多概念格的横向合并算法

李 云[1,2], 刘宗田[1], 陈 [2], 徐晓华[2], 程 伟[2]
(1. 上海大学计算机学院, 上海 200072; 2. 扬州大学信息工程学院, 江苏扬州 225009)

摘 要: 由于概念格自身的完备性, 构造概念格的时间复杂度一直是影响形式概念分析应用的主要因素. 本文首先从形式背景的纵向、横向合并出发, 定义了内涵独立和内涵一致的形式背景和概念格; 还定义了内涵一致的形式背景、概念的横向加运算和概念格的横向并运算, 并证明了横向合并的子形式背景的概念格和子背景所对应的子概念格的横向并是同构的. 最后结合子概念格中概念间固有的泛化- 特化关系, 提出一种多概念格的横向合并算法来构造概念格. 试验表明, 该算法和直接用形式背景来构造概念格的算法相比, 其时间复杂度有显著改善. 显然, 该算法适用于对概念格进行分布并行构造.

关键词: 概念格; 形式背景; 子格; 子背景; 横向合并
中图分类号: TP18 文献标识码: A 文章编号: 0372-2112 (2004) 11-1849-06

Horizontal Union Algorithm of Multiple Concept Lattices

LI Yun[1,2], LIU Zong-tian[1], CHEN Ling[2], XU Xiao-hua[2], CHENG Wei[2]
(1. School of Computer Science, Shanghai University, Shanghai 200072, China;
2. Institute of Information Engineering, Yangzhou University, Yangzhou, Jiangsu 225009, China)

Abstract: Since the completeness of concept lattice, the time complexity of building concept lattice is a factor restricting the application of formal concept analysis. Based on the horizontal and vertical combination in formal contexts, this paper defines the independent or consistent contexts and lattices in attribute field; and also defines the horizontal addition operation between contexts or concepts and the horizontal union operation between concept lattices. In addition, we prove that the concept lattice of subcontexts horizontally combined is isomorphic to the horizontal union of sublattices of these subcontexts. Using the inherent general-special relation between concepts in sublattice, the horizontal union algorithm of multiple concept lattices to construct the concept lattice is also presented. Experimental results show that the time complexity of this algorithm is much better than that of other construction algorithm of concept lattice from whole formal context. Evidently, our algorithm is very suitable for constructing concept lattice in parallel and distributed system.

Key words: concept lattice; formal context; sublattice; subcontext; horizontal union

1 引言

自从德国的 Wille 教授提出了形式概念分析[1]以来, 作为形式概念分析的核心数据结构-概念格已经引起了人们的广泛关注, 并且已经在知识发现、软件工程、信息检索等诸多领域得到了一定的应用[2~4].

概念格的构造是概念格应用的前提. 由于概念格自身的完备性, 构造概念格的时间复杂度一直是影响形式概念分析应用的主要障碍. 研究采用新的方法和手段来构造概念格, 就成为概念格研究的主要内容之一. 现在已经提出了构造概念格的多种算法, 取得了一系列重要成果[5~7], 但是这些研究基本上是针对单个概念格的.

随着网络技术特别是互联网的飞速发展, 数据分布式存储与并行处理的需求越来越迫切. 概念格的分布处理[8]思想就是通过形式背景的拆分, 形成分布存储的多个子背景, 然后构造相应的子概念格, 再由子概念格的合并得到所需的概念格. 也就是说通过多个概念格的合并来构造所需的概念格.

形式背景的拆分有横向和纵向之分. 因而, 多概念格的合并就有横向合并和纵向合并[9]两种. 本文主要讲述多概念格的横向合并算法.

2 概念格的基本概念

对于给定的数据信息表 K= (G, M, I), 在形式概念分析中称为形式背景(formal context), 如表 1 所示. 其中 G 是对象

收稿日期: 2003-10-16; 修回日期: 2004-04-23
基金项目: 国家自然科学基金项目(No. 60275022)

集合, M 为属性集合, I 是 G 和 M 间的二元关系. 对于一个对象 g I G, 属性 m I M, 那么 gIm 就表示对象 g 具有属性 m.

表 1 形式背景示例

G \ M	a	b	c	d
1	@	@		@
2	@		@	
3		@	@	
4	@	@		@
5	@			

形式背景的对象集 AI P(G), 属性集 B I P(M) 之间按如下关系连接:

f(A) = {m I M| P g I A, gIm };

g(B) = {g I G| P m I B, gIm };

则称从形式背景中得到的每一个满足 A= g(B), B= f(A) 的二元组(A, B) 为一个形式概念. 其中 A 是对象密集 P(G) 的元素, 称为概念(A, B) 的外延, B 是属性密集 P(M) 的元素, 称为概念(A, B) 的内涵.

