

## 基于同类概念的概念格横向合并算法

张磊<sup>1,2</sup>, 沈夏炯<sup>1,2</sup>, 贾培艳<sup>1</sup>, 许研<sup>1</sup>

(1. 河南大学 计算机与信息工程学院, 河南 开封 475001;

2. 河南大学 数据与知识工程研究所, 河南 开封 475001;)

(zhanglei@henu.edu.cn)

**摘要:**提出了同类概念的观点,在格的合并算法中通过利用同域概念格之间的同类概念和概念的父—子关系实现对其所有子节点的快速更新,以提高概念格横向合并算法的时间效率。并进一步论证了把节点数量较少的格插入到节点多的格中比反着做更省时间。实验表明,该算法和相关文献中的同域概念格横向合并算法相比,其时间性能有明显改善。

**关键词:**同域概念格;并行构造;横向合并;同类概念

**中图分类号:** TP311.12 **文献标识码:** A

## Horizontal union algorithm of concept lattice based on Congener concept

ZHANG Lei<sup>1,2</sup>, SHEN Xia-jiong<sup>1,2</sup>, JIA Pei-yan<sup>1</sup>, XU Yan<sup>1</sup>

(1. School of Computer and Information Engineering, Henan University, Kaifeng Henan 475004, China;

2. Institute of Data and Knowledge Engineering, Henan University, Kaifeng Henan 475004, China)

**Abstract:** Congener concept was introduced into the lattice union algorithm, which was used to update each child node in less time according to the order relation of the concepts. It was further testified that to insert a small lattice into a large one would spend less time. The test results show that this algorithm makes better improvement in time complexity than the horizontal union algorithms of common universe concept lattice found in related literature.

**Key words:** common universe concept lattice; parallel construction; horizontal union; congener concept

## 0 引言

概念格是形式概念分析(Formal Concept Analysis, FCA)<sup>[1]</sup>的核心数据结构,它表现了数据的知识视图,目前正在知识发现<sup>[2]</sup>、机器学习<sup>[3]</sup>、软件工程<sup>[4]</sup>等诸多领域得到了一定的应用。

如何降低从一个大规模、超大规模的形式背景中构造概念格的时间复杂度是目前形式概念分析领域研究的一个重点和难点。将原来的串行算法改造成并行算法,从而利用并行计算能力改善时间性能,成为一个新的途径<sup>[6,7]</sup>。其中分解形式背景然后并行构造概念格<sup>[5]</sup>被认为是一个有效的解决方法:将形式背景进行分解,针对每一个子形式背景分别构造概念格,然后对这些概念格进行合并。文献[5]中的横向合并算法使用按外延对象数的升序插入格节点的方式进行概念格的合并,避免了对更新和新生节点的重复比较,提高了算法效率。本文对格节点进一步细分,提出一种利用同类概念更新节点的快速合并算法,进一步提高概念格横向合并的效率。

## 1 概念格横向合并的基本概念和算法

给定形式背景  $K = (U, D, I)$ , 如果形式背景  $K_1 = (U_1, D_1, I_1)$  和  $K_2 = (U_2, D_2, I_2)$  满足  $U_1 \subseteq U, U_2 \subseteq U, D_1 \subseteq D, D_2 \subseteq D$ , 则称  $K_1$  和  $K_2$  是同域形式背景,它们都为  $K$  的子形式背景,同时称形式背景  $K_1$  的概念格  $L(K_1)$  和形式背景  $K_2$  的概念格  $L(K_2)$  是同域概念格。对象域不同、属性域相同的同域

概念格的合并操作称为概念格的纵向合并。对象域相同、属性域不同的同域概念格的合并称为概念格的横向合并。

表1 形式背景示例

对象	属性			
	a	b	c	d
1	1	0	1	1
2	0	1	0	0
3	1	1	1	0
4	0	0	0	1

概念格的横向合并算法(本文中简记该算法为 HUMCL 算法)<sup>[5]</sup>思想是将原概念格  $L(K_1)$  和  $L(K_2)$  中的概念都按外延的势的大小从小到大排列,然后采用改进的基于属性的概念格渐进式生成算法(本文中简记该算法为改进的 CLIF-A 算法)把  $L(K_2)$  中的概念一一插入到  $L(K_1)$  中。

