算法构造 A_db;
   调用矩阵压缩算法对 A_db 进行压缩生成 A^k_db;
   调用频繁 k-项集生成算法得到 L^k_db
   }
b)更新 DB 的频繁 k-项集。
if s=s' L^k_DB-new=L^k_DB;
else if s<s' L^k_DB-new=L^k_DB -所有支持度小于 s 的项集//由性质 6 和 7
      else
      { //由性质 6、7 调用矩阵生成算法生成 A_DB;
      产生新的频繁 k-项集 L^k_DB-new(k≥2):
      if L^k_DB≠∅ L^k_DB-new=L^k_DB;
      else L^k_DB-new=∅;
      调用矩阵压缩算法生成 A^k_DB;
      调用频繁 k-项集更新算法产生新增的频繁 k-项集,并添加到 L^k_DB-new 中}
c)当 L^k_DB-new≠∅ or L^k_db≠∅时,将 L^k_DB-new 与 L^k_db 中的项集 I 进行比较(k≥1),产生 L^k_DB+db。
if 项集 I 在 DB 中为 k-频繁项集
   if 项集 I 在 db 中为 k-频繁项集
   {L^k_DB+db=L^k_DB+db+I; |I|_DB+db=|I|_DB+|I|_db//性质 4}
   else { //性质 5
        从 B^k_db 中 I 所在行最后一个元素得|I|_db;
        if |I|_DB+|I|_db>=s'×|DB+db|
        L^k_DB+db=L^k_DB+db+I}
else if I 在 db 中是 k-频繁项集
   {//性质 5
   由 B^k_DB 中 I 所在行最后一个元素得到|I|_DB;
   if |I|_DB+|I|_db>=s'×|DB+db|
     L^k_DB+db=L^k_DB+db+I}
```

4 算法实现

4.1 实例分析

本文仅分析数据库记录数增加、最小支持度变小的情况。

假设在 2009 年 3 月 10 日 10 点设置时间间隔为 15 天,则在 2009 年 3 月 10 日 10 点以后的 15 天内新增的事务构成 db,如图 1 所示,$|db|=5$;2009 年 3 月 10 日 10 点以前的数据库记为 DB,如图 2 所示,$|DB|=10$;s 为 20%,s′为 10%;DB 中原最小支持数为 $20\%\times10=2$,新的最小支持数为 $10\%\times10=1$。而 db 中新的最小支持数为 $10\%\times5=0.5$,DB + db 中的最小支持数为 $10\%\times(10+5)=1.5$。

1)求 L_{db}^{k}　如图 1 所示,当 s'为 10% 时,调用矩阵生成算法生成 A_{db};假设求频繁 2-项集,调用矩阵压缩算法将 A_{db} 转换成 A_{db}^{2}。调用频繁 k-项集生成算法生成 $L_{db}^{2}=\{\{1,2\},\{1,4\},\{1,5\},\{2,4\},\{2,5\},\{4,5\},\{4,6\}\}$。同理,生成 $L_{db}^{3}=\{\{1,2,4\},\{1,2,5\},\{1,4,5\},\{2,4,5\}\}$;$L_{db}^{4}=\{\{1,2,4,5\}\}$,$|L_{db}^{4}|=1<4$,该子算法结束。相关矩阵及数组如图 1 所示。

2)求 $L_{DB\text{-}new}^{k}$　对于 DB,当 s 为 20% 时,已知 $L_{DB}^{2}=\{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{2,4\},\{2,5\}\}$、$L_{DB}^{3}=\{\{1,2,3\},\{1,2,4\},\{1,2,5\}\}$。而当 s'变为 10% 时,