若概念 C_1= (A_1, B_1) 和 C_2= (A_2, B_2) , 满足 A_1 A A_2, 则称(A_1, B_1) 为子概念(或亚概念) , (A_2, B_2) 为父概念(或超概念), 记为: (A_1, B_1) F (A_2, B_2) . 若不存在 C_3= (A_3, B_3) , 满足 (A_1, B_1) < (A_3, B_3) < (A_2, B_2) , 则称(A_1, B_1) 为直接子概念, (A_2, B_2) 为直接父概念. 这种由形式背景中所有形式概念的超概念2亚概念的偏序关系(也称泛化2特化关系) 所诱导出的格称为概念格, 记为 L(K) .

概念格可以用图形化形式表示为有标号的线图, 图中的节点表示一个概念, 节点间的连线表示节点间存在泛化2特化关系. 这种线图也称为 Hasse 图, 它是概念格的可视化表示. 图 1 所示的是表 1 的形式背景对应的概念格的 Hasse 图.

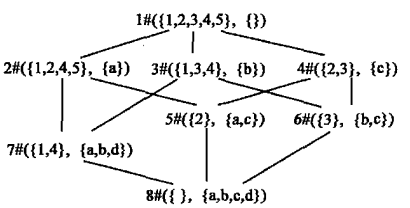

图 1 表 1 的形式背景所对应的概念格

3 多概念格横向合并的依据

由于概念格是其形式背景中的概念间关系的表现形式, 它和对应的形式背景是一一对应的. 因此, 对概念格的分布处理必然涉及到形式背景的拆分、合并等处理. 根据 wille R 在文[1] 中提出了的有关思想, 有如下定义:

定义 1 设 K= (G, M, I) , K_1= (G_1, M_1, I_1) 和 K_2= (G_2, M_2, I_2) 是形式背景. 对 j I {1, 2}, 使用几个缩写: $\hat{G}_j$= {j} @ G_j, $\hat{M}_j$= {j} @ M_j 和 $\hat{I}_j$= {((j, g) , (j, m)) | (g, m) I I_j}, 可以定义:

(1) 如果 G_1= G_2= G, 则 K_1 和 K_2 的并置为:

$$K_1 | K_2 = (G, \hat{M}_1 \cup \hat{M}_2, \hat{I}_1 \cup \hat{I}_2)$$

(2) 如果 M_1= M_2= M, 则 K_1 和 K_2 的叠置为:

$$\frac{K_1}{K_2} = (\hat{G}_1 \cup \hat{G}_2, M, \hat{I}_1 \cup \hat{I}_2)$$

定义中通过用 $\hat{G}_i$ 表示{i} @ G_i、用 $\hat{M}_i$ 表示{i} @ M_i, 是为了确保集合是不相交的.

该定义可以推广到多个形式背景的并置和叠置.

实际上, 并置就可理解为对象域相同、属性项不同的形式背景间的合并, 也可以称为横向合并; 叠置可理解为对象域不同、属性项相同的形式背景间的合并, 也可以称为纵向合并.

定义 2 如果相同对象域的形式背景 K_1= (G, M_1, I_1) 和 K_2= (G, M_2, I_2) 满足 M_1 A M, M_2 A M, 则称 K_1 和 K_2 是同属性项上的形式背景, 简称为同项背景. 形式背景 K_1 和 K_2 所对应的概念格 L(K_1) 和 L(K_2) 是同项概念格.

在同项背景中, 属性 M_1 和 M_2 由两种不同的情形: M_1 H M_2= U 或 M_1 H M_2 X U.

定义 3 在同项形式背景 K_1 和 K_2 中, 若 M_1 H M_2= U, 则称 K_1 和 K_2、L(K_1) 和 L(K_2) 分别是内涵独立的; 若 M_1 H M_2 X U, 对于任意 g I G 和任意 m I M_1 H M_2 满足 gI_1m Z gI_2m, 则称 K_1 和 K_2、L(K_1) 和 L(K_2) 分别是内涵一致的.

显然, 内涵独立的是内涵一致的特例, 内涵独立的一定也是内涵一致的.

