文章编号:1000 - 582X(2002)01 - 0075 - 05

渐进式序列模式挖掘算法 IMSP 分析*

陈金玉,曹长修
(重庆大学 自动化学院,重庆 400044)

摘　要:序列模式挖掘是数据挖掘中最重要的研究课题之一。基于记录频繁集各元素的 Ctid 表的基础上,有研究者提出了一种渐进式序列模式挖掘算法 IMSP,目的是当支持度保持不变,而数据库变化不大时,如何利用前次的结果和中间结果,以加速本次挖掘过程。笔者深入分析了算法 IMSP 结构,指出该算法在时间复杂度、挖掘规则的完备性上的不足,同时也指出利用该算法所可能得到的错误结果。

关键词:序列模式;算法 IMSP;频繁集
中图分类号:TP 18　　**文献标识号**:A

序列模式(Sequential Pattern)是 R、Agrawal[1] 首先提出的,它研究的是要从一个或多个事件序列中发现经常出现的事件集合。目前,绝大多数序列模式挖掘算法都采用一种宽度优先的搜索模式、如 AprioriAll、AprioriSome 和 DynamicSome[2],GSP[3] 等。算法一般分为两个阶段:1)频繁序列的发现;2)规则的产生。算法的计算量主要集中在第一阶段上。

但是,上述算法都有两个公共的前提:1) 交易数据库中的元组数不变;2) 预先指定的最低支持度不变。如果这两个前提有一个不成立,应该怎么办? 这是一个有着实际意义的问题。周斌、吴泉源[4] 基于记录频繁集各元素的 Ctid 表的基础上、提出了一种渐进式序列模式挖掘算法 IMSP,目的是当支持度保持不变,而数据库变化不大时、如何利用前次的结果和中间结果,以加速本次挖掘过程。

笔者深入分析了算法 IMSP 结构,指出该算法在时间复杂度、挖掘规则的完备性上的不足,同时也指出利用该算法所可能得到的错误结果。

1　定义及相关例子

为使本文能在概念上自包,现给出所有可能涉及到的定义[4]。它们将体现在对算法 IMSP 的理解上,同时它们还是本文立论的基础。

定义 1　非空集合 $I = \{i_1, i_2, \cdots, i_m\}$ 称为项集,其中 i_k 称为项。

定义 2　序列是项集的有序表,记为 $\alpha = \alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_n$,其中 $\alpha_k \subset I (k = 1,2,\ldots,n)$、含有 k 个项的序列的长度为 k,称为 k 序列($k = \sum |\alpha_i|$)。

定义 3　令序列 $\alpha = \alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_n$,序列 $\beta = \beta_1 \rightarrow \beta_2 \rightarrow \cdots \rightarrow \beta_m$. 若存在整数 $i_1 < i_2 < \cdots < i_n$ 使得 $\alpha_1 \subset \beta_{i1}, \alpha_2 \subset \beta_{i2}, \cdots, \alpha_n \subset \beta_{im}$,则称序列 α 是序列 β 的子序列,或序列 β 包含序列 α. 在一组序列中,如果某序列 α 不包含在其它任何序列中,则称 α 是该组中最长序列(Maximal sequence)。一组序列中可能有多个最长序列。

定义 4　函数 $c: I \rightarrow I^+$(正整数集)、称为项集 I 的标识函数。函数 $t: I \rightarrow I^+$,称为项集 I 的时间函数,它表示项集 I 对应的时刻。事务 T 是由标识函数值及时间函数值标识的项集,标识函数 c 称为事务的标识号,记为 $c(T)$。时间函数值称为事务发生的时刻.记为 $t(T)$. 数据以 $(c(T), t(T))$ 表的形式存储,简称为 ctid 表。事务序列的集合称为事务数据库。

定义 5　给定序列 α、事务数据库 DB,若 D 为 DB 中所有事务标识号的集合模,即 $D = |\{c(T) | T \in DB\}|$,则 $\sigma(\alpha, DB) = |\{c(s) |$ 序列 s 包含 α 且 $s \in DB\}| / D$,称为 α 在 DB 上的支持度。支持度大于最小支持度(min_sup) 的 k - 序列,称为 DB 上的频繁 k -

* 收稿日期:2001-06-08
基金项目:国家教育部博士生基金资助项目(98061117)
作者简介:陈金玉(1970 -),福建省仙游县人,重庆大学博士研究生。主要研究方向:数据挖掘与企业互联网。

