第33卷 第16期 计 算 机 工 程 2007年8月
Vol.33 No.16 Computer Engineering August 2007
·软件技术与数据库· 文章编号：1000—3428(2007)16—0050—03 文献标识码：A 中图分类号：TP391

MFCPLG：微阵列数据中频繁闭合模式挖掘

金 波[1]，缪裕青[1,2]
(1. 桂林电子科技大学计算机系，桂林 541004；2. 中国科学技术大学计算机科学与技术系，合肥 230026)

摘 要：微阵列数据集行少列多的特征，使得传统基于列枚举空间的算法应用于其中进行频繁闭合模式挖掘时其复杂性迅速增长。基于行枚举的CARPENTER算法较好解决了该问题。但CARPENTER算法使用映射转置表(TT)来完成频繁闭合模式完全集的挖掘效率不高。该文在CARPENTER算法基础上，提出LG-tree数据结构，并基于此结构提出挖掘频繁闭合模式的新算法MFCPLG。真实数据集的实验表明，MFCPLG算法的时间性能优于CARPENTER算法。

关键词：微阵列数据集；频繁闭合模式；MFCPLG；LG-tree

MFCPLG: Mining Frequent Closed Patterns in Microarray

JIN Bo[1], MIAO Yu-qing[1,2]
(1. Dept. of Computer Science, Guilin University of Electronic Technology, Guilin 541004;
2. Dept. of Computer Science and Technology, University of Science and Technology of China, Hefei 230026)

【Abstract】 Because microarray dataset contains a large number of columns and a small number of rows, mining the complete set of frequent closed patterns in microarray poses a great challenge for traditional frequent closed patterns mining algorithms which are based on column-enumerate. CARPENTER, based on row-enumerate, has addressed this problem. However, CARPENTER is inefficient by using the transposed table (TT) to mine the complete set of frequent closed patterns in microarray. A new algorithm, MFCPLG, is proposed inspired by CARPENTER, and a LG-tree structure is suggested. Several experiments are performed on real-life microarray data to show that the MFCPLG algorithm is faster than the CARPENTER algorithm.

【Key words】 microarray dataset; frequent closed patterns; MFCPLG; LG-tree

微阵列数据集通常包含上万列(基因)和相对少量行(采样条件)，如很多基因表达数据集包含10 000~100 000列，却只有100~1 000行。现存的大部分频繁闭合模式挖掘算法对项(列)枚举空间进行搜索，这些算法应用于传统数据库时性能较好，但对于高维数微阵列数据集，此类算法性能急剧下降。因为它们的运行时间随行平均长度增加呈指数增长，微阵列数据集行少列多的特点使得采用行枚举算法看起来更为合理。

文献[1]提出的CARPENTER算法正是解决微阵列数据集频繁闭合模式挖掘问题的，它采用深度优先搜索行枚举空间策略，同时结合一些修剪技术，进一步减小了搜索空间。在CARPENTER算法基础上，提出新算法MFCPLG(mining frequent closed patterns by Line-Gene tree)，主要工作如下：

(1)提出新的数据结构LG-tree(Line-Gene tree)，用来取代CARPENTER算法中递归生成的TT表(transposed table)。

(2)提出适用于LG-tree结构的单路径修剪策略，通过修剪行枚举树从而减小搜索空间。

实验表明，MFCPLG算法的时间性能优于CARPENTER算法。

1 频繁闭合模式和CARPENTER算法

1.1 频繁闭合模式

定义1 设$I=\{i_1,i_2,\ldots,i_m\}$为项集，$R=\{r_1,r_2,\ldots,r_n\}$为行集，数据集$D=\{(r_1,X_1), (r_2,X_2),\ldots,(r_n,X_n)\}$，其中$r_k$表示行号，$X_k$称为模式且$X_k\subset I$。如图1，$I=\{a,b,c,d,e,f,g,h,l,o,p,q,r,s,t\}$，D包含5个2元组，行号分别为1，2，3，4，5。

r_i	X_i
1	a,b,c,l,o,s
2	a,d,e,h,p,l,r
3	a,c,e,h,o,q,t
4	a,e,f,h,p,r
5	b,d,f,g,l,q,s,t

图1 数据集D

为简便起见，本文用“123”表示行集$\{r_1,r_2,r_3\}$，用“aeh”表示模式$\{a,e,h\}$。

定义2 给定模式I’，R’(I’)表示包含模式I’的行集，|R’(I’)|表示模式I’的支持数；

定义3 给定行集R’，I’(R’)表示R’中各行包含的共同模式；

定义4 给定模式I’，如果不存在同时满足如下条件的I’’：(1)I’$\subset$I’’；(2)|R’(I’)|=|R’(I’’)|，那么模式I’是闭合的；

定义5 给定模式I’，如果满足：(1)I’闭合；(2)|R’(I’)|$\geq$minsup，其中，minsup是用户定义的最小支持数，那么模式I’是频繁闭合模式。