在此基础上,本文提出了同域概念格的同类概念,并分析用它进行概念格的横向合并算法的理论依据及意义,然后提出基于同类概念的概念格横向合并算法。

## 2 基于同类概念的概念格横向合并算法

同类概念在同域概念格的横向合并中具有良好的性质,可以很容易地通过更新其子节点的方式进行合并。

### 2.1 同类概念的快速合并

**定义1** 如果  $C_1$  和  $C_2$  分别是两个同域概念格  $L(K_1)$  和

收稿日期:2006-02-13;修订日期:2006-03-24 基金项目:河南省自然科学基金资助项目(0311011700)

**作者简介:**张磊(1981-),男,河南博爱人,硕士研究生,主要研究方向:FCA 应用和网络安全技术; 沈夏炯(1963-),男,河南开封人,副教授,主要研究方向:软件工程、知识发现、分布式/并行计算及分布式存储; 贾培艳(1979-),女,河南淮滨人,硕士研究生,主要研究方向:软件工程和数据挖掘; 许研(1982-),女,河南开封人,硕士研究生,主要研究方向:网络技术和知识发现。

$L(K_2)$  中的概念,且  $Extent(C_1) = Extent(C_2)$ , 则概念  $C_1$  和  $C_2$  互为同类概念。

**定理 1** 将概念格  $L(K_2)$  中的格节点  $C_2 = (O_2, D_2)$  插入到  $L(K_1)$  时,如果  $\exists C_1 = (O_1, D_1) \in L(K_1)$ , 且  $O_2 = O_1$ , 即  $C_1$  和  $C_2$  是一对同类概念,那么  $C_2$  的插入仅仅引起  $C_1$  及其所有子节点的更新,在  $L(K_1)$  中不会产生新生节点。

**证明:** 设  $C_p(O_p, D_p) \in L(K_1)$  且  $O_p \subseteq O_1, D_p \supseteq D_1$ , 即  $C_p$  是  $C_1$  或者  $C_1$  的子节点。假设  $C_2$  和  $C_p$  产生新生节点  $C_n(O_n, D_n)$ , 则  $O_n = O_p \cap O_2 = O_p \cap O_1 = O_p, D_n = D_p \cup D_2, C_n$  和  $C_p$  外延相等,所以  $C_n$  是  $C_2$  更新  $C_p$  产生的更新节点而不是新生节点。

设  $C_q(O_q, D_q) \in L(K_1)$  且  $O_q \cap O_1 \neq O_q$ , 即  $C_q$  不是  $C_1$  或者  $C_1$  的子节点。假设  $C_2$  和  $C_q$  产生新生节点  $C_m(O_m, D_m)$ , 则  $O_m = O_q \cap O_2 = O_q \cap O_1, D_m = D_q \cup D_2$ , 因为  $C_q(O_q, D_q)$  和  $C_1(O_1, D_1)$  是  $L(K_1)$  中的节点,  $O_m$  必然等于  $L(K_1)$  中的某个原有节点(该节点是  $C_q$  和  $C_1$  的子节点), 所以不会产生  $C_m$  这样的新生节点。定理得证。

由定理 1 可知, 对同域概念格  $L(K_1)$  和  $L(K_2)$  进行横向的合并运算时, 对于  $L(K_2)$  中的任何一个格节点  $C_2$  如果在  $L(K_1)$  中存在  $C_2$  的同类概念  $C_1$ , 则只需将  $C_1$  以及  $C_1$  的所有子节点用  $C_2$  的外延进行更新即可, 不需要采用改进的 CLIF-A 算法将  $C_2$  插入到  $L(K_1)$  中去。

**定理 2** 当采用改进的 CLIF-A 算法将  $L(K_2)$  中的节点  $C_o$  插入到  $L(K_1)$  中后, 如果  $L(K_1)$  中节点  $C_n$  被标记, 则其所有子节点必然也已被标记。

**证明:** 由改进的 CLIF-A 算法可知, 被标记的节点只能是新生节点或更新节点。设  $C_o = (O_o, D_o) \in L(K_2), C_n = (O_n, D_n)$  是  $C_o$  插入后产生的新生节点或者更新节点, 则  $O_o \supseteq O_n, D_o \subseteq D_n$ 。假设  $\exists C_p = (O_p, D_p) \in L(K_1), C_p \leq C_n$ , 即  $C_p$  是  $C_n$  的子节点, 且  $C_p$  没有被标记, 则  $C_o$  和  $C_p$  比较时, 因为  $O_p \subseteq O_n \subseteq O_o$ , 所以  $O_o \cap O_p = O_p$ 。因此,  $C_o$  会更新  $C_p$  节点并加上标记。所以, 在  $C_o$  插入  $L(K_1)$  中后,  $C_n$  和  $C_n$  的所有子节点都已被更新。定理得证。