序列,记为 F_k^{DB}。

定义 6 序列模式是形如 $\alpha \rightarrow \beta[\sigma,\delta]$ 规则的,其中 $\alpha \rightarrow \beta$ 为最长频繁序列,$\sigma(\alpha \rightarrow \beta, DB)$ 称为规则的支持度。$\delta = \sigma(\alpha \rightarrow \beta, DB)/\sigma(\alpha, DB)$ 称为规则的置信度。置信度大于最小可信度用户给定阀值,记作(min_conf)的规则被认为是用户感兴趣的规则。序列模式挖掘就是从事务数据中找出满足用户指定的序列模式的过程。

为算法讨论及理解的方便,就文献[4]中的例子展开讨论。如下图所示,(a)中 DB 为时刻 15 时挖掘的数据库。(b)中 DB' 为时刻 25 时挖掘的数据库。其中 A—H 为商品(条码数据),T 为顾客交易事务,$c(T)$ 为顾客号,$t(T)$ 为事务号(图 1 中用 T,c,t 表示)。且设 $db^+ = DB' - DB$, $db^- = DB - DB'$, $D = |\{c(T) \mid T \in DB\}|$, $D' = |\{c(T) \mid T \in DB'\}|$

(a) DB

c	t	T
1	10	$\{C\ D\}$
1	15	$\{A\ B\ C\}$
2	10	$\{D\}$
2	15	$\{A\ B\ F\}$
3	15	$\{A\ B\ F\}$
4	10	$\{D\ G\ H\}$

(b) DB'

c	t	T
1	15	$\{A\ B\ C\}$
1	20	$\{A\ B\ F\}$
1	25	$\{A\ C\ D\ F\}$
2	15	$\{A\ B\ F\}$
2	20	$\{E\}$
3	15	$\{A\ B\ F\}$
4	20	$\{B\ F\}$
4	25	$\{A\ G\ H\}$

(c) db^-

c	t	T
1	10	$\{C\ D\}$
2	10	$\{D\}$
4	10	$\{D\ G\ H\}$

(d) db^+

c	t	T
1	20	$\{A\ B\ F\}$
1	25	$\{A\ C\ D\ F\}$
2	20	$\{E\}$
4	20	$\{B\ F\}$
4	25	$\{A\ G\ H\}$

图 1 事务数据库

在本文的讨论中,笔者取 min_sup = 25%。现给出上图 1(a) DB 中 F_k^{DB} 的中各元素及其 Ctid 表。

$F_1^{DB} = \{(A)[(1,15)(2,15)(3,15)],$
$(B)[(1,15)(2,15)(3,15)]$
$(D)[(1,10)(2,10)(4,10)],$
$(F)[(2,15)(3,15)]\}$

$F_2^{DB} = \{(AB)[(1,15)(2,15)(3,15)],$
$(AF)[(2,15)(3,15)]$
$(BF)[(2,15)(3,15)],$
$(D \rightarrow B)[(1,15)(2,15)],$
$[(1,15)(2,15)]\}$

$F_3^{DB} = \{(ABF)[(2,15)(3,15)],$
$(D \rightarrow AB)[(1,15)(2,15)]\}$

$F_4^{DB} = \phi$

2 挖掘算法 IMSP 的分析

算法 IMSP 是基于两个前题下展开的:

前题 1 $db^+, db^- \ll DB, DB', d = (D' - D) \geq 0$ 以及 min_sup' = min_sup

前题 2 前次挖掘出的 $F_k^{DB}(k = 1,\cdots,n)$ 中各元素的 Ctid 表为已知。

算法 IMSP 的基本思想是将 DB' 中的频繁 k – 序列划分为两部分考虑:

1) $Old_k = \{x \mid x \in F_k^{DB'}$ 且 $x \in F_k^{DB}, k = 1,\cdots,n\}$

2) $New_k = \{x \mid x \in F_k^{DB'}$ 且 $x \notin F_k^{DB}, k = 1,\cdots,n\}$

其算法框架是首先产生候选集,然后扫描数据库,反复迭代。由于对上面两部分的处理不同,文[4]将算法的每次迭代过程分为两个阶段讨论。其中最重要的是该文提出的时序连接算法 Temp_join (α,β)

● 算法 Temp_join(α,β) 分析

以下给出原算法[4]:

算法 Temp_join(α,β)

① for all tid $t_b \in \beta.Ctid$ do

② if $\exists$ tid $t_a \in \alpha.Ctid$ 使得 $t_b > t_a$ 且 time_cons(t_a,t_b) = TRUE then

$(\alpha \rightarrow \beta).\sup^{++}$ 并记录(cid,t_b);

③ if $\exists$ tid $t_a \in \alpha.Ctid$ 使得 $t_b = t_a$ 且 time_cons(t_a,t_b) = TRUE then