定义 4 如果 K_1= (G, M_1, I_1) , K_2= (G, M_2, I_2) 是两个内涵一致的形式背景, 则:

K_1 ° K_2= (G, M_1 G M_2, I_1 G I_2) 称为两个形式背景的横向加运算, 用加横线的加操作符(°) 表示.

定义 5 对于 K= (G, M, I) 中的形式概念 C_i= (A_i, B_i) 、C_j= (A_j, B_j) (i X j) ,

(1) 如果 A_i= A_j, 则称 C_i= C_j, 即概念相等.

(2) 如果 A_i = A_j, 则称 C_i 大于 C_j.

(3) 如果 A_k= A_i H A_j, B_k= B_i G B_j, 则 C_k= (A_k, B_k) = C_i ° C_j, 即概念间横向加运算.

有了上述的定义, 就可以进行概念格的横向合并运算.

定义 6 如果 L(K_1) 和 L(K_2) 是两个内涵一致的概念格, 设它们的横向并运算 L(K_1) G L(K_2) 等于概念格 L, 那么对于 L(K_1) 中的某个概念 C_1 和 L(K_2) 中的某个概念 C_2, 令 C_3= C_1 ° C_2, 如果在 L(K_1) 中的所有小于 C_1 的概念中不存在等于或大于 C_3 的概念, 并且在 L(K_2) 中的小于 C_2 的所有概念中不存在等于或大于 C_3 的概念, 则 C_3 I L.

同样地, 横向并运算用加横线的并操作符(G) 表示.

定理 1 如果 L(K_1) 和 L(K_2) 是内涵一致的概念格, 则 L(K_1) G L(K_2) = L(K_1 ° K_2) .

证明:

(1) 证明在 L(K_1) G L(K_2) 中的概念一定在 L(K_1 ° K_2) 中:

假定 C_3= (A_3, B_3) I L(K_1) G L(K_2) , 如果 C_3 由定义 6 生成的, 则有 C_1= (A_3 G A_x, B_1) I L(K_1) 和 C_2= (A_3 G A_y, B_2) I L(K_2) , 且 A_x H A_y= U 和 B_1 G B_2= B_3. 那么在 K_1 中有 B_1 使得 g(B_1) = A_3 G A_x, f(A_3) B f(A_3 G A_x) = B_1, 这时若 A_x= U, 则 f(A_3) = f(A_3 G A_x) = B_1; 若 A_x X U, 由于任何小于 C_1 的概念中不存在等于或大于 C_3 的概念, 所以只能有 f(A_3) = f(A_3 G A_x) = B_1, 这时(A_3, B_1) 不是一个概念, 因为 A_3 X g(f(A_3)) . 同理在 K_2 中有 B_2 使得 g(B_2) = A_3 G A_y 和 f(A_3) = B_2, 因此在 K_1 ° K_2 中有 B_1 G B_2= B_3 满足 g(B_3) = A_3 和 f(A_3) = B_3, 即 C_3= (A_3, B_3) I L(K_1 ° K_2) ;

(2) 证明在 L(K_1 ° K_2) 中的概念一定在 L(K_1) G L(K_2)

中:

假定 $C_3=(A_3, B_3)\in L(K_1\cup K_2)$, 如果 $B_3=B_1\cap B_2$, 且 B_1 在 K_1 中和 B_2 在 K_2 中, 则在 K_1 中有 $g(B_1)=A_3\cap A_x$, $f(A_3\cap A_x)=B_1$ 和在 K_2 中有 $g(B_2)=A_3\cap A_y$, $f(A_3\cap A_y)=B_2$ 且 $A_x\cup A_y=U$, 即有 $C_1=(A_3\cap A_x, B_1)\in L(K_1)$ 和 $C_2=(A_3\cap A_y, B_2)\in L(K_2)$. 并且由于在 $K_1\cup K_2$ 中有 $f(A_3)=B_3=B_1\cap B_2$, 则在 K_1 中有 $f(A_3)=B_1=f(A_3\cap A_x)$, 而 (A_3, B_1) 不是 $L(K_1)$ 中的一个概念, 因为 $A_3\subset g(f(A_3))$, 也就是说在 $L(K_1)$ 小于 C_1 的概念中不存在等于或大于 C_3 的概念; 同理在 K_2 中有 $f(A_3)=B_2=f(A_3\cap A_y)$, 并且在 $L(K_2)$ 小于 C_2 的概念中不存在等于或大于 C_3 的概念. 则 C_3 能由 $L(K_1)$ 中的 C_1 和 $L(K_2)$ 中的 C_2 根据定义 6 生成, 因此 $C_3=(A_3, B_3)\in L(K_1)\cup L(K_2)$.