问题定义：给定数据集D，挖掘出所有频繁闭合模式，其中，D满足|R’|<<|I’|。

1.2 CARPENTER算法

文献[1]中提出的行枚举树结构，图2、图3是对应于图1的行枚举树和TT表。

作者简介：金 波(1983－)，男，硕士研究生，主研方向：数据挖掘；缪裕青，副教授、博士研究生

收稿日期：2006-11-06 **E-mail：**jhyster83@tom.com

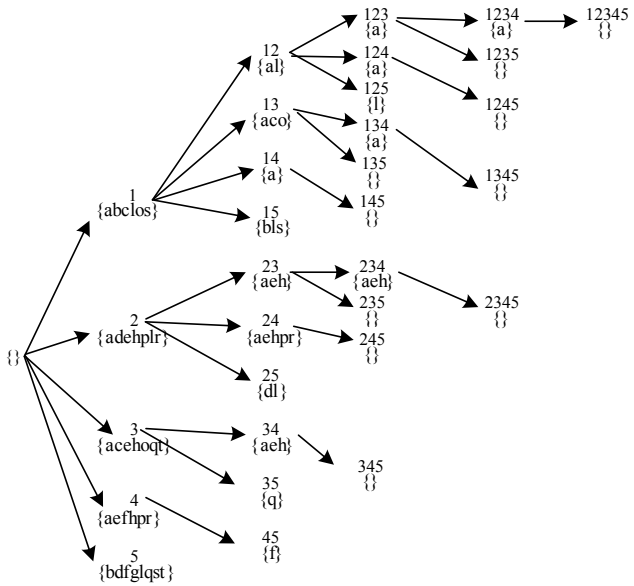

图 2　行枚举树

f_i	$R(f_i)$
a	1,2,3,4
b	1,5
c	1,3
d	2,5
e	2,3,4
f	4,5
g	5
h	2,3,4
l	1,2,5
o	1,3
p	2,4
q	3,5
r	2,4
s	1,5
t	3,5

图 3　映射转置表 TT|ø

图 3 为初始映射转置表TT|ø。映射转置表是$(f_i,R(f_i))$的二元组集合，本文用i表示项(基因)f_i，用R(i)表示项i所对应的行集。行枚举树的每个结点包含特定行集R及其相应模式I(R)。由于微阵列数据集行少列多的特点，与传统项枚举算法相比，行枚举树大大减小了搜索空间。在CARPENTER算法中递归产生的转置表代表了图中的每个结点，如$TT|_{\{2,3\}}$代表行枚举树中的结点{2,3}，如图 4。

i	R(i)
a	4
e	4
h	4

图 4　$TT|_{\{2,3\}}$

CARPENTER算法通过对D的 1 次扫描，构造初始转置表TT|ø，它代表行枚举树的根结点。此时，频繁闭合模式挖掘对象由D转变成TT|ø。对于转置表$TT|_X$中的每个行号i，通过扫描$TT|_X$中的所有记录，取出包含行号i的那些记录，删除这些记录中小于i的行号后得到$TT|_{X\cup\{i\}}$。

以上是个递归过程，直到遍历完行枚举树，算法完成。在每次生成$TT|_X$后CARPENTER算法都使用 3 步修剪策略来修剪行枚举树：

(1)若I(R)所能达到的最大支持数小于minsup，则修剪$TT|_R$的子孙结点；

(2)若I(R)所有项在行集Y中的每行都出现，则修剪$TT|_R$包含行集Y的子孙结点；

(3)若新发现的频繁闭合模式 F(X)已经被发现过，则返回。

2　LG-tree 和 MFCPLG 算法

CARPENTER算法的主要工作是递归对$TT|_X$扫描，生成下级转置表$TT|_{X\cup\{i\}}$。实验表明，大部分的CPU时间都被扫描$TT|_X$占用；而且转置表中很多记录都包含共同前缀，可以通过对时间和空间的优化来提高算法性能。本文借用了FP-tree部分构造思想提出了新的数据结构LG-tree并在行枚举树中以LG-tree来代替TT表。文献[2]提出了FP-tree数据结构，FP-tree是数据集的压缩存储形式，各交易以共享前缀的方式存放在FP-tree中，保存在树各结点中的项从树根到树叶以支持数递减的顺序存放。与FP-tree关联的还有一个项头表。频繁项及其支持数按支持数递减的顺序存放在项头表中，项头表中的记录还包含一个链表头，链表链接了与该条记录中项相应的FP-tree中的结点。实验表明[2~4]，FP-tree是用于挖掘频繁(闭合)模式最为有效的数据结构之一。