$(\alpha\beta).\sup^{++}$ 并记录(cid,t_b);

④ for all tid $t_a \in \alpha.Ctid$ do

⑤ if ∃ tid $t_b \in \beta.Ctid$ 使得 $t_a > t_b$ 且 time_cons(t_a, t_b) = TRUE then

($\beta \rightarrow \alpha$).sup++并记录(cid, t_a)；

现估计算法 Temp-join 的时间复杂度。设 $|\alpha|$，$|\beta|$ 为 α，β 中的 Ctid 表元素的个数，则其最坏情况的时间复杂度为 $2|\alpha||\beta|$，这是因为当取定 $t_b \in \beta.Ctid$ 时，t_a 要扫描 β 中 Ctid 表的缘故。注意到算法第④、⑤步，它是为了防止形如 (A)[(1,15)(2,15)(3,15)]，(B)[(1,15)(2,15)(3,15)] 时，$A \rightarrow B$ 被估算两次而设计的。但从算法 Temp_join 本身，还有一些被重复估计的情况没有被剥除，如构造 Ctid 表如下：{(AB)[(1,20)]，(AF)[(1,20)]} 以及 Ctid 表：{(F)[(1,20)]，(B)[(1,20)]}。可以看到，当这两个 Ctid 表进行时序连接时，由算法 Temp_join，(ABF)[(1,20)] 将不可避免的被估计两次，这是不对的。实际上，算法在支持度的计算上也有值得商榷的地方。如(A)[(1,15)] 与 (B)[(1,25)，(1,30)] 的时序连接，由算法 Temp_join 可知，($A \rightarrow B$).sup 将增加 2，而实际上($A \rightarrow B$).sup 只能添加 1。

就连接而言，算法也有令人费解的地方。如($A \rightarrow B$)[(1,20)] 与 (C)[(1,20)] 的时序连接，显然得到的 3－候选序列不会是($AC \rightarrow B$)[(1,20)]，而应该是($A \rightarrow BC$)[(1,20)]。但是，原算法没有体现不可能集的剔除。否则形如($A \rightarrow B$)[(1,20)]，甚至($D \rightarrow A \rightarrow B$)[(1,20)] 与 (C)[(1,10)] 的连接问题，则生成的新候选集就难以想象了。出现这一不确定性和 Ctid 表的设计中没有体现 A、D 的位置有关。

● 算法 Find_New(F_{k-1}^{DBF}) 及引理 3 分析

现分析算法 Find_New(F_{k-1}^{DBF}) 的结构。首先给出其原算法[4] 如下：

算法 Find_New(F_{k-1}^{DBF})

1) C_k^{DBF} := Apriori-gen(F_{k-1}^{DBF})；

2) C_k^{DBF} := C_k^{DBF} − F_k^{DBF}；

3) C_k^{DBF} := Prune_by_lemma_3(C_k^{DBF})；

4) for each $X \in C_k^{DBF}$ do

Temp_join(X.head, X.tail)；/ * X.head $\in F_{k-1}^{DB}$，X.tail $\in F_1^{DB}$ 且 X = X.head + X.tail * /

5) New_k := {$X \in C_k^{DBF}$ | X^{DBF}.sup > (min_sup × D')}；

6) return New_k；

为了能利用算法 Temp_join(α，β) 来计算支持度，算法 Find_New(F_{k-1}^{DBF}) 先利用 Apriori_gen 算法来生成候选集，令人不解的也在这里。我们知道，Apriori_gen 算法是 R. Agrawal[1] 在研究关联规则时引入的，它不考虑时间的约束。在本文中有很强的时间约束性下，如何相应地使用应加以交待。文[4] 不加说明的引用，是值得商榷的。因为如(AB)[(1,20)] 与($A \rightarrow F$)[(1,20)]，($A \rightarrow D$)[(1,30)]，如何运用 Apriori_gen 算法？甚至更为复杂的序列如($A \rightarrow BC \rightarrow D$)[(1,20)] 与 ($CA \rightarrow BE \rightarrow D$)[(1,20)]，又是如何生成下级候选集的？这不是简单运用 Apriori_gen 算法能对就得到的。

为利用算法 Temp_join(α，β) 来估计支持度与记录其 Ctid 表，注意到算法 Find_New(F_{k-1}^{DBF}) 是从候选集 C_k^{DBF} 中选择出 X.head，X.tail 的，这就有两个疑问：第一，X.head，X.tail 如何选法？如有候选元素 ($AB \rightarrow CD$)[(1,20)(2,30)] 等时，它的 X.head 及 X.tail 能唯一确定吗？不能。第二，这种做法有无必要？实际上，可以把记录其 Ctid 表的工作放在候选集的生成中同时完成。这时，对支持度的估计是非常简单的。见文[6]，这样它也同时避免如何选择 X.head，X.tail 的麻烦，而这意味着算法的大规模改写。