至此, 就为多概念格的横向合并提供了依据. 也就是说, 若要构造一个形式背景的概念格, 可先进行形式背景的横向拆分. 为简单起见, 可以采用均等拆分, 把原背景拆分为多个小的子形式背景, 然后采用相应的概念格生成算法构造相应的子概念格, 再通过子概念格的横向并运算就可得到形式背景的概念格.

4　概念格横向合并的算法描述及示例

4.1　概念格横向合并的算法描述

概念格横向合并算法的思想是先利用基于属性的概念格渐进式生成算法[10](简记为 CLIF. A 算法)构建子概念格, 然后依次把一个子格中的概念渐进插入另一个子格中, 形式所需的概念格.

在概念格 $L(K)$ 中追加一个概念 (A, B) 时, 首先根据格中的所有节点和新增的概念间的关系, 找到需要修改的概念; 概念间的关系发生变化时, 相应的边也要作相应的修改.

对于概念 C, 设其内涵、外延分别用 Intent(C) 和 Extent(C) 表示.

定义 7　对于一个概念 $C=(A, B)$, 如果在概念格 $L(K)$ 中存在一个概念 $C_1=(A_1, B_1)$, 并满足 $A_1\subset$ Extent(C), 则称概念 C_1 为对于概念 C 的更新概念.

显然地, 对于一个更新概念来说, 它将被更新为 $(A_1, \text{Intent}(C)\cup B_1)$.

定义 8　对于某个概念 $C=(A, B)$, 如果在概念格 $L(K)$ 中存在一个概念 $C_1=(A_1, B_1)$, 并满足: (1) Newextent = Extent(C) $\cap A_1$, 在格中不存在任意概念 C_2, 使 Extent(C_2) = Newextent; (2) 对于 C_1 概念的任意孩子概念 C_3, 都没有 Extent(C_3) $\cap$ Extent(C) = Newextent; 则称 C_1 为和概念 C 形成新增概念的产生子概念.

对于定义 8 中的条件(2), 实际上就是要保证产生子概念是其新增概念的上确界(supremum)概念.

定理 2　如果概念格 $L(K)$ 某个概念 $C_1=(A_1, B_1)$ 是和概念 $C=(A, B)$ 形成新增概念的产生子概念, 则由其产生的新增概念 $C_{nav}=(\text{Extent}(C)\cap A_1, \text{Intent}(C)\cup B_1)$.

证明: 由定义 8 知道, 在格中原不存在外延等于 Newextent = Extent(C) $\cap A_1$ 的概念, 所以在新概念格中一定需增加一个新节点, 其外延就等于 Newextent, 且 Newextent $\subset$ Extent(C_1). 对于概念格中的一个概念来说, 其内涵是最大化的, 即对内涵的任意扩大都将导致外延的减少. 根据概念中外延和内涵之间的固有关系, 即外延的减少其内涵就相应地增大, 可以得到新增概念的内涵一定是 Intent(C) $\cup B_1$.

这样, 对于同项概念格 $L(K_1)$ 和 $L(K_2)$, 只要把 $L(K_2)$ 中的概念一一插入到 $L(K_1)$ 中就可得到 $L(K_1)\cup L(K_2)$. 采用的算法类似于基于属性的概念格渐进式生成算法, 只是前者是渐增概念而后者是渐增属性. 同时 $L(K_2)$ 中的概念间存在固有的泛化- 特化关系, 因此在把 $L(K_2)$ 中的概念一一插入到 $L(K_1)$ 的过程中要利用概念间的关系, 来降低算法的时间复杂度.

定理 3　假定在原概念格 $L(K_1)$ 和 $L(K_2)$ 中的概念都按外延的势的大小从小到大顺序排列, 如果 $L(K_1)$ 的概念 C_1 是对应 $L(K_2)$ 中的概念 C 的更新概念或新增概念, 并且 $L(K_2)$ 中的概念 C_c 是在概念 C 之后插入 $L(K_1)$, 则无需考虑概念 C_c 和概念 C_1 间的运算.