2.1　LG-tree

2.1.1　由 TT|ø 构造 LG-tree|ø

LG-tree 存储了所有频繁闭合模式的信息。以图 3 为例建立的初始 LG-tree|ø 如图 5。

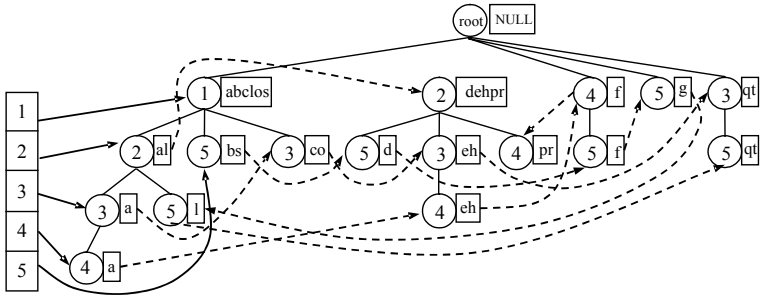

图 5　LG-tree|ø

由图 5 可知，TT|ø 中的每条记录以共享前缀的方式存储于 LG-tree|ø 中，这就减少了行号的存储空间。树中每个结点包含某特定行号及其相应模式，同行号的结点以链表形式连接。如插入 TT|ø 前 3 条记录后的 LG-tree|ø 如图 6。

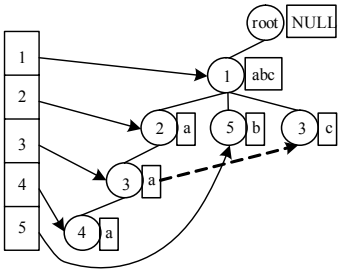

图 6　插入前 3 条记录后的 LG-tree|ø

LG-tree|ø项头表中的每条记录包含行号和链表头，链表链接了LG-tree|ø中与该记录行号相应的结点。树根结点相应的模式计数信息则隐含在$LG\text{-tree}|_X$的行集X中，如对于$LG\text{-tree}|_{\{23\}}$，行集 23 的长度 2 即为隐含的支持数信息。

2.1.2　由$LG\text{-tree}|_X$构造$LG\text{-tree}|_{X\cup\{i\}}$

在完成LG-tree|ø构造之后，频繁闭合模式的挖掘对象由TT|ø转化为LG-tree|ø。由$LG\text{-tree}|_X$构造$LG\text{-tree}|_{X\cup\{i\}}$步骤如下，其中，i为$LG\text{-tree}|_X$项头表中的某行号：

(1)扫描$LG\text{-tree}|_X$ 1 次，只扫描$LG\text{-tree}|_X$中以i作为根结点的各子树，建立$LG\text{-tree}|_{X\cup\{ri\}}$的项头表，项头表由大于i的行

集按递增的顺序排列；

(2)第 2 次扫描LG-tree|$_X$，同样只扫描以i作为根结点的各子树，将各子树合并：根结点行号为i；保持各子树原有父子关系不变；子树同层同行号结点结合成新结点时，新结点行号不变，其相应模式为两结点相应模式的并集。子树合并有效地控制了树的规模。

由LG-tree|ø构造的LG-tree|$_{\{2\}}$如图 7。

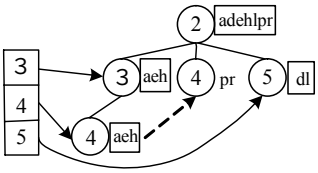

图 7　LG-tree|$_{\{2\}}$

2.2　MFCPLG 算法

MFCPLG 算法的描述如下：

Algorithm CreateOriginTree(TT|ø)

输入　初始转置表 TT|ø

输出　初始 LG 树 LG-tree|ø

方法：CreateOriginTree(TT|ø)。

步骤：

(1)FOR 数据集所有行号，建立项头表；

(2)FOR TT|ø 中的每条记录，插入到 LG-tree|ø 中。

Algorithm MFCPLG(LG-tree|$_X$,R',FCP)

输入　LG-tree|ø

输出　频繁闭合模式完全集，FCP

方法：初始化FCP=ø,R为项头表中所有项的集合，调用MFCPLG (LG-tree|ø,R,FCP)；MFCPLG (LG-tree|$_X$,R',FCP)。

{

(1)扫描LG-tree|$_X$1 次(只需要扫描LG-tree|$_X$中以i为根结点的子树)，以至少出现一次的行号建立LG-tree|$_{X\cup\{i\}}$的项头表；

(2)第 2 次扫描LG-tree|$_X$，合并各子树，建立LG-tree|$_{X\cup\{i\}}$；

(3)修剪 1：令R'为LG-tree|$_{X\cup\{ri\}}$项头表中行号的集合，IF |R'|+|X|<minsup返回；

(4)修剪 2：IF LG-tree|$_X$是单路径树，R'=ø，FOR路径中的每个结点x，IF |I(x)|=|I(x->child)|，那么x=x->child，否则执行修剪 3；ELSE 对LG-tree|$_X$根结点，执行修剪 3；其中x->child表示结点x的孩子结点；