由定理 2 可知, 如果在  $L(K_1)$  中存在  $C_2$  的同类概念  $C_1$ , 在用  $C_2$  的外延进行更新  $C_1$  子节点时, 如果  $C_1$  的某个子节点  $C_p$  已经被标记, 则不用再更新  $C_p$  的子节点。

由定理 1 和定理 2, 设计更新同类概念子节点算法如下:

**算法 1** 更新  $L(K_1)$  中  $C_2$  的同类概念  $C_1$  以及  $C_1$  的所有子节点的算法

PROCEDURE UpdateNode( $L(K_1); C_1; C_2$ )

输入: 格  $L(K_1)$ , 格  $L(K_1)$  中的概念  $C_1, C_1$  的同类概念  $C_2$

输出: 更新过的格  $L(K_1)$

BEGIN

IF  $C_1$  未被标记 THEN

用  $C_2$  的外延更新  $C_1$  并标记  $C_1$

FOR  $C_1$  的每个子节点  $C_p$  DO

UpdateNode( $L(K_1); C_p; C_2$ )

ENDFOR

ENDIF

END

同域概念格  $L(K_1)$  和  $L(K_2)$  采用 HUMCL 算法进行横向合并运算时, 需将  $L(K_2)$  中的每个格节点  $C_2$  依次插入到  $L(K_1)$  中。如果在  $L(K_1)$  中存在  $C_2$  的同类概念  $C_1$ , 在改进的 CLIF-A 算法中,  $C_2$  需要和外延对象数比  $C_1$  少的所有节点  $C$  进行比较, 如果  $C$  没有被标记且  $Extent(C_2) \supseteq Extent(C)$ , 则更

新  $C_o$ 。在算法 1 中, 无须这样比较, 仅仅用  $C_2$  更新  $C_1$  的所有子节点即可, 能够有效地节省改进的 CLIF-A 算法中非  $Extent(C_2) \supseteq Extent(C)$  情况的比较时间。

一个新的问题是: 如何快速找到  $L(K_1)$  中  $C_2$  的同类概念  $C_1$ 。为此可采用如下方法进行排序查找:

设形式背景  $K = (G, M, I)$ , 其中  $G = \{g_1, g_2, g_3, g_4, \dots, g_n\}$ , 给定  $M$  中所有元素的顺序为  $g_1 < g_2 < g_3 < g_4 < \dots < g_n$ , 对于  $K$  对应的概念格中的概念  $C(O, D)$ , 将  $O$  中的元素按照前述顺序排序后得到的  $O$ , 称为  $O$  的标准型。显然,  $O$  的标准型是唯一, 并且给定一组  $O$  的标准型, 可以按字典序排列出唯一的序列。对于  $O_1$  和  $O_2$ , 如果  $O_1$  在  $O_2$  的前面, 则记  $O_1 < O_2$ 。

设  $S_1$  和  $S_2$  是两组外延为标准型的概念的有序集合, 集合中元素按照概念外延的字典序升序排列。则查找  $S_1$  和  $S_2$  中对应的同类概念算法如下:

**算法 2** 查找同类概念的算法

FUNCTION Find( $S_1; S_2$ )

输入:  $S_1, S_2$  外延为标准型的概念的有序集合

输出: 二维数组  $T$ , 即  $S_1$  和  $S_2$  中对应的同类概念的二维组集合

BEGIN

$i \leftarrow j \leftarrow n \leftarrow 0$

WHILE  $i < Size(S_1)$  AND  $j < Size(S_2)$  DO

IF  $Extent(S_1[i]) = Extent(S_2[j])$

THEN

$T[n] \leftarrow (S_1[i], S_2[j])$

$n \leftarrow n + 1; i \leftarrow i + 1; j \leftarrow j + 1$

ELSE IF  $Extent(S_1[i]) < Extent(S_2[j])$

THEN

$i \leftarrow i + 1;$

ELSE

$j \leftarrow j + 1$

ENDIF

ENDWHILE

RETURN  $T$

END

由算法 1 和算法 2 可以得到概念格横向合并算法如下:

**算法 3** PROCEDURE UnionLattice( $L(K_1), L(K_2)$ )

输入:  $L(K_1)$  和  $L(K_2)$ , 它们是两个同域概念格

输出:  $L(K_1) \cup L(K_2)$

BEGIN

分别将  $L(K_1)$  和  $L(K_2)$  中的格节点  $C_1$  及  $C_2$  按照  $\|Extent(C_1)\|$  和  $\|Extent(C_2)\|$  的大小归类到  $B_1$  和  $B_2$  中,  $B_1[i] = \{C_1: \|Extent(C_1)\| = i\}, B_2[i] = \{C_2: \|Extent(C_2)\| = i\}$ , 且  $B_1$  和  $B_2$  中的元素按照外延标准型的字典序排列

FOR  $i \leftarrow 0$  TO  $Size(B_1)$  DO

$T \leftarrow Find(B_1[i]; B_2[i])$

FOR 每个格节点  $C \in B_2[i]$  DO

IF  $C$  IN  $T$  THEN

UpdateNode( $L(K_1), T(C, 1), T(C, 2)$ )

ELSE

采用改进的 CLIF-A 算法将  $C$  插入到  $L(K_1)$  中去, 同时更新  $L(K_1)$

ENDIF

ENDFOR

ENDFOR

更新后的  $L(K_1)$  就是  $L(K_1) \cup L(K_2)$

END

$L(K_2)$  中的节点采用算法 3 插入到  $L(K_1)$  中时, 可被视

为两部分,一部分采用算法 1 直接更新同类概念及其子节点,另一部分采用改进的 CLIF-A 算法插入到  $L(K_1)$  中。显然,前一部分节点数量越多,后一部分节点数量越少,就越省时间。

**定义 2** 两个同域概念格  $L(K_1)$  和  $L(K_2)$  中同类概念的数量与  $L(K_1)$  中节点的数量比值称为  $L(K_1)$  对  $L(K_2)$  的概念同类率。

根据定义 2,  $L(K_2)$  对  $L(K_1)$  的概念同类率越高,算法 3 合并概念格就越省时间。并且,对于同域概念格  $L(K_1)$  和  $L(K_2)$ ,如果  $\|CS(K_1)\| \neq \|CS(K_2)\|$ ,  $L(K_1)$  对  $L(K_2)$  的概念同类率和  $L(K_2)$  对  $L(K_1)$  的概念同类率是不同的。

设  $\|CS(K_1)\| \geq \|CS(K_2)\|$ , 且  $L(K_1)$  和  $L(K_2)$  共有  $N$  个同类概念,则有  $(N/\|CS(K_1)\|) \leq (N/\|CS(K_2)\|)$ , 所以  $L(K_2)$  对  $L(K_1)$  的概念同类率大于  $L(K_1)$  对  $L(K_2)$  的概念同类率。对采用算法 3 将  $L(K_1)$  的概念逐个插入  $L(K_2)$  的过程和将  $L(K_2)$  的概念逐个插入  $L(K_1)$  的过程进行比较可以发现,因为同类概念数量相等,所以直接利用算法 1 进行同类概念及其子节点更新的节点数量相等,但是将  $L(K_2)$  的概念逐个插入  $L(K_1)$  的过程中,剩余的采用改进的 CLIF-A 算法插入的节点数量少,耗费的时间就会少。

根据实验(参见第 3 节实验二)可以看到,在两个同域概念格  $L(K_1)$  和  $L(K_2)$  的合并过程中,如果  $\|CS(K_1)\| \geq \|CS(K_2)\|$ , 则将  $L(K_2)$  中的节点采用算法 3 逐个插入  $L(K_1)$  比将  $L(K_1)$  中的节点采用算法 3 逐个插入  $L(K_2)$  更加节省时间。