证明: 因为设概念 C_1 是 $L(K_1)$ 中的概念 C_{c1} 和 $L(K_2)$ 中的概念 C 形成的, 即 $A\cap A_{c1}=A_1$, $B\cup B_{c1}=B_1$. 如果概念 C 和概念 C_c 间存在泛化- 特化关系, 即概念 C 是概念 C_c 的后代, 则有 $A_1\subset A\subset A_c$, $B_1=B=B_c$, 所以 $C_1\cup C_c=C_1$, 则在概念 C_c 插入 $L(K_1)$ 时, 无需考虑概念 C_c 和概念 C_1 间的运算.

如果概念 C 和概念 C_c 间不存在泛化- 特化关系, 即不存在父子关系, 那么最近的关系是兄弟关系. 设 $A\cap A_c=A_d$, $B\cup B_c=B_d$, 那么如果 $A_d=U$, 则概念 C_1 和概念 C_c 不存在任何关系, 即 $A_1\cap A_c=U$, 它们之间的运算也就无需考虑. 如果 $A_d\subset U$, 则 $L(K_2)$ 中概念 $C_d=(A_d, B_d)$ 是概念 C 和 C_c 的孩子. 设概念 C_1 和概念 C_c 形成概念 $C_{d1}=(A\cap A_{c1}\cap A_c, B\cup B_{c1}\cup B_c)=(A_{c1}\cap A\cap A_c, B_{c1}\cup B\cup B_c)$, 由于概念 C_d 先于概念 C_c 插入 $L(K_1)$, 概念 C_{d1} 一定由概念 C_{c1} 和概念 C_d 形成了, 所以也无需考虑概念 C_c 和概念 C_1 间的运算.

对于多于两个的同项概念格, 可以依次地把其他的子格插入到某个子格中.

多概念格的横向合并算法(Horizontal Union Algorithm of Multiple Concept Lattices, 简记为 HUMCL 算法) 的伪码描述如下:

```
INPUT: L(K1)、L(K2), …, L(Kn), n 个同项一致的概念格(n≥2)
OUTPUT: L(K1)∪L(K2)∪…∪L(Kn)
BEGIN
    FOR  L(Ki)中每个概念按外延的势的大小从小到大顺序排列, i= 2, …, n  DO
        采用改进的基于属性的概念格生成算法把概念(A, B)∈L(Ki)插入到L(K1)
    ENDFOR
END
```

改进的基于属性的概念格渐进式生成算法

INPUT: 概念格 $L(K_1)$ 和概念 (A, B)

OUTPUT: 新的概念格 L(K_1)
BEGIN
FOR 每个概念节点(A_1, B_1) I L(K_1), 按照| A_1| 的升序排列 DO
IF 节点(A_1, B_1) 的更新或新增标志 THEN CONTINUE ENDIF (*)
IF A_1 A A THEN; {更新概念}
将 B 加到 B_1 中, B_1= B_1 G B;
将(A_1, B_1) 加入到 VISITED_CS 中;
置(A_1, B_1) 节点的更新或新增标志; (*)
IF A_1= A THEN exit ENDIF
ELSE
Newextent= A_1 H A; {可能是产生子概念}
IF 不存在某个(Ac_1, Bc_1) I VISITED_CS 满足 Ac_1 = Newextent THEN 创建一个新节点 C_{new}= (Newextent, B_1 G B));
增加边(A_1, B_1) y C_{new};
FOR VISITED_CS 中的每个节点 C_a DO
IF (Extent(C_a) Newextent) THEN
Child: = true;
FOR C_a 的每个父节点 C_p DO
IF (Extent(C_p) Newextent) THEN child: = false; break; ENDIF
ENDFOR
IF child THEN
IF C_a 是(A_1, B_1) 的孩子节点 THEN 删除边(A_1, B_1) y C_a; ENDIF
增加边 C_{new} y C_a; {C_a 是新增节点的直接孩子概念}
ENDIF
ENDIF
ENDFOR
将 C_{new} 加入到 VISITED_CS 中;
置 C_{new} 节点的更新或新增标志; (*)
ENDIF
ENDIF
ENDFOR
END

算法中标注(*) 的行是对基于属性的概念格渐进式生成算法 (CLIF_A) 的改进部分.

