高技术通讯 2010 年 第 20 卷 第 2 期:133 ~ 137

doi:10.3772/j.issn.1002-0470.2010.02.005

一种高效的基于位图序列模式挖掘算法①

张长海②* 胡孔法③*** 陈 * 宋爱波**

(*扬州大学信息工程学院 扬州 225009)

(**东南大学计算机科学与工程学院 南京 210096)

摘 要 为实现在大型事务数据库中挖掘有价值的序列数据,提出了一种基于位图的高效的序列模式挖掘算法(SMBR)。SMBR 算法采用位图表示数据库的方法,提出一种简化的位图表示结构。该算法首先由序列扩展和项扩展产生候选序列,然后通过原序列位图和被扩展项位图位置快速运算生成频繁序列。实验表明,应用于大型事务数据库,该方法不仅能有效地提高挖掘效率,而且挖掘处理过程中产生的临时数据所需的内存大大降低,能够高效地挖掘序列模式。

关键词 数据挖掘,序列模式,位图

0 引 言

在大型事务数据库中寻找序列模式是数据挖掘的重要课题,对大型事务数据库中有价值序列数据的挖掘有着广泛的应用前景,例如可用于顾客购买行为分析、网络访问模式分析、科学实验分析、疾病治疗早期诊断、自然灾害预测、DNA 序列模式分析等。这类问题首先由 Agrawal 和 Srikant 两人提出,他们总结了序列模式的定义,提出了一种基于 Apriori 的改进算法——泛化序列模式(generalized sequential patterns,GSP)算法[1]。后来,研究人员提出了挖掘频繁序列片段[2]的问题及多种挖掘算法,如基于规则表达式约束[3]的挖掘,基于垂直格式存储的序列模式挖掘算法——SPADE 算法[4],基于投影的模式增长的 Freespan 算法[5],基于 Freespan 的改进算法 Prefixspan[6],分布式序列模式挖掘算法 FMGSP[7],以解决分布式环境下序列模式挖掘问题。

2002 年 Ayres 提出了基于位图的序列模式挖掘算法 SPAM[8],该方法利用垂直位图表示数据库,通过序列扩展和项扩展产生候选序列。近来宋世杰等提出了 HVSM[9]算法,该算法采用垂直位图法表示数据库,先横向扩展项集,将挖掘出的所有大项集组成一大序列项集,再纵向扩展序列,将每个一大序列项集作为"集成块",在挖掘 k 大序列时重用大项集,并以兄弟节点为种子生成候选大序列,利用 1st—TID 对支持度进行快速计数,高效地生成频繁序列模式。SPAM 算法和 HVSM 算法通过位图计数技术在挖掘时间上有显著优势,但是这种优势是建立在有足够大的可用内存的基础上的。为此,本文提出了一种改进的基于位图的序列模式挖掘算法(sequential pattern mining based on bitmap representation,SMBR),该算法提高了挖掘效率,降低了时间复杂度,同时挖掘处理过程中产生的临时数据所需的内存大大降低,能够有效地挖掘序列模式。

1 基本知识和概念

设 $I = \{i_1, i_2, \cdots, i_n\}$ 是一个项目集合,项目集或者项集(items)就是各种项目组成的集合,即 I 的所有子集。一个序列就是若干项集的有序列表,一个序列 S 可表示为 $< s_1, s_2, \cdots, s_n >$,其中 s_j 为项集,也称作 S 的元素。元素由不同的项组成,可表示为(x_1, x_2, …, x_n)。当元素只包含 1 项时,一般省去括号,如(x_2)一般表示为 x_2。元素之间是有顺序的,但元素内的项是无序的,一般定义为词典序。序列包含项的个数称为序列的长度,长度为 L 的序列记为 L-序列。序列数据库就是元组(tuples)

① 国家自然科学基金(60773103,60673060),江苏省自然科学基金(BK2009697,BK2008206),江苏省教育厅自然科学基金(08KJB520012),江苏省"六大人才高峰"基金和江苏省"青蓝工程"基金资助项目。

② 男,1980 年生,硕士;研究方向:数据库,数据仓库与数据挖掘;E-mail: zhangchanghai8088@163.com

③ 通讯作者,E-mail:kfhu05@126.com

(收稿日期:2009-02-16)