a)产生 $L_{DB\text{-}new}^{2}$(一般不求频繁 1-项集,因此省略)。由性质 7 得 $L_{DB\text{-}new}^{2}=\{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{2,4\},\{2,$

5}}。调用矩阵压缩算法将 A_{DB} 转换为 A^2_{DB},调用频繁 k-项集更新算法得到新增的频繁 2-项集{1,6}、{3,6}、{3,4}、{3,5}、{3,6}和{4,5},将它们添加到 $L^2_{DB\text{-}new}$ 中,即 $L^2_{DB\text{-}new}$ = {{1,2},{1,3},{1,4},{1,5},{1,6},{2,3},{2,4},{2,5},{3,4},{3,5},{3,6},{4,5}}。

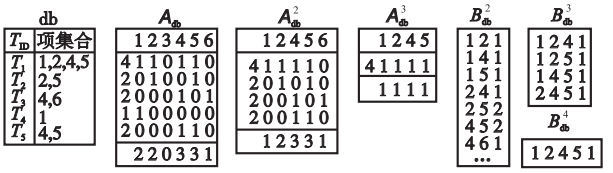

图1 db及其相关矩阵和数组

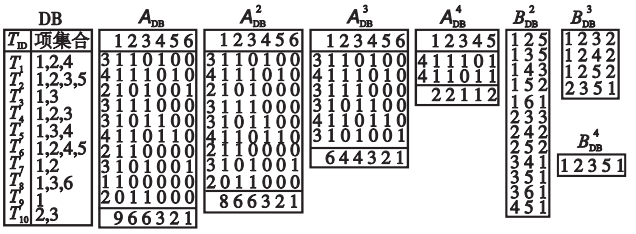

图2 DB的相关矩阵和数组

b)产生 $L^3_{DB\text{-}new}$。由性质7得 $L^3_{DB\text{-}new}$ = {{1,2,3},{1,2,4},{1,2,5}}。调用矩阵压缩算法将 A_{DB} 转换为 A^3_{DB},调用频繁 k-项集更新算法得到新增的频繁 3-项集{1,4,5}、{2,3,5},将它们添加到 $L^3_{DB\text{-}new}$ 中,即 $L^3_{DB\text{-}new}$ = {{1,2,3},{1,2,4},{1,2,5},{1,4,5},{2,3,5}}。

由于 $|L^3_{DB\text{-}new}|=5>3$,还应该有频繁 4-项集[10],调用矩阵压缩算法将 A^3_{DB} 转换为 A^4_{DB}。调用频繁 k-项集生成算法得到频繁 4-项集 $L^4_{DB\text{-}new}$ = {{1,2,3,4}}。由于 $|L^4_{DB\text{-}new}|=1<4$,DB 中最大频繁项集的次数为4,该子算法结束。

3)将 $L^k_{DB\text{-}new}$ 与 L^k_{db} 中的项集进行比较,产生 L^k_{DB+db}

a)求 L^2_{DB+db}。由于{1,2}、{1,4}、{1,5}、{2,4}、{2,5}、{4,5}均在 $L^2_{DB\text{-}new}$、L^2_{db} 中,有 L^2_{DB+db} = {{1,2},{1,4},{1,5},{2,4},{2,5},{4,5}}。{1,3}仅属于 $L^2_{DB\text{-}new}$,{1,3}是否为全局频繁项集取决于 $|\{1,3\}|_{DB+db}=|\{1,3\}|_{DB}+|\{1,3\}|_{db}\geqslant s'\times|DB+db|$ 是否成立。查 B^2_{DB}、B^2_{db} 可知,$|\{1,3\}|_{DB+db}=5\geqslant 1.5$,所以{1,3}为全局频繁项集;同理,{2,3}也为全局频繁项集。而 $|\{1,6\}|_{DB+db}=1<1.5$,因此,它不是全局频繁项集;同理,{3,4}、{3,5}、{3,6}、{4,6}均不是全局频繁项集。因此 L^2_{DB+db} = {{1,2},{1,4},{1,3},{1,5},{2,3},{2,4},{2,5},{4,5}}。

b)求 L^3_{DB+db}。由于{1,2,4},{1,2,5},{1,4,5}均在 $L^3_{DB\text{-}new}$、L^3_{db} 中,有 L^3_{DB+db} = {{1,2,4},{1,2,5},{1,4,5}}。{1,2,3}仅属于 $L^3_{DB\text{-}new}$,所以{1,2,3}是否为全局频繁项集取决于 $|\{1,2,3\}|_{DB+db}=|\{1,2,3\}|_{DB}+|\{1,2,3\}|_{db}\geqslant s'\times|DB+db|$ 是否成立。查 B^3_{DB} 和 B^3_{db} 可知,$|\{1,2,3\}|_{DB+db}=2\geqslant 1.5$,所以{1,2,3}为全局频繁项集。而 $|\{2,3,5\}|_{DB+db}=1<1.5$,因此,{2,3,5}不是全局频繁项集;同理,{2,4,5}也不是全局频繁项集。因此 L^3_{DB+db} = {{1,2,3},{1,2,4},{1,2,5},{1,4,5}}。