综上所述,基于同类概念的概念格横向合并算法(Horizontal Union algorithm of Concept Lattice Based on Congener Concept, HUCLBCC)如下:

**算法 4** 基于同类概念的概念格横向合并算法

输入:  $L(K_1)$  和  $L(K_2)$ , 它们是两个同域概念格

输出:  $L(K_1) \cup L(K_2)$

BEGIN

IF  $\|CS(K_1)\| \geq \|CS(K_2)\|$  THEN

UnionLattice( $L(K_1)$ ,  $L(K_2)$ )

ELSE

UnionLattice( $L(K_2)$ ,  $L(K_1)$ )

ENDIF

END

可以看到,若同类概念率为零,算法退化为 HUMCL 算法,时间性能会比 HUMCL 算法略低,因为还需要花费时间去寻找同类概念。同类概念率越高,算法优越性越明显,若同类概念率为 1,将全部采用更新同类概念及其子节点的方式进行节点插入,算法性能达到最优。

## 2.2 算法示例

设把表 1 所示的形式背景横向分成两个子背景  $K_1 = (G, M_1, I_1)$  和  $K_2 = (G, M_2, I_2)$ , 其中  $M_1 = \{a, b\}$  和  $M_2 = \{c, d\}$ 。其对应的概念格为  $L(K_1)$  和  $L(K_2)$ , 分别如图 1 中的 (a)、(b) 所示。

两个概念格中的节点按其外延对象数的多少从小到大大进行处理。为方便区别,图中的节点均加上了标号。现在把  $L(K_2)$  中的节点依次加入到  $L(K_1)$  中。

① 将  $L(K_2)$  中的 #5 节点插入到  $L(K_1)$  中,没有同类概念,采用改进的 CLIF-A 算法完成插入过程。生成的  $L(K_1)$  如图 2 所示。其中 #9、#10 节点是插入过程中的新生节点,已经

加有“\*”标记,在以后的操作中不再参与比较。

② 将  $L(K_2)$  中的 #6 节点插入到  $L(K_1)$  中,因为此时 #6 和 #2 节点是同类概念。所以调用 UpdateNode 算法,用 #6 节点更新 #2 节点以及 #2 节点的所有子节点(已有“\*”标记的除外):将 #2 节点更新为 #2( $\{1,3\}, \{a,c\}$ ),并加上“\*”标记;因为 #2 节点的子节点 #10 节点已经被标记,所以不再比较 #10 节点及其子节点;将 #2 节点的子节点 #1 节点更新为 #1( $\{3\}, \{a,b,c\}$ ),并加上标记;#1 节点的子节点 #9 节点已经被标记,所以不再比较。#6 节点插入过程完成。

③ 重复上述过程,依次将  $L(K_2)$  中的 #7、#8 节点插入到  $L(K_1)$  中,完成概念格的合并操作。

从本例中可以看到,如果存在同类概念,采用 UpdateNode 算法可以很快完成节点的插入操作,省去了无关节点的比较时间。显然,概念同类率越高,算法性能越优。

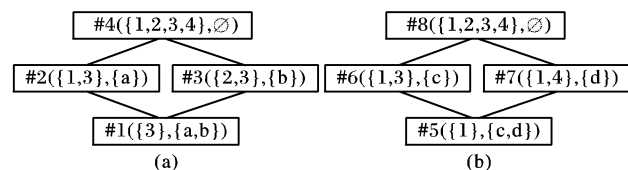


图 1 子背景  $K_1$  和  $K_2$  对应的格  $L(K_1)$  和  $L(K_2)$

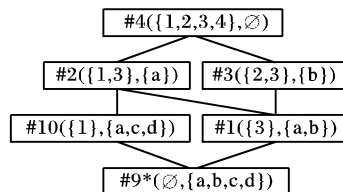


图 2 #5 节点插入后的  $L(K_1)$

## 3 实验及其讨论

为了验证上述基于同类概念的概念格横向合并算法的有效性,在 Windows 2000 下用 Delphi 实现了该算法,并在奔腾 IV 1.7G 的计算机上对随机产生的一系列形式背景  $K$  以及  $K$  的两个同属性域子形式背景  $K_1$  和  $K_2$ , 根据不同目的分别进行了两个实验。