$<sid, s>$的集合，其中 s 是序列，sid 是该序列的序列号，元组的个数称为序列数据库的大小，记作：*SDB*。对于序列 $s_1 = <a_1, a_2, \cdots, a_n>$ 和 $s_2 = <b_1, b_2, \cdots, b_m>$，当且仅当 $1 \leqslant j_1 < j_2 < \cdots < j_n \leqslant m$，$a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \cdots, a_n \subseteq b_{jn}$ 时，称序列 s_2 为序列 s_1 的超序列，序列 s_1 为序列 s_2 的子序列，又称序列 s_2 包含序列 s_1，记作：$s_1 \subseteq s_2$。

与 SPAM 算法不同，SMBR 算法对序列模式的某些术语进行重新定义。

定义 1 序列数据库 D 是元组 $<sid, S>$ 的集合，sid 为序列标识号。如果序列 T 是 S 的子序列（即 $T(S)$），称元组 $<sid, S>$ 包含序列 T，则序列 T 在序列数据库 D 中的支持数是数据库中包含 T 的元组数，即 $support(T) = |\{<sid, S> | <sid, S> (D (T(S\}|$，记作 $support(T)$。给定一个最小支持度阈值 *minsup*（对应的最小支持数为 *mincount*），如果序列 s 的支持度不小于 *minsup*（即序列 s 的支持数不小于 *mincount*），则称序列 s 为频繁序列模式。

定义 2 序列扩展（sequence extension，SE）是在原序列 s 的末尾添加一个项，新添加项作为序列的一个新元素，即 $s \diamondsuit_s \alpha = <a_1, a_2, \cdots, a_n, \alpha>$。项扩展（item extension，IE）是在原序列 s 的末尾添加一个项，该新添加项和原序列 s 的最后一个元素来组成一个新的元素，其中新添加项为新元素的最后一项，即 $s \diamondsuit_i \alpha = <a_1, a_2, \cdots, a_n \cup \alpha>$ 且 $k \in a_n$，$k < \alpha$（表示 k 发生在 α 前面）。

例 1 假设 $s = <a(abc)(cf)d>$，$\alpha = c$，那么 $s \diamondsuit_s \alpha = <a(abc)(cf)dc>$，$s \diamondsuit_i \alpha = <a(abc)(cf)(dc)>$。

性质 1 （Apriori 性质）如果一个序列 s 的支持度小于 *minsup*，也就是不频繁的序列模式，那么该序列 s 的任何超序列都是不频繁的序列。

推理 1 （SE 剪枝）如果存在序列 $s = <a_1, a_2, \cdots, a_n>$ 不满足最小支持度 *minsup*，那么序列 s 进行序列扩展产生新序列 $s \diamondsuit_s \alpha = <a_1, a_2, \cdots, a_n, \alpha>$ 也不频繁。即在产生候选序列时，不必在节点序列 s 上进行序列扩展。

推理 2 （IE 剪枝）如果存在序列 $s = <a_1, a_2, \cdots, a_n>$ 不满足最小支持度 *minsup*，那么序列 s 进行项扩展产生新序列 $s \diamondsuit_i \alpha = <a_1, a_2, \cdots, a_n \cup \alpha>$ 也不频繁。即在产生候选序列时，不必在节点序列 s 上进行项扩展。

2 位图序列挖掘算法

2.1 位图结构

为了快速计数，提出一种简化位图结构来表示序列数据库。如将序列数据库（表 1）转化为频繁序列首位置表（the first _ position table，FPT）（表 2）和重复项 0 1 位图表（the bit(01) table，BT）（表 3）。