另一方面，由于候选集由集 F_{k-1}^{DBF} 用算法 Apriori_gen 生成，则从时间复杂度而言，算法 IMSP 把频繁集 F_{k-1}^{DBF} 分为 Old_k 与 New_k 的意义，就不是很大了。笔者还注意到算法中出现的另一情况，即算法 Find_New(F_{k-1}^{DBF}) 的第③步，它的作用是根据引理 3 将 C_k^{DBF} 中不可能成为频繁集的元素删去，以加速计算过程。现给出原引理 3：

引理 3[4]　对于 $\forall X \notin F_k^{DB}$，若 X^{DB^-}.sup $\leqslant d \times$ min_sup，则 $X \notin F_k^{DBF}$

这个引理在文[4] 中的作用很是微妙。可以看出，正是这个引理，才引出了前题 1 中的条件 $d = (D' - D) \geqslant 0$。其实文[4] 中也提到，它的思路是来自文[5] 的思想，不同的是其考虑数据库中数据既可增加也可减少的一般情况。笔者以为，不存在数据减少的情况。因为一般而言，数据挖掘的对象是数据仓库，而数据仓库的定义是：一个面向主题的、集成的、非易失的且随时间变化的数据集合。问题就很清楚了，数据库数据的可减少的情况在实践中出现的可能性较为牵强。当然，当局限在某时间进行数据挖掘，文[4] 的思路就有用处了。但这时前题 1 中的条件 $d = (D' - D) \geqslant 0$ 就有

点令人不解了。因为这时的顾客 c 有可能变化，数量也有变少的可能，即有可能 $d \leqslant 0$。就算不考虑以上的情况，引理 3 的作用亦很有限。因为当顾客数不变，即当 $d = 0$ 时，引理 3 就失掉了用处。

● 算法 Find_Old(F_k^{DB})、引理 1 及定理 1 分析

现分析算法[4] Find_Old(F_k^{DB})，先给出算法：

算法 Find_Old(F_k^{DB})

1) for each $X \in F_k^{DB}$ do {

if ($k = 1$) {

2) X^{db^+}.Ctid := Fetch_Ctid_from_$db(X, db^+)$;

3) X^{db^-}.Ctid := Fetch_Ctid_from_$db(X, db^-)$;

4) X^{DB}.sup = Compute_sup_by_lemma1(X);}

5) else if ($k >= 2$) Temp_join(X.head, X.tail);}

/* X.head $\in F_{k-1}^{DB}$, X.tail $\in F_1^{DB}$ 且 X = X.head + X.tail */

6) Old_k := {$X \in F_k^{DB}$ | X^{DB}.sup > (min_sup × D')}

7) $Lose_k$:= {$X \in F_k^{DB}$ | X^{DB}.sup ≤ (min_sup × D')};

8) F_{k+1}^{DB} := Prune_by_lemma_2(F_{k+1}^{DB}, $lose_k$);

9) return Old_k;

算法 Find_Old(F_k^{DB}) 中的 Temp_join 部分的不足如前面部分所述。现分析其另外的不足。其实按文[4]的思路，算法 Find_Old(F_k^{DB}) 中第 5) 步中 X 的取值应改为 X.head $\in$ Old$_{k-1}^{DB}$，第 8) 步应改为

For ($t = k, F_t^{DB} \neq \phi, t++$)

F_{t+1}^{DB} := Prune_by_lema_2(F_{t+1}^{DB}, $lose_t$)

更为合理。我们关心的是第 4) 步，文[4]认为其正确性可由引理 1 保证。现给出引理 1：

引理 1[4] 令 X^{DB}.ctid 表示序列 X 在数据库 DB 中的 Ctid 表，对任意序列 $X \in F_k^{DB}$ 有

$$\sigma(X, DB') = |X^{DB}.\text{ctid} - X^{db^-}.\text{Ctid}) \cup X^{db^+}.\text{Ctid}| / D'$$

文[4]没有给出证明，只是说："由定义 4 及定义 5 立证"。不知文[4]是如何证明的，但可以肯定其结论是错误的。现构造反例如下图 2 所示(为了方便，略去无关的数据)。

