第35卷 第21期 计 算 机 工 程 2009年11月
Vol.35 No.21 Computer Engineering November 2009
·软件技术与数据库· 文章编号：1000—3428(2009)21—0062—03 文献标识码：A 中图分类号：TP311.12

基于最大概念的概念格增量构造算法

余 远，钱 旭，钟 锋，李晓瑞
(中国矿业大学机电与信息工程学院，北京 100083)

摘 要：针对增量概念格构造过程中，节点更新和生成元判定效率较低、边更新阶段的复杂度较高等问题，提出基于最大概念的概念格增量构造算法，通过跟踪与概念格中的概念具有相同真实内涵的最大概念，简化生成元的判断过程。该算法缩小了寻找新生节点父节点时的搜索范围，避免对生成元非必要边的判断，提高构造概念格的速度。复杂度分析结果表明，该算法的时间性能优于其他同类算法。

关键词：形式背景；概念格；增量算法；对象内涵

Increment Construction Algorithm for Concept Lattice Based on Maximal Concept

YU Yuan, QIAN Xu, ZHONG Feng, LI Xiao-rui
(School of Mechanical Electronic and Information Engineering, China University of Mining and Technology, Beijing 100083)

【Abstract】 Aiming at the problems during construction process of increment concept lattices such as low efficiency of nodes updating and generation element judgement, this paper presents an increment construction algorithm for concept lattice based on maximal concept which reduces the judgement process of generation element by tracking the largest concept with same connotations of concepts in concept lattice. The algorithm limits the search space while searching the father node of a new generated node, avoids judging unnecessary borders of generation element and improves the speed of concept lattice construction. Complexity analysis results demonstrate that this algorithm has better time performance than other kindred algorithms.

【Key words】 formal context; concept lattice; increment algorithm; object intension

概念格作为形式概念分析理论中的核心数据结构，被广泛用于机器学习、数据挖掘、知识发现和信息检索等领域，在应用过程中，概念格的构造效率一直是制约其应用的关键问题。很多学者对此进行研究并提出相关算法，如NextClosure算法[1]和Godin算法[2]。Godin算法在背景相对疏松时性能较好。当数据集大而稠密时，基于闭包运算的NextClosure算法性能较好。本文提出一种基于最大概念的增量算法——MaxLink，通过跟踪最大概念提高增量式构造概念格时的效率。

1 基本概念

定义1 一个形式背景由2个集合G和M以及G与M间的关系R组成。G的元素称为对象，M的元素称为属性。$(g,m)\in R$或gRm表示对象g具有属性m。

定义2 设A是对象集合G的一个子集，定义
$f(A)=\{m\in M|gRm,\forall g\in A\}$
相应地，设B是属性集合M的一个子集，定义
$g(B)=\{g\in G|gRm,\forall m\in M\}$
容易知道$f\circ g$和$g\circ f$分别是G和M上的闭包算子。

定义3 背景(G,M,R)上的一个形式概念是二元组(A,B)，其中，$A\subseteq G$, $B\subseteq M$，且满足$f(A)=B$, $g(B)=A$，称A是概念(A,B)的外延，B是概念(A,B)的内涵。

对$g\in G$，常用$f(g)$表示$f(\{g\})$，称为对象内涵。对$m\in M$，常用$g(m)$表示$g(\{m\})$，称为属性外延。若$X=(A,B)$是个概念，常用$Intent(X)$表示概念内涵，用$Extent(X)$表示概念外延。

定义4 若(A_1, B_1), (A_2, B_2)是某个背景上的2个概念，且$A_1\subseteq A_2$(等价地，$B_2\subseteq B_1$)，则称(A_1,B_1)是(A_2,B_2)的亚概念，(A_2,B_2)是(A_1,B_1)的超概念，并记作$(A_1,B_1)\leq(A_2,B_2)$，关系“$\leq$”称为是概念的“层次序”(简称序)。(G,M,R)的所有概念用这种序组成的集合称为背景(G,M,R)上的概念格。

定义5 若2个概念X和Y，$X\leq Y$且不存在一个概念Z满足：$Z\neq X$, $Z\neq Y$, $X\leq Z\leq Y$，则称概念X为Y的下近邻(或孩子)，称Y为X的上近邻(或父亲)，记此关系为$X\prec Y$。

2 算法简介

2.1 增量算法

增量算法的基本思想是先将当前要插入对象的对象内涵和格中所有概念内涵相交，然后根据相交的结果采取不同行动，主要解决3个问题：(1)新概念的生成；(2)避免重复生成概念；(3)边的更新。设当前背景为(G,M,R)，对应概念格为L，现要将对象x加入背景，x具有的属性集记为$f(x)$，记此时新背景$(G\cup\{x\},M,R\cup\{x\}\times f(x))$对应的概念格为$L^*$。在由$L$形成$L^*$的过程中，一般将$L^*$中节点分为4类。