表 1 序列数据库

ID 号	序列
1	<b(cd)de>
2	<cbd(acd)>
3	<aac>
4	<(acd)de>

表 2 频繁项首位置

项	首位置
a	0 4 1 1
b	1 2 0 0
c	2 1 3 1
d	2 3 0 1
e	4 0 0 3

表 3 重复项 0 1 位图

ID 号	项	重复项
1	d	0 1 1 0
2	c	1 0 0 1
2	d	0 0 1 1
3	a	1 1 0
4	d	1 1 0

频繁序列首位置表构造方法：以序列中的元素为计数单位，序列首次出现在序列中第几个元素则标示几；该序列没有出现，则在该序列位置标示 0。例如：项 a 在 ID = 1 序列中没有出现则该位为 0，在 ID = 2 序列中首次出现在第 4 元素（acd）中，则该位为 4，同理，在 ID = 3 和 ID = 4 位分别为 1，1，所以项 a 在所有序列中的首位置为 0411。简称：$FP_a(s_1) = 0, FP_a(s_2) = 4, FP_a(s_3) = 1, FP_a(s_4) = 1, FP_a() = 0411$。$FP_{bd}(s_1) = 2, FP_{bd}(s_2) = 3, FP_{bd}(s_3) = 0, FP_{bd}(s_4) = 0, FP_{bd}() = 2300$。

重复项 0 1 位图表构造方法：以序列中的元素为计数单位，将同一序列中重复项以 0 1 位图形式标示，若重复项出现在某元素位置则标示 1，否则标示 0。例如：项 d 在 ID = 1 序列中第 2、3 元素位置出现，则该重复项位图为 0110。简称：$Bit_d(s_1) = 0110$。

2.2 SMBR 算法描述

算法 1 生成并简化 FPT 表和 BT 表算法

1： For each sequence s_i

2: For each element $e_i \in s_i$
3: For each item k$\in e_i$
4: $FP_k(s_i)=j$;
5: $Bit_k(s_i)=1$;/* if k not in s_i, then $FP_k(s_i)=0$; $Bit_k(s_i)=0$. */
6: For each item k in FPT
7: If count(k) < *minsup*// count(k): $FP_k()$ 中非 0 值的个数
8: Remove the tuple [k, $FP_k()$] from FPT;
9: Remove $Bit_k()$from BT;
10: Else
11: $L_1 = L_1 \cup \{k\}$;
12: If $count_{s_i}(k) = <1/*\ count_{s_i}(k)$:k 在序列 s_i 重复出现的次数*/
13: Remove $Bit_k(s_i)$ corresponding s_i in BT。

算法 2 生成所有频繁序列算法

输入:FPT 表和 BT 表,用户定义最小支持度 *minsup*

输出:所有频繁序列模式 $S=\{L_1, L_2, \cdots, L_k\}$

1: For each frequent sequence L_p
2: Generate C_{p+1} by SE() and IE();
3: For each frequent sequence L_p
4: For each item T in L_p
5: Call $FP_{Lp}()$ and $FP_T()$; /*判定通过序列扩展形成新序列是否频繁*/
6: IF $FP_{Lp}(s_i) < FP_T(s_i)$ /* $FP_{Lp}(s_i)$和 $FP_T(s_i)$比较前提均非零值*/
7: $FP_{Cp+1}(s_i) = FP_T(s_i)$;
8: Continue to compare $FP_{Lp}(s_{i+1})$ and $FP_T(s_{i+1})$;
9: Else
10: Call $Bit_T(s_i)$, /*移动操作直到出现第二个值"1"*/
11: If $FP_{Lp}(s_i) < \{FP_T(s_i)+h\}$
12: $FP_{Cp+1}(s_i) = \{FP_T(s_i)+h\}$;
13: Continue to compare $FP_{Lp}(s_{i+1})$ and $FP_T(s_{i+1})$;
14: Else
15: Continue to perform the left _ shift operation until the last bit; /*继续移动操作直到最后一位*/
16: If $FP_{Lp}(s_i) > \{FP_T(s_i)+h\}$
17: $FP_{Cp+1}(s_i)=0$;
18: IF $FP_{Lp}(s_i) = FP_T(s_i)$ /*判定通过项扩展形成新序列是否频繁*/
19: $FP_{Cp+1}(s_i) = FP_T(s_i)$; /* $FP_{Lp}(s_i)$和 $FP_T(s_i)$比较前提均非零值*/
20: Continue to compare $FP_{Lp}(s_{i+1})$ and $FP_T(s_{i+1})$;
21: Else
22: Call $Bit_T(s_i)$, Perform the left _ shift operation until the second value "1" apprear;
23: If $FP_{Lp}(s_i) = \{FP_T(s_i)+h\}$
24: $FP_{Cp+1}(s_i) = \{FP_T(s_i)+h\}$;
25: Continue to compare $FP_{Lp}(s_{i+1})$ and $FP_T(s_{i+1})$;
26: Else
27: Continue to perform the left _ shift operation until the last bit;
28: If $FP_{Lp}(s_i) = !\ \{FP_T(s_i)+h\}$
29: $FP_{Cp+1}(s_i)=0$;
30: If count(C_{p+1}) > = *mincount*/*统计计数超过 *mincount* 则为频繁序列*/
31: $L_{p+1} = L_{p+1} \cup \{C_{p+1}\}$;
32: $S = S \cup \{L_1, L_2, \cdots, L_p...\}$。