实验一,分别采用 HUMCL 算法和 HUCLBCC 算法对形式背景  $K_1$  和  $K_2$  对应的概念格进行合并操作,结果如图 3 所示。可以看到,在实验分组 1 中,同类概念率接近零,此时 HUCLBCC 算法几乎退化为 HUMCL 算法, HUCLBCC 算法与 HUMCL 算法的时间比值高于 1,因为此时 HUCLBCC 算法要多花出一些时间查找同类概念。在实验分组 21 中,同类概念率接近 0.9,此时两种算法的时间比值约为 0.1,即 HUCLBCC 算法大约只用了 HUMCL 算法所用时间的十分之一即完成合并操作。总体看来,合并子形式背景对应的概念格  $L(K_1)$  和  $L(K_2)$  时, HUCLBCC 算法与 HUMCL 算法所耗费时间的比值随着概念同类率的增加而降低。这说明概念同类率越高, HUCLBCC 算法比 HUMCL 算法越省时间。

实验二,观察同域概念格的节点规模对算法 3 的影响。采用算法 3 进行合并格,分别将  $L(K_2)$  插入到  $L(K_1)$  中和将  $L(K_1)$  插入到  $L(K_2)$  中,实验结果如图 4(a) 和 (b) 所示。(a) 中的  $x$  轴表示实验的分组序号,  $y$  轴表示每组对象中  $L(K_1)$  和  $L(K_2)$  中的节点数量。在 (b) 中  $x$  轴表示实验的分组序号,与 (a) 中的实验分组序号相对应,  $y$  轴表示合并两个格  $L(K_1)$  和  $L(K_2)$  所耗费的时间,每组实验的两个柱形自左向

右分别是  $L(K_2)$  插入  $L(K_1)$  中耗费的时间和  $L(K_1)$  插入  $L(K_2)$  中耗费的时间。

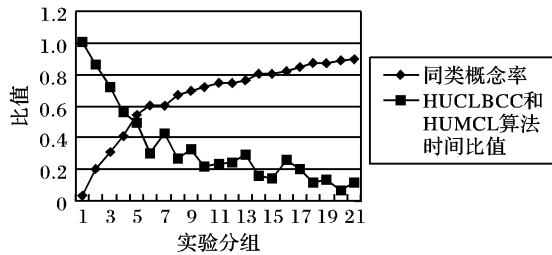


图3 时间性能随概念同类率趋势图

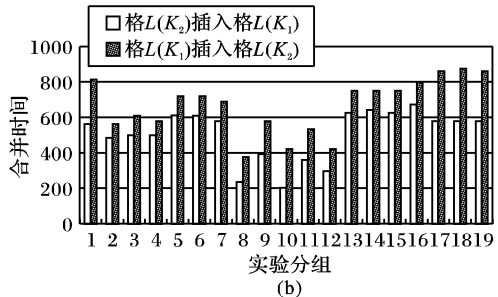
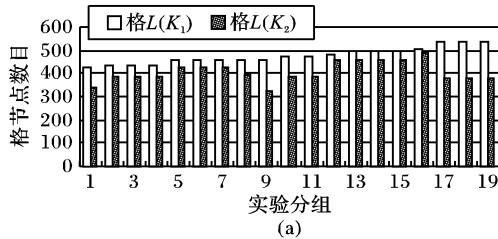


图4  $L(K_1)$  和  $L(K_2)$  插入顺序不同对算法3性能的影响

图4表明:把节点数量较少的格插入到节点多的格中比反着做更省时间。例如,实验中第7个分组,格  $L(K_1)$  的节点个数比较多,将格  $L(K_2)$  插入  $L(K_1)$  进行合并耗费的时间要

少于将格  $L(K_1)$  插入  $L(K_2)$  耗费的时间,说明把节点数量较少的格插入到节点多的格中比反着插入更省时间。

#### 4 结语

本算法利用了待插入节点的同类概念仅需更新自身及其子节点即可完成插入的特点,节省大量的比较时间,实验表明,本文所提出的基于同类概念的概念格横向合并算法是有效的。并且,该算法也很适用于对概念格进行并行构造,将本算法真正地用于概念格分布并行构造是我们将要进行的下一步工作。