定义6 概念$(A,B)\in L^*$称为新的概念，如果$B\notin Intent(L)$。

定义7 概念$(A,B)\in L$是生成元，如果满足$(A,B)=sup\{(C,D)\in L|D\cap f(x)=F$，对某个新概念$(E,F)\in L^*\}$，此时$E=A\cup$

基金项目：教育部重点基金资助项目(107021)
作者简介：余 远(1974—)，男，博士研究生，主研方向：形式概念分析，数据挖掘，机器学习；钱 旭，教授、博士生导师；钟 锋、李晓瑞，博士
收稿日期：2009-05-14 **E-mail：**ywnfallm@163.com

$\{x\}$, $F=B\cap f(x)$。

容易知道新概念和生成元是一一对应的，因此，只要有效定位生成元，就可以顺利解决第(1)个和第(2)个问题。

定义 8 概念$(A,B)\in L$ 是更新的概念，如果 $B\subseteq f(x)$。L 中剩余的节点称为旧概念，这些节点的概念及其父子关系在 L^*中不变。

在边的更新阶段，一般维护一张存有已处理的更新节点和新节点的表，通过查表寻找当前生成新节点的父节点以及取消其生成元不再需要的边。

2.2 算法原理

由上述讨论可知，在增量算法中如果能快速判断旧节点，则生成元和更新节点判断的问题随之解决。在此问题的解决中，文献[3]提出渐进式算法，借助辅助索引树提高判断一个内涵是否是新内涵的速度，并在一定程度上缩小了父节点的搜索范围，但没有很好地解决边的更新问题。本文算法通过跟踪生成元父节点的某类最大概念为解决该问题提供了一条新思路。

命题 1 L 中概念 X 是生成元或更新节点当且仅当 X 的任意父节点 Y 满足

$Intent(Y)\cap f(x)\subset Intent(X)\cap f(x)$

证明：由生成元和更新节点的定义即得必要性。下文证明充分性，即证明 $Intent(X)\cap f(x)\notin Intent(L)$或 $Intent(X)\subseteq f(x)$成立。事实上，若存在一个概念 Z 满足 $Intent(Z)\subseteq f(x)$且 $Intent(X)\cap f(x)=Intent(Z)$，则 $X\leq Z$，因此，必然存在有限概念链 $X=Z_1,Z_2,\cdots,Z_k=Z$ 满足 $Z_1\prec Z_2\prec\cdots\prec Z_k$，由假设 $Intent(X)\cap f(x)=Intent(Z)$和 $Intent(X)\cap f(x)\supset Intent(Z_2)\cap f(x)$($Z_2$ 是 X 的父节点)可知 $k=1$,即 $Z=X$, $Intent(X)=Intent(Z)\subseteq f(x)$，命题成立。

旧节点的判定取决于其父节点，一个新节点生成后，可以通过其生成元的父节点寻找新节点的父节点。因此，可以为每个节点添加一个指针 $MNode$，用于跟踪如下最大概念：

$sup\{Y\in L^*|Intent(X)\cap f(x)=Intent(Y)\cap f(x)\}$

命题 2 如果 X 是生成元，Y 是其对应的新节点，那么 $\{Z\in L^*|Y\prec Z\}\subseteq\{Z\rightarrow MNode|X\prec Z\}$。

证明：如果 $Z\in L^*$且 $Y\prec Z$，那么 $X\prec Z$，可以断言 $Z\in\{Z\rightarrow MNode|X\prec Z\}$，否则，如果存在 X 的父节点 Z_0 满足 $X\prec Z_0\prec Z$ 且 $Intent(Z_0)\cap f(x)\supset Intent(X)\cap f(x)$，由命题 1 可知，$Intent(Z_0\rightarrow MNode)\subseteq f(x)$，即 $Intent(Y)=Intent(X)\cap f(x)\supset Intent(Z_0\rightarrow MNode)\supset Intent(Z)$，与 $Y\prec Z$ 矛盾，因此，命题成立。

命题 2 可以有效化简新节点的父节点搜索空间。在遍历过程中，由于当前处理的生成元的父节点都已设定了最大概念链接，因此只要检查这些链接即可。如果父节点也是生成元，则有如下结论：

命题 3 如果 X 是生成元，Y 是对应的新概念，$X\prec P$，那么 $Y\prec P\rightarrow MNode$。

由命题 3 可知，在边的更新阶段只要判断生成元与其父节点是更新节点的边的状态是否被取消即可。

3 算法设计

表 1 描述了当前背景，图 1 是其对应的概念格，添加对象 6 且 $f(6)=\{b, c, f, i\}$。遍历按概念内涵的势从低到高依次进行，若$\#n$ 是更新节点或生成元，则用$\#n'$表示更新后节点和生成元对应新节点。