412 算法分析及简单示例

现以 L(K_1) -G L(K_2) 为例, 来分析 HUMCL 算法的复杂度. HUMCL 算法的核心部分是改进的基于属性的概念格渐进式生成算法. 设在格 L(K_1) 包含的概念数为| L_1| , 格 L(K_2) 包含的概念数为| L_2| . 先考虑不含改进部分的 CLIF_A 算法, 若要在原格 L(K_1) 中要插入概念(A, B) = (g(B) , B) , 至多产生 $2^{|g(B)|}$ 个外延包含于 g(B) 的概念. 因此, 更新节点和新增节点的数目至多为 $2^{|g(B)|}$. 假设对于格 L(K_2) 中的所有概念(g(B_i) , B_i) , 有 average(| g(B_i) |) F K, 则可认为插入一个概念时更新节点和新增节点的数目为 2^K, 所以原 CLIF_A 算法的时间复杂度可表示为 O(2^K@| L_1|), 这时格 L(K_1) 包含的概念数| Lc_1| F 2^K+ | L_1| . 若考虑再插入一个概念, 那么算法的时间复杂度为 O(2^K@| Lc_1| = O(2^K(2^K+ | L_1|)) = O(2^{2K}+ 2^K @| L_1|)) , 这时格 L(K_1) 包含的概念数| Ld_1| 2@2^K+ | L_1| . 现在要把| L_2| 个概念依次插入 L(K_1), 其时间复杂度可表示为 O(2^{2K}@((| L_2| - 1) + , + 3+ 2+ 1) + 2^K@| L_1| @| L_2|) = O (2^{2K}@((| L_2| - 1)| L_2|)/ 2+ 2^K@| L_1| @| L_2|) = O(2^{2K}@| L_2 | ^{2}+ 2^K@| L_1| @| L_2|), 即若采用不含改进部分的 CLIF_A 的 HUMCL 算法的复杂度为 O(2^{2K}@| L_2| ^{2}+ 2^K@| L_1| @| L_2|). 对于改进的 CLIF_A 算法, 由于每次加入概念都无需考虑和以前概念插入时新增或更新概念之间的运算, 那么当插入第一个概念后再插入一个新概念时的时间复杂度仍为 O(2^K@| L_1|), 所以要把| L_2| 个概念依次插入 L(K_1), 其时间复杂度可表示为 O(2^K@| L_1| @| L_2|), 即若采用含改进部分的 CLIF_A 的 HUMCL 算法的复杂度为 O(2^K @| L_1| @| L_2|) . 可以看出, 这时算法的复杂度显著降低.

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}. 其对应的子格为 L (K_1) 和 L(K_2), 分别如图 2 中的(a) 、(b) 所示.

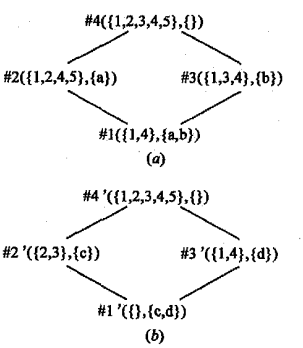

图 2 子背景 K_1 和 K_2 对应的子格 L(K_1) 和 L(K_2)

两个子格中的节点按其外延的势的大小从小到大按序进行处理, 其序号标注在节点旁. 现在把 L(K_2) 中的节点依次加入到 L(K_1) 中.

¹ 加入节点# 1c: 首先和# 1 运算, {} H {1, 4}= {}, {a, b} G {c, d}= {a, b, c, d}, 得到节点# 5({}, {a, b, c, d}), 而在 L(K_1) 和 L(K_2) 中都不存在小于# 1 和# 1c 节点, 所以节点# 5({}, {a, b, c, d}) 是新增节点, 其产生子是# 1; 而# 1c 和# 1 的后续节点都产生外延和# 5 相同的节点, 所以不会产生新增节点. 加入# 1c 后的格如图 3(a) 所示, 其中加粗的节点是新增节点, 加粗的实线表示它和其产生子节点之间的连接线.

º 加入节点# 2c: 由于它是节点# 1c 的后加入节点, 所以无需考虑它和新增节点# 5 间的运算. 和# 1 运算, 由于{1, 4} H {2, 3}= {}, 不会产生新节点; 和# 2 运算, {1, 2, 4, 5} H {2, 3}= {2}, { a} G { c}= { a, c}, 得到节点# 6({2}, { a, c}), 由于在 L(K_1) 和 L(K_2) 中小于# 2 和# 2c 的节点都没有等于或大于# 6, 所以节点# 6({2}, { a,

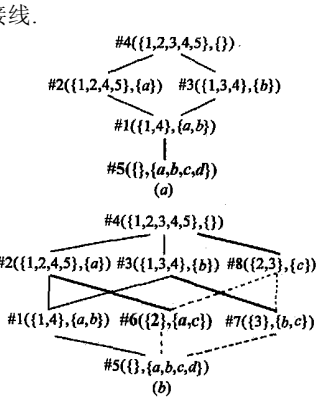

图 3 加入 L(K_2) 中的节点时 L(K_1) 的变化示意

c})为新增节点, 其产生子是# 2; 同理, 和节点# 3 运算产生新增节点# 7({3}, {b, c}), 和节点# 4 运算产生新增节点# 8({2, 3}, {c}), 这里的新增是指新增到 $L(K_1)$ 中的节点. 加入# 2c 后的格如图 3(b)所示, 其中加粗的节点是新增节点, 加粗的实线表示它和其产生子节点之间的连接线, 加粗的虚线表示它和其直接孩子节点之间的连接线.