#### 参考文献:

- [1] GANTER B, WILLE R. Formal Concept Analysis: Mathematical Foundations[M]. Berlin: Springer-Verlag, 1999.
- [2] KROHN U, DAVIES NJ, WEEKS R. Concept lattices for knowledge management[J]. BT Technology Journal, 1999, 17(4): 108 - 113.
- [3] KUZNETSOV SO. Machine learning on the basis of formal concept analysis[J]. Automation and Remote Control, 2001, 62(10): 1543 - 1564.
- [4] TILLEY T, COLE R, BECKER P, et al. A Survey of Formal Concept Analysis Support for Software Engineering Activities[A]. Proceedings of 1st International Conference on Formal Concept Analysis[C]. 2003.
- [5] 李云, 刘宗田, 陈峻, 等. 多概念格的横向合并算法[J]. 电子学报, 2004, 11(11): 1849 - 1854.
- [6] NJIWOUA P, NGUIFO EM. A parallel algorithm to build Concept Lattice[A]. Proceedings of the 4th Groningen International Information Technology Conference for Students[C]. University of Groningen, The Netherlands: Fevrier 1997. 103 - 107.
- [7] FU HG, NGUIFO EM. A Parallel Algorithm to Generate Formal Concepts for Large Data[A]. Second International Conference on Formal Concept Analysis, ICFCA 2004[C]. Sydney, Australia: Springer, 2004. 394 - 401.

(上接第1899页)

BCTAR 算法在不同的支持度阈值下均表现出较好的性能,而时序 Apriori 算法则随着阈值的不同差异很大。

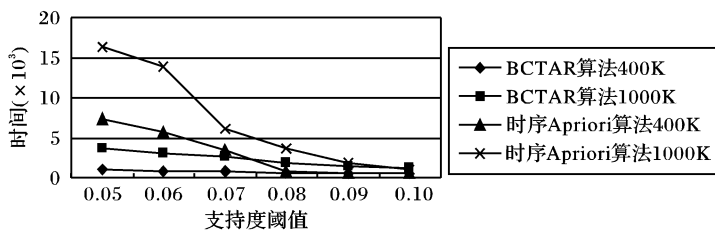


图1 算法在不同的事务数据库和不同的支持度阈值下的执行时间比较

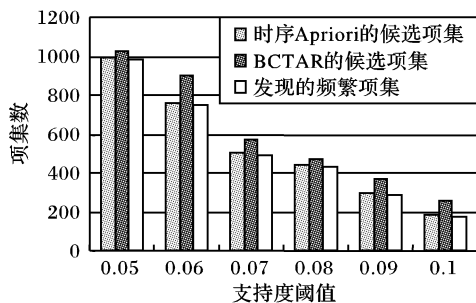


图2 算法所产生的候选项集的比较

实验结果表明,本文提出的基于日历格的时序关联规则挖掘算法能够用于发现时序数据库中任意时间间隔内的时序关联规则,且当支持度阈值较小时,该算法具有较好的性能。

#### 参考文献:

- [1] AGRAWAL R, IMIELINSKI T, SWAMI AN. Mining association rules between sets of items in large databases[A]. Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data[C]. 1993. 207 - 216.
- [2] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules in large databases[A]. Proceedings of the 1994 International Conference on Very Large Data Bases[C]. 1994. 487 - 499.
- [3] ALE JM, ROSSI GH. An approach to discovering temporal association rules[A]. Proceedings of the 2000 ACM Symposium on Applied Computing[C]. 2000. 294 - 300.
- [4] HAN J, DONG G, YIN Y. Efficient Mining of Partial Periodic Patterns in Time Series Databases[A]. Proceedings of the International Conference on Data Engineering[C]. 1999. 106 - 115.
- [5] OZDEN B, RAMASWAMY S, SILBERSCHATZ A. Cyclic Association Rules[A]. Proceedings of the 15th International Conference on Data Engineering[C]. 1998. 412 - 421.
- [6] LI Y, NING P, WANG XS, et al. Discovering Calendar-based Temporal Association Rules[J]. Data and Knowledge Engineering, 2003, 44(2): 193 - 218.
- [7] RAMASWAMY S, MAHAJAN S, SILBERSCHATZ A. On the Discovery of Interesting Patterns in Association Rules[A]. Proceedings of the International Very Large Database Conference[C]. 1998. 368 - 379.