基于 Bitmap 的序列模式挖掘的改进算法

王红侠,胡学钢

(合肥工业大学 计算机与信息学院,安徽 合肥 230009)

摘 要:结合 BBSP,提出了一种称做最终位置归纳序列模式挖掘(LPI-SPM)的新算法,该算法可以有效地从大型数据库中获取所有的频繁序列模式。该策略与以前工作的不同点在于:当判断一个序列是否是模式时,通过扫描数据库创建 S-矩阵来实现(PrefixSpan)或者通过对候选项进行交运算(SPADE)或并运算(BBSP)统计其数量来实现。相反,在基于下列事实的基础上 LPI-SPN 会很容易实施这一过程,即若一个项的最终位置小于当前前缀位置,在相同的顾客序列中,该项就不会出现在当前前缀的后面。LPI-SPM 在序列挖掘过程中可以大大缩减搜索空间,而且挖掘序列模式的效力可观。实验结果表明,在各种数据集合中 LPI-SPM 胜过 BBSP 三倍。

关键词:KDD;位图;序列模式

中图分类号:TP18;TP301.6 **文献标识码**:A **文章编号**:1673-629X(2007)08-0084-04

An Improved Algorithm for Mining Sequential Pattern Based on Bitmap

WANG Hong-xia, HU Xue-gang

(Computer & Information Technology Institute, Hefei University of Technology, Hefei 230009, China)

Abstract: In this paper, by combining BBSP(bitmap based sequential patterns), propose a new algorithm called Last Position Induction Sequential Pattern Mining (LPI-SPM), which can efficiently get all the frequent sequential patterns from a large database. The main difference between our strategy and the previous works is that when judging whether a sequence is a pattern or not, they use S-Matrix by scanning projected database (PrefixSpan) or count the number by joining (SPADE) or ANDing with the candidate item (BBSP). In contrast, LPI-SPM can easily implement this process based on the following fact -if an item's last position is smaller than the current prefix position, the item can not appear behind the current prefix in the same customer sequence. LPI-SPM could largely reduce the search space during mining process and is considerable effectiveness in mining sequential pattern. Our experimental results show that LPI-SPM outperforms BBSP up to three times on all kinds of dataset.

Key words: KDD; bitmap; sequential patterns

0 引 言

Rakesh Agrawal 等[1,2]对超市数据进行分析时首先提出了序列模式(sequential patterns),发现这一 KDD 分支。经典的序列模式发现算法包括:R. Agrawal 等人提出 AprioriAll 算法[1]和 GSP 算法[2];PSP[3]算法是 GSP 算法的改进,由 Han 等人提出,称为基于序列模式增长(sequential patterns growth)方法,包括 FreeSpan[4],PrefixSpan[3]算法。但这些算法都需要多次扫描数据库。

多次扫描数据库需要花费大量的时间,降低了算法的执行效率。此外支持度的频繁计算也成为影响算法执行效率的关键。针对这两个问题,孙莹[5]等提出了一种基于 bitmap 的序列发现算法 BBSP(bitmap based sequential patterns)。把整个交易数据库中的记录映射到一张 bitmap 表中,并采用逐层的序列模式发现方法。由于存储在内存中的 bitmap 保留了原交易数据库的所有信息,所以不需要再反复扫描数据库,提高了算法的时间性能。同时,bitmap 技术可以将对数据库数据的复杂操作转化为对 bitmap 的按位的简单的 AND,OR 和 NOT 操作[6~8],大大提高了算法的执行效率。但是,大量的与运算和或运算依然花费系统的开支,文中提出一种改进算法,通过最终位置归纳,可大大缩减搜索空间,更进一步提高效率。

收稿日期:2006-10-12

作者简介:王红侠(1969-),女,安徽宿州人,讲师,硕士研究生,研究方向为 KDD;胡学钢,教授,硕士生导师,研究方向为 KDD、人工智能、算法设计等。

1 基本概念

1.1 序列模式

在交易数据库 DB 中,每个商品称为一个数据项

(item,以下简称项),非空集合 $I = \{i_1, i_2, \cdots, i_m\}$ 称为数据项集(itemset,以下简称项集),其中每个 $i_k(1 \leqslant k \leqslant m)$ 是一个项。长度为 k 的项集称为 k 项集。DB 中每个交易(transaction) 由顾客号、交易时间和该交易所购买的项集构成。不考虑顾客购买项的数量并假定同一时间一位顾客只进行一次交易。

定义 1:序列(sequence) 是项集的有序表,记作 $< s_1, s_2, \cdots, s_n >$,其中 $s_k(1 \leqslant k \leqslant n)$ 是项集。