2.3 算法流程

SMBR 算法以表 1 序列数据库为例描述挖掘序列模式全过程,假设 *mincount* = 2。

第一步:扫描表 2,统计每项的 count(x),count(a) = 3 > *mincount*,故序列 a 频繁。同理,$L_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$。

第二步:由 L_1 产生长度为 2 的候选序列 $C_2 = \{\{aa\}\{ab\}\{ac\}\{ad\}\{ae\}\{(aa)\}\{(ab)\}\{(ac)\}\{(ad)\}\{(ae)\}\ \{ba\}\{bb\}\{bc\}\{bd\}\{be\}\{(bb)\}\{(bc)\}\{(bd)\}\{(be)\}\{ca\}\{cb\}\{cc\}\{cd\}\{ce\}\{(cd)\}\{(ce)\}\{da\}\{db\}\{dc\}\{dd\}\{de\}\{(de)\}\{ea\}\{eb\}\{ec\}\{ed\}\{ed\}\{ee\}\}$。

验证{cd}是否频繁:

1) 调用 $FP_c()=2131$, $FP_d()=2301$。$FP_c(1) = FP_d(1)$,调用 $Bit_d(s_1)=0110$。移动 1 个单位第二次出现值"1",则 $FP_c(1) < \{FP_d(1)+1\}$,$FP_{cd}(1)=3$。

2) $FP_c(2) < FP_d(2)$, 则 $FP_{cd}(2)=3$。

3) $FP_d(3)=0$,则跳过。

4) $FP_c(4) = FP_d(4)$,调用 $Bit_d(s_4)=110$。移动 1 个单位第二次出现值"1",则 $FP_c(4) < \{FP_d(4)+1\}$,$FP_{cd}(4)=2$。$FP_{cd}()=3302$,count(cd) = 3 > *mincount*,

故序列{cd}频繁。

验证{(cd)}是否频繁：

1) $FP_c(1) = FP_d(1)$，$FP_{(cd)}(1) = 2$。

2) $FP_c(2) < FP_d(2)$，$FP_{(cd)}(2) = 0$。

3) $FP_d(3) = 0$，则跳过。

4) $FP_c(4) = FP_d(4)$，$FP_{(cd)}(4) = 1$。$FP_{(cd)}() = 2001$，count({(cd)}) = 2 ≥ *mincount*，故序列{(cd)}频繁。

同理，$L_2 = \{\{(ac)\}\ \{(ad)\}\ \{bc\}\{bd\}\{cd\}\{ce\}\{(cd)\}\ \{dd\}\{de\}\}$

第三步：由 L_2 产生长度为 3 的候选序列 $C_3 = \{\{(acd)\}\{bcd\}\ \{b(cd)\}\{bdd\}\ \{cdd\}\ \{cde\}\ \{(cd)d\}\ \{(cd)e\}\ \{ddd\}\{dde\}\}$

验证{cde}是否频繁：

1) 调用 $FP_{cd}() = 3302$，$FP_c() = 4003$。$FP_{cd}(1) < FP_e(1)$，$FP_{cde}(1) = 4$。

2) $FP_e(2) = 0$，则跳过。

3) $FP_e(3) = 0$，则跳过。

4) $FP_{cd}(4) < FP_c(4)$，$FP_{cde}(4) = 3$。$FP_{cde}() = 4003$。count(cde) = 2 ≥ *mincount*，故序列{cde}频繁。

同理，$L_3 = \{\{(acd)\}\ \{b(cd)\}\{bdd\}\{cde\}\ \{(cd)d\}\ \{(cd)e\}\ \{dde\}\}$。