C	t	T
1	10	|A|
1	20	|B|

(a) DB

c	t	T
1	20	|B|

(b) DB'

c	t	T
1	10	|A|

(c) db^-

(d) db^+

图 2 引理 1 反例数据库

可知$(A \rightarrow B)^{DB}$.Ctid = (1,20), $(A \rightarrow B)^{db^-}$.Ctid = ϕ, $(A \rightarrow B)^{db^+}$.Ctid = ϕ，但由引理 1 却得到 $\sigma(A \rightarrow B, DB') = 1$

这是不可能的。可知引理 1 是不正确的。文[4]在证明的不严密上还体现在其完备性定理的证明上。给出其定理如下：

定理 1[4] IMSP 算法恰好发现当前数据库中所有频繁序列。

由前面的讨论可知，算法 IMSP 是不可能恰好发现当前数据库中所有频繁序列的。如利用文[6]的算法 ISP 就可得到比算法 IMSP 更多、更全的频繁集。那么，文[4]的证明错在什么地方？问题主要出在 $F^{DB} \subseteq$ Answer 的证明上。现抄录文[4]相关证明如下：

"若 $X \in Old_k$，则 $X \in F_k^{DB}$，由引理 1 知 X^{DB}.sup 必可由 Find-Old 第(4)、(5)步求出。…，故 X 必被算法 Find_Old 发现；若 $X \in New_k$，则由 Apriori_gen 算法的正确性知，$X \in C_k^{DB}$，…，故 X 必被算法 Find_New 发现。"

从数学的角度上看，以上的证明是不通的。因为文中"必可"、"必被"的地方，正是其证明最为微妙、最需要说明之处。忽略掉这部分的推理，其证明就毫无意义了。笔者置疑的也正是这些地方。

3 结 论

已知，磁盘空间与 CPU 或人力时间相比，是要便宜得多。实际上，磁盘空间一开始就比较便宜，而且每年还不断下降。随着科技的发展，相信磁盘的容量将不断增大，而价格将会进一步下调。另外，从工程的角度

上看，任何能提高挖掘速度的尝试都是有益的。这样，综合考虑，以牺牲磁盘为代价来记录前面的挖掘数据表 Ctid，以加速本次挖掘速度。文[4]的思路无疑是非常正确的。但由于其考虑上的不周，算法 IMSP 在时间复杂度、挖掘规则的完备性上都有着明显的不足，同时还应看到，利用其算法还可能得到错误的结果。综上所述，笔者认为该算法是不完善的。

参考文献：

[1] AGRAWAL R, SRIKANT R. Mining sequential patterns[A]. In: Proc International Conference on Data Engineering, Taipei, Taiwan, 1995. 3 - 14.

[2] SRIKANT R, AGRAWAL R. Mining sequential patterns: generalizations and performance improvements. In: Proc international Conference on Extending Database Technology, Avignon France, 1996. 3 - 17.

[3] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules. In: proceedings of the 20th International conference on Very Large Database [C], Santiago, Chile, September 1994. 487 - 499.

[4] 周斌，吴泉源. 序列模式挖掘的一种渐进算法[J]. 计算机学报，1999，22(8)：882 - 887.

[5] CHEUNG D W, HAN J W. Maintenance of discovered association rules in large databases: an incremental updating technique[A]. In: Proc International Conference on Data Engineering[C], New Orleans, Lousiana, 1996. 106 - 114.

[6] 陈金玉，樊兴华，曹长修. 序列模式的一种挖掘算法[J]. 重庆大学学报(自然科学版)，2001，24(1)：92 - 94.

Analysis of the Algorithm of Incremental Mining of Sequential Patterns

CHEN Jin-yu, CAO Chang-xiu

(College of Automation, Chongqing University, Chongqing 400044, China)

Abstract: The author analyzes an algorithm for Incremental Mining of Sequential Patterns (IMSP), which is presented in a paper to speed up the current mining process by using the previous results when the underlying database has minor changes. Some improper results of in the paper are pointed out, and counterexamples are presented respectively.

Key words: sequential pattern; algorithms IMSP; frequent sets

(责任编辑　吕赛英)

·下期论文摘要预告·

智能型自动驾驶系统的多源信息融合算法

易正俊[1]，黄翰敏[2]，黄席樾[2]，廖传锦[2]

(重庆大学 数理学院，重庆　400044；2. 重庆大学 自动化学院，重庆　400044)

摘　要：利用模糊积分方法融合高速公路上智能自动驾驶的多源信息，确定汽车应采用的安全运行模式，使之经过专家系统的知识推理，调整汽车运行中的可控参数值，为建立智能型自动驾驶系统奠定了基础，并为具体实现智能型自动驾驶系统创造了重要条件，进而避免交通事故发生，提高道路的通行能力。