(5)修剪 3：IF I(x)∈FCP，返回；ELSE IF |x|+j≥minsup，将I(x)加入到FCP中；其中j为LG-tree |$_X$中x对应结点在树中的层次，设根结点的层次为 0，以此类推；

(6)FOR EACH i∈R',

R'=R'-{i}

MinePattern(LG-tree|$_{X\cup\{i\}}$,R',FCP)。

}

首先MFCPLG算法通过对数据集D的 1 次扫描，构造TT|ø，接着扫描TT|ø中的每条记录，构造LG-tree|ø，即调用CreateOriginTree。此时，挖掘对象由数据集D转变成LG-tree|ø。对LG-tree|$_X$项头表中的每个行号i (MinePattern语句 6)，执行 2.1.2 中的两个步骤完成LG-tree|$_{X\cup\{i\}}$的构造。

以上是一个深度优先递归过程，即按照LG-tree|ø, LG-tree|$_{\{1\}}$,…,LG-tree|$_{\{12345\}}$,LG-tree|$_{\{1235\}}$,…,LG-tree|$_{\{5\}}$ 的顺序递归生成LG-tree|$_X$，直到遍历完行枚举树，算法完成。每次生成LG-tree|$_X$之后进行 3 步修剪以减小搜索空间：

修剪 1(MinePattern语句 3) 若I(X)所能达到的最大支持数小于minsup，则修剪行枚举树中LG-tree|$_X$ 对应结点的所有子孙结点；

修剪 2(MinePattern语句 4，单路径修剪) 若LG-tree|$_X$为单路径树，修剪行枚举树中LG-tree|$_X$ 对应结点的所有子孙结点，同时对该树中的每个结点执行修剪 3 操作。

引理(单路径修剪) 若LG-tree|$_X$ 为单路径树，则行枚举树中LG-tree|$_X$ 对应结点的所有子孙结点对应的LG-tree必为单路径树。

证明　由 2.1.2 节LG-tree|$_X$构造LG-tree|$_{X\cup\{i\}}$的过程可知，若LG-tree|$_X$为单路径树，则LG-tree|$_{X\cup\{i\}}$必为单路径树。以此类推，行枚举树中LG-tree|$_X$ 对应结点的所有子孙结点都为单路径树。修剪策略 2 由引理支持。

单路径树LG-tree|$_{\{23\}}$如图 8 所示。

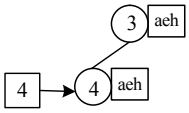

图 8　LG-tree|$_{\{23\}}$

根据修剪 2，修剪行枚举树中LG-tree|$_{\{23\}}$对应结点的所有子孙结点，同时对行号 3、行号 4 对应的结点进行修剪 3 操作。其中结点 3 对应模式aeh的支持数为 2，结点 4 对应模式aeh的支持数为 3，两个结点对应模式相等而结点 4 对应模式的支持数较大，所以，只需要对结点 4 进行修剪 3 操作即可。

修剪 3(MinePattern 语句 5) 若 I(x)已存在，则返回；若不存在且 I(x)的支持数满足 minsup，则将 I(x)及其支持数加入 FCP。

3　实验评估

本节对 MFCPLG 算法的性能进行评估。实验的运行环境为 Intel 1500MHz P4 CPU，256MB SDR 内存及 Windows2000 操作系统，算法以 C++语言编写运行。本文严格按照文献[1]中的算法描述编写 CARPENTER 算法。

实验基于两个真实数据集：Erythroid 和 Mlung，其中 Erythroid 包含 6 个 tissue samples，每个 sample 由 255 个基因描述，而 Mlung 包含 12 个 tissue samples，每个 sample 由 217 个基因描述。

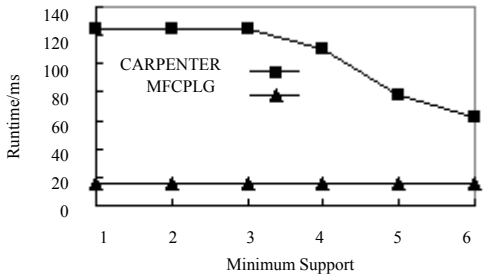

图 9　Erythroid 数据集

图 9 显示了算法以 Erythroid 数据集作为输入的运行情况。在 sample 较少的情况下 MFCPLG 算法处理不同支持数时运行时间的差距很小。整体上来说 CARPENTER 算法处理不同支持数的运行时间都多于 MFCPLG 算法，而且随着支持数的递减，运行时间递增。图 10 显示了算法以 Mlung 数据

(下转第 55 页)