第四步：由 L_4 产生长度为 4 的候选序列 $C_4 = \{\{(cd)de\}\}$。同理验证序列 count({(cd)de}) = 2 ≥ *mincount*，故 $L_4 = \{\{(cd)de\}\}$，没有长度为 5 候选序列生成，结束。

3 实验结果分析

为测试 SMBR 算法的有效性，在 Windows XP、Pentium IV 2.8GHz、内存 512MB 实验环境下利用 Visual C++ 实现对算法的测试，实验数据为合成数据（http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html），生成数据的主要参数依据：|D|：总序列交易数；|C|：序列中交易的平均长度；|T|：交易中项的平均长度；|S|：最大序列平均长度；|I|：最大序列中交易的平均长度。本文将 SMBR 算法与 SPAM 算法、BitSPADE 算法[10]在算法执行时间和内存使用情况两方面进行比较：第 1 组实验是对数据集（D3KC6T5S5I5），分别在 0.05、0.1、0.15、0.2、0.25 等 5 个不同最小支持度 *minSup* 下的算法执行时间，其实验结果如图 1 所示。第 2 组实验加大数据库事务数，同时提高最小支持度 *minSup*，对数据集（D10KC15T10S10I10），分别在 0.65、0.7、0.75、0.8、0.85 等 5 个不同最小支持度 *minSup* 下的算法执行时间，其实验结果如图 2 所示。第 3 组实验是对数据集（D8KC10T10S10I8）分别在 0.05、0.1、0.15、0.2、0.25 等 5 个不同最小支持度 *minSup* 下的内存使用情况，其实验结果如图 3 所示。

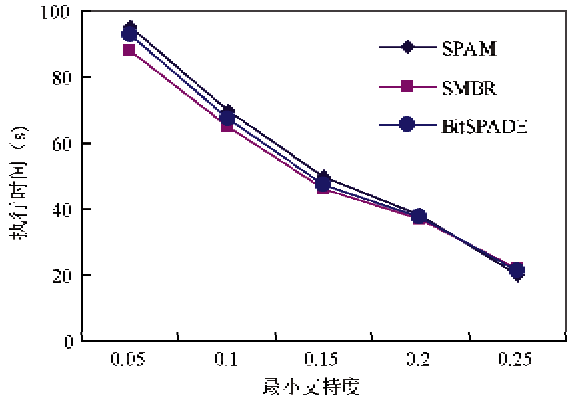

图 1 最小支持度变化时执行时间

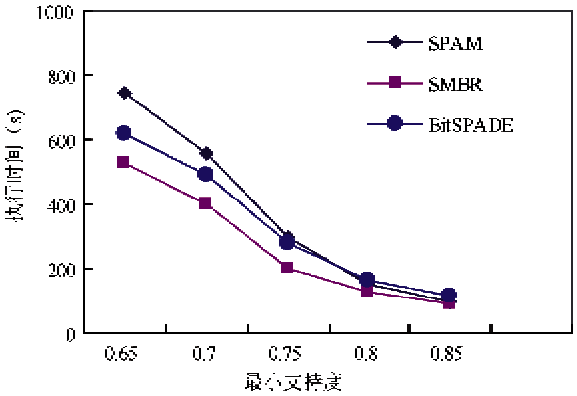

图 2 最小支持度变化时执行时间

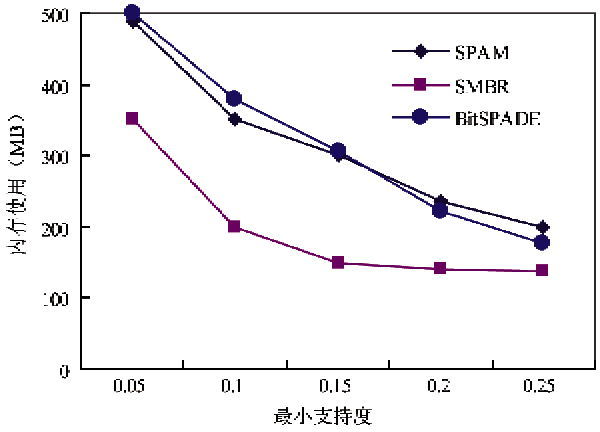

图 3 最小支持度变化时内存使用情况

由图 1 可以得出，支持度小于 0.2 时，SMBR 算法执行时间小于 SPAM 算法和算法 BitSPADE，支持度大于 0.2 时，3 种算法执行时间相当。对于小数