» 加入节点# 3c: 由于无需考虑和新增节点间的运算, 它首先和# 1 运算, 因为{1, 4} A {1, 4}, 所以节点# 1 需要更新, 形成更新节点({1, 4}, {a, b} G {d}) = ({1, 4}, {a, b, d}); # 3c 和后面# 2、# 3、# 4 节点运算都形成外延{1, 4}, 所以不需要再进行处理. 加入# 3c 后的格实际上已经和图 1 相同.

¼ 加入节点# 4c: 只需考虑它和节点# 2、# 3、# 4 运算, 明显地, 节点# 2、# 3、# 4 需更新, 但由于节点# 4c 的内涵为{}, 所以这些节点不变.

可以看出, $L(K_1) \cup L(K_2) = L(K_1 \circ K_2) = L(K)$.

5　试验及其讨论

为了验证上述多概念格的横向合并算法的有效性, 我们在 windows 2000 下用 Java 2 编程实现了该算法, 在 P4 1.7G 的计算机上对随机产生的数据采用不同的拆分方案进行了测试, 并和基于对象的渐进式生成概念格算法(如 Godin 算法[5])、基于属性的概念格渐进式生成算法(CLIF. A 算法[10])直接形成概念格进行了比较. 试验中, 形式背景的对象个数、属性个数及其对象属性间存在关系的概率由程序随机产生.

考虑到在实际的数据表中, 数据表的记录(对象)的个数会越来越大, 而字段(属性)的个数往往是有限的. 首先, 我们随机产生 10 个形式背景. 每个形式背景的属性个数固定为 30, 对象属性间存在关系的概率为 0.20, 对象个数从 50 开始, 每次递增 50 个, 直至 500 为止. 对形成的形式背景直接采用 Godin 算法和 CLIF. A 算法构造概念格, 然后把形式背景拆分为 2 个均等的子形式背景和 4 个均等子背景分别采用多概念格的横向合并算法进行试验, 两个拆分方案的合并算法分别记为 HUMCL. p2 和 HUMCL. p4. 试验结果如图 4 所示.

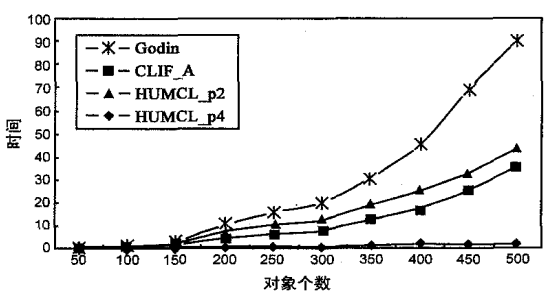

图 4　多概念格的横向合并算法和其他算法的试验比较

从试验结果的对比中, 可以看出:

¹ 随着对象数的增加, Godin 算法所需的时间显著地增加, 而 CLIF. A 算法和 HUMCL. p2 算法的时间虽然也不断增加, 但比 Godin 算法较优.

º 把形式背景的属性均等拆分为二部分, 形成 2 个子背景, 然后分别构建相应的子概念格, 再进行子格的横向合并的 HUMCL. p2 算法和直接采用 CLIF. A 算法构造格的时间相比, 两者相差不大, HUMCL. p2 算法用时稍多些.

» 把形式背景的属性细分为 4 个部分, 采用 HUMCL. p4 算法构造概念格的时间复杂度比其他方法要优越得多.

随着对象的增加, 形式背景规模变大, 形成的概念的数目会随着指数性地增加[5]. 由于 Godin 属于基于对象的算法, 它对对象的变化更敏感些, 其时间复杂度就会随着对象的增加而急剧增加; 而 CLIF. A 算法和 HUMCL 算法是基于属性的算法, 其时间复杂度虽会因形式背景规模扩大而有所增加, 但影响程度较小些.