c)求 L^4_{DB+db}。由于{1,2,3,5}仅属于 $L^4_{DB\text{-}new}$ 且 $|\{1,2,3,5\}|_{DB+db}=1<1.5$,{1,2,3,5}不为全局频繁项集;同理,{1,2,4,5}也不为全局频繁项集,因此 $L^4_{DB+db}=\varnothing$。算法结束。

结合 UABM 的算法思想和本实例可以看出:

a)UABM 算法以时间为基准,将更新后的数据库分为原数据库 DB 和新增数据库 db(数据库不变的情况下,db 为空),无论支持度怎样变化,都可分别挖掘出 DB 和 db 的频繁 k-项集,然后再从中挖掘 DB + db 的频繁 k-项集。这样就可以灵活处理最小支持度不变而数据库记录数增大、最小支持度发生变化而数据库记录数不变以及最小支持度改变而数据库记录数也增大的多种情况下频繁项集的更新,不像 FUP、IUA 算法只能处理频繁项集更新的单一情况。

b)在求 $L^k_{DB\text{-}new}$ 时,当 $s<s'$ 时,利用性质7,只需在 L^k_{DB} 中删除所有支持度小于 s' 的项集,即可得到 $L^k_{DB\text{-}new}$。而当 $s>s'$ 时,利用性质7,将 $L^k_{DB\text{-}new}$ 的初值定为 L^k_{DB},然后对矩阵进行裁剪。对裁剪后的矩阵 $A_{r\times p}(k<p)$ 的第 $1\sim p-1$ 列所对应的 ID 号进行 k 维组合(有 $s=C^k_{p-1}$ 个组合对),并用数组 $B_{s\times(k+1)}$ 来存放各个组合对(下标从0开始,第 k 列代表各组合对的支持数);从 $B_{s\times(k+1)}$ 中删除原频繁 k-项集对应的组合对,对余下的每个组合对求 k 维支持数,k 维支持数小于最小支持数的组合对所对应的项集则为新增的 k-频繁项集。这就避免了对 L^k_{DB} 中原频繁 k-项集的重复挖掘。

c)UABM 采用位运算的思想产生频繁 k-项集,只需扫描数据库一次且不产生庞大的候选项集,避免了 Apriori 算法存在的处理大量候选项集和重复扫描数据库的问题;在挖掘过程中对矩阵进行裁剪,极大地减少了求高次频繁项集的运算时间;跨越了由低次到高次逐步查找频繁子集的限制,在不知低次 k-1 频繁项的前提下同样可以直接计算频繁 k-项集,而且 k 越大,搜索的矩阵范围越小。这与基于 Apriori 的 FUP、IUA 算法相比,避免了大量不必要的比较和计算。

d)UABM 算法在挖掘过程中采用数组结构和位运算操作,相比 IM 算法要保存庞大的 FP-tree、头链表、条件基等信息,节省了大量的存储空间。

4.2 实验仿真

为了进一步证明 UABM 算法的正确性和有效性,笔者在 WindowsXP(Intel® Pentium® M Processor 1.73 GHz、内存为 1 GB)平台上、SQL Server 2000 和 Delphi 7.0 的环境下实现了 Apriori 算法、FUP 算法、IUA 算法和 UABM 算法,交易数据库由数据项集{$A,B,\cdots,Z$}中以随机方式产生任意长度(1~6)的数据项的 100 000 条交易记录组成。实验结果表明,在最小支持度改变而数据库大小不变、最小支持度不变而数据库大小改变、最小支持度和数据库大小都改变这三种情况下,分别运行 UABM 算法与 Apriori 算法,两者所得的频繁项集完全相同,这说明 UABM 算法是正确的。