据集(D3KC6T5S5I5),支持度相对较大时,满足 *minSup*≥0.2 条件的序列锐减,*SMBR* 算法无法发挥简化位图结构的优势。由图 2 可以得出,当数据集增大时,*SMBR* 算法执行时间和其他两种算法执行时间拉开差距,*SMBR* 算法充分发挥简化位图优势,利用序列首位置和被扩展项首位置快速运算,快速统计计数,减低时间复杂度。由图可见对大数据集(*D10KC15T10S10I10*)当 *minSup*≤0.7 时,SMBR 算法执行时间减少近 SPAM 算法执行时间的 1/3,SMBR 算法执行时间减少近 BitSPADE 算法执行时间的 1/2。由图 3 可以得出,SMBR 算法内存占用小于 SPAM 算法和 BitSPADE 算法,SMBR 算法在挖掘处理过程中,仅需保存上层频繁序列首位置图和重复项(01)位图,且操作过程中关于重复项(01)位图对同一序列中出现一次的项进行移除,降低了内存使用量。

4 结 论

本文提出用一种改进的基于位图的序列模式挖掘算法——SMBR 算法,来解决大型事务数据库中序列模式挖掘问题。该算法采用位图表示数据库的方法来表示事务数据库的简化位图结构。通过该位图结构,对频繁序列首位置和被扩展项首位置快速运算,获得生成候选序列的首位置图,最终快速统计计数,生成所有频繁序列。实验结果和算法分析表明,在大型事务数据库中,该方法不仅能有效地提高了挖掘效率,而且挖掘处理过程中产生的临时数据所需的内存大大降低,能够高效地挖掘序列模式。今后我们将考虑对该算法的优化及更新的研究。

参考文献

[1] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Proceedings of the 5th International Conference on Extending Database Technology. Heidelberg, Germany: Springer, 1996. 3-17

[2] Mannila H, Toivonen H, Verkamo A I. Discovery of frequent episodes in sequences. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, USA: ACM Press, 1995. 10-21

[3] Garofalakis M, Rastogi R, Shim K. Spirit: Sequential pattern mining with regular expression constraints. In: Proceedings of the 25th International Conference on Very Large Data Bases. San Francisco, USA: Morgan Kaufmann, 1999. 223-234

[4] Zaki M. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 2001, 41(2): 31-60

[5] Han J, Pei J. Freespan: frequent pattern - projected sequential pattern mining. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, USA: ACM Press, 2000. 355-359

[6] Pei J, Han J, Mortazavi-Asl B, et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of 2001 International Conference on Data Engineering. Heidelberg, Germany: Springer, 2001. 215-224

[7] 胡孔法, 张长海, 陈 等. 分布式序列模式挖掘技术研究. 计算机集成制造系统, 2007,13(11):2229-2235

[8] Ayres J, Flannick J, Gehrke J, et al. Sequential pattern mining using a bitmap representation. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, USA: ACM Press, 2002. 168-173

[9] 宋世杰, 胡华平, 嘉伟等. 一种基于大项集重用的序列模式挖掘算法. 计算机研究与发展, 2006,43(1): 68-74

[10] Aseervatham S, Osmani A. BitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation. In: Proceedings of the 6th International Conference on Data Mining. San Francisco, USA: Morgan Kaufmann, 2006. 792-797

An improved sequential pattern mining algorithm based on bitmaps

Zhang Changhai*, Hu Kongfa*,**, Chen Ling*, Song Aibo**

(* College of Information Engineering, Yangzhou University, Yangzhou 225009)

(** School of Computer Science & Engineering, Southeast University, Nanjing 210096)

Abstract

For mining valuable sequence data in large transaction databases, the paper proposes an algorithm for sequential pattern mining based on bitmap representation (SMBR). The SMBR algorithm uses bitmaps to represent databases, and presents a simplified bitmap structure. First the algorithm generates candidate sequences by sequence extension (SE) and item extension (IE), and then obtains all frequent sequences by comparing the original bitmap and the extended item bitmap. The experiments show that when using the algorithm in large transaction databases the required memory size for storing temporal data during mining process is greatly decreased, and all sequential patterns can be efficiently mined.

Key word: data mining, sequential patterns, bitmap