表 1 当前背景

R	a	b	c	d	e	f	g	h	i
1	×		×			×		×	
2	×		×				×		×
3	×			×			×		×
4		×	×			×		×	
5		×			×		×		

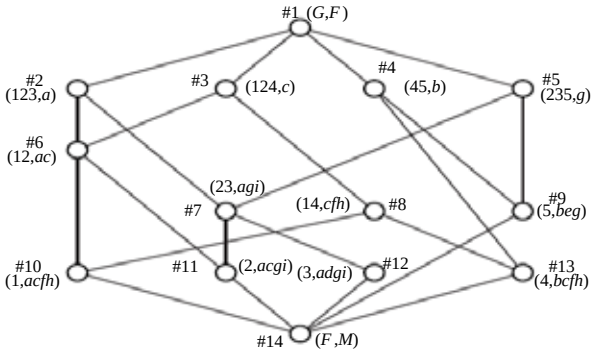

图 1 概念格

第 1 层是根节点#1，更新节点并设置#1 $\rightarrow$ $MNode$=#1'。

第 2 层依次为#2 $\rightarrow$ $MNode$=#1', #3 $\rightarrow$ $MNode$=#3', #4 $\rightarrow$ $MNode$=#4', #5 $\rightarrow$ $MNode$=#1'。

第 3 层为#6 $\rightarrow$ $MNode$=#3'。

第 4 层中#7 满足命题 1 是生成元，#7' $\prec$ #5 $\rightarrow$ $MNode$=#1', #7 $\prec$ #7', #7 $\rightarrow$ $MNode$=#7'。#8 也是生成元，#8' $\prec$ #3 $\rightarrow$ $MNode$=#3', #8 $\prec$ #8'。由于#3 是更新节点，因此要取消#8 与#3 的边，#8 $\rightarrow$ $MNode$=#8', #9 $\rightarrow$ $MNode$=#1'。

第 5 层为#10 $\rightarrow$ $MNode$=#8', #11 $\prec$ #11', #11' $\prec$ #6 $\rightarrow$ $MNode$=#3', #11 $\prec$ #7 $\rightarrow$ $MNode$=#7', #11 $\rightarrow$ $MNode$=#11', #12 $\rightarrow$ $MNode$=#7', #13 $\prec$ #13', #13' $\prec$ #8 $\rightarrow$ $MNode$=#8', #13' $\prec$ #4 $\rightarrow$ $MNode$=#4'，由于#4 是更新节点，因此删除原边，#13 $\rightarrow$ $MNode$=#13'。

第 6 层为#14 $\prec$ #14', #14' $\prec$ #13 $\rightarrow$ $MNode$=#13', #14' $\prec$ #11 $\rightarrow$ $MNode$=#11', #14 $\rightarrow$ $MNode$=#14'。

L 中的概念按概念内涵的势散列存放于结构数组 $Lattice$ 中，$|\ |$表示集合的势。

算法描述如下：

输入 结构数组 $Lattice$ 和添加的对象 x
输出 结构数组 $Lattice$

```
MaxLink(Lattice[]; {x})
BEGIN
 FOR i=0 TO |M|
  FOR each LNode in Lattice[i]
   IF Intent(LNode) ⊆ f(x)
    MODIFY(LNode)
    LNode → MNode=LNode
    IF Intent(LNode)=f(x)
      exit algorithm
    ENDIF
   ELSE
    FOR each parent Par of LNode
     IF Intent(LNode) ∩ f(x)=
Intent(Par → MNode) ∩ f(x)
      LNode → MNode= Par → MNode
      LNode → Old=TRUE
      exitFOR
    ENDIF
  ENDFOR
```

```
LNode → Old=FALSE
IF NOT LNode → Old
  GENERATE New
  FOR each parent Par of LNode
    IF Intent(Par → MNode) ⊆ f(x)
      IF ExistChild(Par → MNode)
        continue
      ELSE
        IF Par → MNode=Par
          CancelEdge(Par,Lnode)
        ENDIF
        AddEdge(Par,New)
      ENDIF
      ELSE
        IF ExistChild(Par → MNode)
          continue
        ELSE
          AddEdge(Par → MNode, New)
        ENDIF
      ENDIF
    ENDFOR
    AddEdge(New,LNode)
    LNode → MNode=New
  ENDIF
 ENDIF
ENDFOR
ENDFOR
END{MaxLink}
```

在上述代码中，第 4 行~第 8 行处理更新节点；对于非更新节点，第 10 行~第 17 行确定该节点的最大概念链接，若该节点不是生成元，则存在命中率问题，只要存在满足第 11 行条件的父节点，就能终止判断并设置其为该节点的最大概念，对于生成元，则产生空循环；第 21 行~第 40 行完成边的更新，函数 ExistChild 用来验证该生成元父节点的最大概念链接是否是新节点的父节点。