图3是在测试数据库记录数不变、最小支持度在 20% ~ 80% 变化时,IUA 和 UABM 算法的执行时间情况对比。由图3可知,当支持度在 0.2~0.6 变化时,UABM 算法的执行速度比 IUA 算法快。这是因为 UABM 算法通过矩阵裁剪、位运算产生频繁项集,仅需扫描数据库一次且不产生庞大的候选项集,其效率比基于 Apriori 框架的 IUA 高。当支持度为 0.2 时,数据库中的频繁项集数目最大,UABM 的这一优势更加突出,因而其执行时间与 IUA 的执行时间相差也最大。因此,UABM 更适合稠密型数据库的频繁项集更新。

图4是在最小支持度不变、测试数据库记录数发生改变时,FUP 和 UABM 算法的执行时间情况对比。从图4可以看出,当测试数据库变大时,UABM 执行时间的增长速度远小于 FUP。这主要是因为,虽然 UABM 和 FUP 都是将新增事务单独组成新增数据库进行处理,但是后者基于 (下转第 863 页)

在线路径规划的实时性要求。

表 2 为分别采用改进 ACS 算法、标准 ACS 算法和实时编码遗传算法规划一条全局优化路径的收敛速度比较。图 5 给出了三者在动态环境中实时规划路径的仿真结果比较。

表 2　收敛速度比较表

	改进 ACS	一般 ACS	real-code GA
Average CPU time each process/s	1.983 280	2.199 543	21.152 507
Average length of Robot path each process(mile)	12.218 947	12.501 450	12.555 060

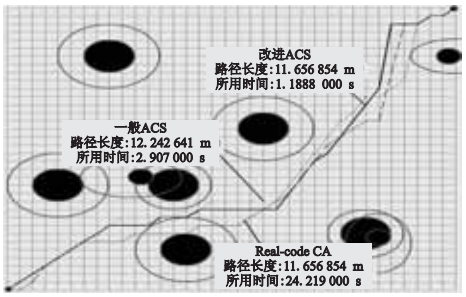

图5　运用三种算法的仿真结果图

本文的方法是使移动机器人在运动过程中,逐步地在线规划出实时局部最优路径来进行导航,因此适用于任何未知的复杂动态环境。

5　结束语

本文以改进的 ACS 算法为基础,提出了一种适用于未知动态环境的移动机器人实时导航方法。该方法利用模糊描述对环境进行建模,将改进 ACS 算法与机器人滚动在线路径规划方法相结合,较好地解决了机器人对环境不确定信息的适应性,以及在线规划的实时性等问题。对未知环境的模糊建模,不仅减弱环境检测误差对机器人导航的影响,而且很好地模拟了人对环境的感知和认知;而改进的 ACS 算法由个体的简单智能,并行地演绎出群体优越的智能行为,又很好地满足了机器人在线路径规划的实时性要求。因此本方法是解决复杂动态环境下机器人在线规划和导航的有效方法。

参考文献:

[1] 庄慧忠,杜树新,吴铁军. 机器人路径规划及相关算法研究[J]. 科学通报,2004,20(3):210-215.
[2] 庄晓东,孟庆春,高云,等. 复杂环境中基于人工势场优化算法的最优路径规划[J]. 机器人,2003,25(6):531-534.
[3] 陈刚,沈林成. 复杂环境下路径规划问题的遗传路径规划方法[J]. 机器人,2001,23(1):40-44.
[4] RAM A, ARLCIN R, BOONE G, *et al*. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation [J]. Adaptive Behavior,1994,2(3):277-304.
[5] 李保国,宗光华. 未知环境中移动机器人实时导航与避障的分层模糊控制[J]. 机器人,2005,27(6):481-485.
[6] 刘成良,张凯,付庄,等. 神经网络在机器人路径规划中的应用研究[J]. 机器人,2001,23(7):605-608.
[7] DORIGO M, GAMBARDELLA L M. Ant colony system: a cooperative learning approach to the traveling salesman problem[J]. IEEE Trans on Evolutionary Computation,1997,1(1):53-66.