把形式背景的属性项均等拆分, 分别构造子格再横向合并形式概念格的方案中, 构造概念格的时间就主要有子格的构建时间和子格的合并时间两部分. 由于形式背景的规模与其概念格的规模之间的指数性的关系, 把形式背景拆分为几个小的子形式背景再构造相应子格的时间之和肯定少于直接由大的形式背景构造概念格所需的时间. 但当构造子格所节省的时间不足于抵消子格合并所需的时间时, 采用 HUMCL 算法的总时间复杂度就会比 CLIF. A 算法稍高, 试验中 HUMCL. p2 算法的结果就属于这种情形. 而当形式背景再细分, 构造子格所花的时间会显著减少(指数性地减少), 虽子格合并的总时间会增加, 但算法总时间复杂度可能会有显著改善, 试验中 HUMCL. p4 算法的结果就属于这种情形.

显然地, 对于大的形式背景, 采用形式背景先拆分再子格合并的 HUMCL 算法的效果会更好些.

6　结论及进一步的工作

试验表明, 本文所提出的多概念格的横向合并的 HUMCL 算法是有效的. 如果考虑到从子形式背景构造相应的子格采用多机并行处理, 则采用 HUMCL 算法会明显地优于 CLIF. A 算法和 Godin 算法. 因此, 多概念格的横向合并算法特别适用于对概念格进行分布并行构造.

对形式背景的拆分方案即可采用均等拆分也可采用随机拆分. 明显地, 采用均等拆分虽然简单, 但不一定是最佳方案. 拆分个数不同其效果也不相同, 但并不是拆分越细效果越好. 一种极端的情况, 对于 n 个属性的形式背景如果拆分为 n 个子形式背景, 则 HUMCL 算法实际上就退化为 CLIF. A 算法. 因此, 研究针对不同的形式背景采用何种拆分方案是进一步要研究的一项有意义的工作.

参考文献:

[1] Ganter B, Wille R. Formal Concept Analysis: Mathematical Foundations [M]. Berlin: Springer2Verlag, 1999.

[2] Baltasar Fernandez2Manjon, Alfredo Fernandez2Valmayor. Building ed2ucational tools based on formal concept analysis[J]. Education and In2formation Technologies, 1998, 3(3- 4): 187- 201.

[3] U Krohn, N J Davies, R Weeks. Concept lattices for knowledge manage2ment[J]. BT Technol J, 1999, 17(4): 108- 113.

[4] S O Kuznetsov. Machine learning on the basis of formal concept analysis [J]. Automation and Remote Control, 2001, 62(10): 1543- 1564.

[5] Godin R, Missaoui R, Alaoui H. Incremental concept formation algorithms based on Galois (concept) lattices[J]. Computational Intelligence, 1995, 11(2): 246-267.

[6] Sergei O Kuznetsov, Sergei A Obiedkov. Algorithms for the construction of concept lattices and their diagram graphs[A]. PKDD 2001, LNAI 2168[C]. Freiburg: Springer-Verlag Heidelberg, 2001. 289-300.

[7] 谢志鹏, 刘宗田. 概念格的快速渐进式构造算法[J]. 计算机学报, 2002, 25(5): 490-496.

[8] Yun Li, Zongtian Liu, et al. Theoretical research on the distributed construction of concept lattices[A]. Proceedings of the Second International Conference on Machine Learning and Cybernetics[C]. Xian: Institute of Electrical and Electronics, 2003. 474-479.

[9] Zongtian Liu, Liangsheng Li, Qing Zhang. Research on a union algorithm of multiple concept lattices[A]. RSFDGrC 2003, LNAI 2639[C]. Berlin: Springer-Verlag Heidelberg, 2003. 533-540.

[10] 李云, 刘宗田, 陈 等. 基于属性的概念格渐进式生成算法[J]. 小型微型计算机系统, 2004, 25(10): 1768-1771.

作者简介:

李 云 男, 1965 年 4 月出生于安徽合肥, 副教授, 杨州大学信息工程学院, 现为上海大学计算机学院在读博士研究生, 研究兴趣: 概念格, 数据挖掘等. E-mail: liyun@yzcn.net.

刘宗田 男, 1946 年 6 月出生于山东省莒南县, 上海大学计算机学院教授, 博士生导师, 目前研究领域: 人工智能, 软件工程. E-mail: ztliu@mail.shu.edu.cn.

陈 扬州大学信息工程学院教授, 研究领域: 并行算法, 优化算法, 人工智能等.