4 结束语

MaxLink 算法的时间复杂度主要由 3 个部分组成：(1)对更新节点的处理；(2)对旧节点的处理；(3)对生成元的处理。第(1)部分与更新节点数呈线性关系，第(2)部分和第(3)部分较复杂。寻找最大概念的过程和概念格中节点的父节点个数有关。对旧节点而言，其最大概念链接有且只有一个，因此，按某种字典序调整其父节点的检查顺序可以提高效率。在实际应用中，一个节点的平均父节点个数(与平均子节点个数相同)为 $O(\ln(|G|))$，因此，搜索最大概念过程的复杂度为 $O(\ln(|G|))$。边的更新阶段需要判断每个父节点的 *MNode*，该阶段复杂度为 $O(\ln^2(|G|))$。添加一个新对象的时间复杂度为 $O((\ln^2(|G|))|L|)$，整个增量构造概念格的复杂度为 $O((\ln^2(|G|))|L||G|)$，与文献[2]的复杂度 $O((|G|+|M|)|G||L|)$相比有较大提高。但概念格构造算法应根据应用中数据集的具体情况来选择[3]。在背景稠密或疏松的情况下，不同算法的效率差异很大。因此，笔者将进一步针对具体应用，对当前主流算法进行实验比较[1-3]，以探索该算法的适用条件。现实数据间通常存在某种内在依赖关系(数据挖掘对象之一)，利用此类内在关系提升 MaxLink 算法的性能是笔者下一步工作的目标之一。

参考文献

[1] Ganter B. Two Basic Algorithms in Concept Analysis[R]. Darmstadt, Germany: Technische Hochschule, Tech. Rep.: 831, 1984.

[2] Godin R, Missaoui R, Alaoui H. Incremental Concept Formation Algorithms Based on Galois Lattices[J]. Computation Intelligence, 1995, 11(2): 243-250.

[3] Xie Zhipeng, Liu Zongtian. Fast Incremental Algorithm for Building Concept Lattice[J]. Chinese Journal of Computers, 2002, 25(5): 490-496.

编辑　陈　晖

（上接第 61 页）

(2)测试执行模块：测试驱动器以测试用例为输入，自动调用部署在 Web 服务容器上的待测试 Web 服务，执行测试用例，记录执行结果。

(3)测试评估模块：测试结果评估程序以测试执行过程中记录的测试结果为输入，借助 Test Oracle 对其进行评估，判断待测试 Web 服务中是否存在错误，并生成测试报告。

测试工具 WSTAS 的体系结构如图 1 所示。原型工具 WSTAS 可以测试部署在 apache-tomcat-5.5.17 和 axis-1_4 上的 Web 服务 StackService。

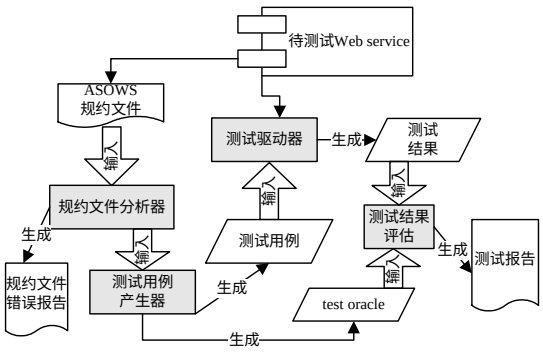

图 1　测试工具 WSTAS 的体系结构

5 结束语

本文提出描述 Web 服务的代数规约语言，讨论代数规约完备性与测试覆盖准则的关系，实现一个基于代数规约测试 Web 服务的原型工具 WSTAS。下一步将研究基于代数规约的 Web 服务测试数据生成策略。

参考文献

[1] Bernot G, Gaudel M C, Marre B. Software Testing Based on Formal Specifications: A Theory and a Tool[J]. Software Engineering Journal, 1991, 6(6): 387- 405.

[2] Gonnon J, McMullin P, Hamlet R. Data-abstraction Implementation, Specification and Testing[J]. ACM Transactions on Programming Languages and Systems, 1981, 3(3): 211-223.

[3] Chen Huoyan, Tse T H, Chen T Y. TACCLE: A Methodology for Object-oriented Software Testing at the Class and Cluster Levels[J]. ACM Transactions on Software Engineering and Methodology, 2001, 10(1): 56-109.

[4] Kong Liang, Zhu Hong, Zhou Bin. Automated Testing EJB Components Based on Algebraic Specifications[C]//Proc. of COMPSAC'07. Beijing, China: [s. n.], 2007.

[5] Yu Bo, Kong Liang, Zhang Yufeng, et al. Testing Java Components Based on Algebraic Specifications[C]//Proc. of International Conference on Software Testing, Verification, and Validation. Lillehammer, Norway: [s. n.], 2008.

编辑　顾姣健