[8] 朱庆保. 全局未知环境下多机器人运动蚂蚁导航算法[J]. 软件学报,2006,17(9):1890-1897.
[9] 席裕庚,张纯刚. 一类动态不确定环境下机器人的滚动路径规划[J]. 自动化学报,2002,28(2):161-175.
[10] DORIGO M. 蚁群优化[M]. 罗旭耀,译. 北京:清华大学出版社,2007.

(上接第 840 页) Apriori 框架,需要多次扫描数据库、产生庞大的候选项集并需要更大的存储空间;而前者将事务数据库表示成矩阵,通过矩阵裁剪、位运算产生频繁项集,既不需要产生庞大的候选项集,也不需要庞大的存储空间且仅需扫描数据库一次。因而前者的运行效率较后者高,其执行时间的增长速度远小于后者。因此,UABM 算法比 FUP 更适合大型数据库的频繁项集更新。

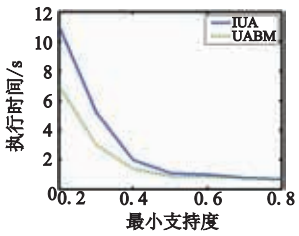

图3　IUA和UABM执行效率对比

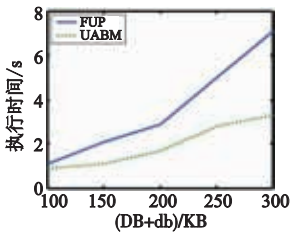

图4　FUP和UABM执行效率对比

5　结束语

UABM 算法首先以时间为基准划分新、旧数据库,然后通过矩阵裁剪、向量运算更新维护频繁项集。它不仅能处理多种条件组合下的频繁项集更新维护,而且只需扫描一遍数据库且不产生候选项集,与其他同类算法[6~9]相比,UABM 具有较高的效率。

由于 UABM 算法在更新或生成频繁 k-项集时,需要对数组 B 中的组合对求 k 维支持数,随着数据库事务数量的增大,求 k 维支持数的时间开销也将增大。在求 k 维支持数之前,能否利用 Apriori 算法的性质剪掉 B 中不可能成为频繁项集的组合对,以减少求 k 维支持数的时间开销,进一步提高 UABM 算法的效率,这将是下一步要研究的问题。

参考文献:

[1] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining association rules between sets of items in large databases[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1993:207-216.
[2] AGRAWAL R, SRIKANT R. Fast algorithm for mining association rules[C]//Proc of the 20th International Conference on VLDB. Santiago Chile:[s,n],1994:487-499.
[3] HAN J, KAMBER M. Data mining: concepts and techniques[M]. Beijing: Higher Education Press, 2001:123-140.
[4] HAN Jia-wei, PEI Jian, YIN Yi-wen. Mining frequent patterns without candidate generation: a frequent-pattern tree approach[J]. Data Mining and Knowledge Discovery, 2004,8(1):53-87.
[5] WANG Jian-yong, Han J, LU Y, *et al*. An efficient algorithm for mining top-k frequent closed itemsets[J]. IEEE Trans on Knowledge and Data Engineering, 2005,17(5):652-663.
[6] CHEUNG D W, HAN Jia-wei, NG V, *et al*. Maintenance of discovered association rules in large database: an incremental updating technique [C]//Proc of the 12th International Conference on Data Engineering. New Orleans: IEEE Computer Society, 1996:106-114.
[7] 朱红蕾,李明. 一种高效维护关联规则的增量算法[J]. 计算机应用研究,2004,21(9):107-109.
[8] 冯玉才,冯剑琳. 关联规则的增量式更新算法[J]. 软件学报,1998,9(4):301-306.
[9] MA Xiu-li, TONG Yun-hai, TANG Shi-wei, *et al*. Efficient incremental maintenance of frequent patterns with FP-tree[J]. Journal of Computer Science and Technology, 2004,19(6):876-884.
[10] 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.
[11] 李广原,雷鸿,龙琥. 一种新的动态频繁项集挖掘方法[J]. 计算机工程与应用,2008,44(21):209-211.