定义 2:给定两个序列 $A = < a_1, a_2, \cdots, a_m >$ 和 $B = < b_1, b_2, \cdots, b_n >$,如果存在一组整数 $i_1 < i_2 < \cdots < i_m$ 使得 $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \cdots, a_m \subseteq b_{im}$,则称序列 A 被序列 B 包含。不包含在任何其它序列中的序列称为最大序列(maximal sequence)。

如果把顾客的所有交易按交易时间排列,将得到一个顾客序列,记作 $<$ itemset(T_1), itemset(T_2),$\cdots$, itemset(T_n) $>$,这里 $T_i(1 \leqslant i \leqslant n)$ 是某顾客的第 i 次交易时间,itemset(T_i) 为该次交易中顾客购买的项集。由全部顾客序列组成的数据库称为顾客序列数据库,记作 SDB。通常,得到 SDB 需要对原交易数据库 DB 重构。

定义 3:顾客支持序列 s 指序列 s 包含于该顾客序列中。序列 s 的支持度指顾客序列数据库 SDB 中支持 s 的顾客数(也称 s 的支持数) 与 SDB 中顾客总数量之比。大于最小支持度(由用户指定的阈值,记为 S) 的序列称为 SDB 上的频繁序列。

给定交易数据 DB 和用户指定的最小支持度 S,序列模式发现的问题就是发现出 DB 中所有满足 S 的最大序列,每一个这样的最大序列代表了一个序列模式(a sequential pattern) 。

1.2　序列模式发现算法

序列模式发现算法基本上可以分为两大类。

第一类是基于 Apriori 特性的、逐层(level - size)的发现方法,包括 AriorAll 算法和 GSP 算法等,这类方法最先由 R. Agrawal 等人提出。关于频繁序列有以下的重要特性(Apriori 特性):在交易数据库中,如果某一长度为 k 的序列不是频繁的,那么它的任何长度为 $k+1$ 的超序列也不可能是频繁的。此类算法根据这个特性在基于已生成的频繁序列搜寻更长的频繁序列的过程中对待检查的序列集进行有效的修剪。除此之外,此类方法中的大多数采取了一种逐层的、候选序列生成和测试方法(candidate generation - and - test approach)。在算法执行中,需要多趟扫描原序列数据库。

另一类方法由 Han 等人提出,称为基于序列模式增长(sequential patterns growth)方法,包括 FreeSpan, PrefixSpan 算法等。这类方法采取了一种分而治之(divide - and - conquer)的思想,挖掘过程中无需生成候选序列,但仍需多次扫描数据库。

文中采取的就是一种改进算法,该算法是建立在 BBSP 算法基础上的。下面先介绍 BBSP 算法。

2　BBSP 算法

BBSP 算法是由孙莹、胡学钢[5]提出的。这里先讨论如何把原交易数据库表示成为所需的 bitmap 结构。理论上,交易数据库是一张三维表,如表 1 所示,在这里把顾客号(CID)和交易时间(TID)合成一列形成二维结构。对这个二维矩阵按列进行映射,数据库中的每一个项目列(以下简称属性列)用一个由 0/1 组成的 bitmap 表示,其中 1 表示该顾客在该时间购买该项目,0 表示没有购买。例:表 1 映射的二维表如表 2 所示。

表 1　交易数据库

顾客号	交易时间	项集
1	93.06.25	C
1	93.06.30	H
2	93.06.10	A,B
2	93.06.15	C,H
2	93.06.20	D,F,G
2	93.06.25	H
3	93.06.25	C,E,G
4	93.06.25	C
4	93.06.30	D,G,H
4	93.07.25	H
5	93.06.12	H

表 2　Bitmap 映射

ID	A	B	C	D	E	F	G	H
(1,6.25)	0	0	1	0	0	0	0	0
(1,6.30)	0	0	0	0	0	0	0	1
(2,6.10)	1	1	0	0	0	0	0	0
(2,6.15)	0	0	1	0	0	0	0	1
(2,6.20)	0	0	0	1	0	1	1	0
(2,6.25)	0	0	0	0	0	0	0	1
(3,6.25)	0	0	1	0	1	0	1	0
(4,6.25)	0	0	1	0	0	0	0	0
(4,6.30)	0	0	0	1	0	0	1	1
(4,7.25)	0	0	0	0	0	0	0	1
(5,6.12)	0	0	0	0	0	0	0	1

面对大规模的项目集,在数据库中使用 bitmap 技术可能存在存储空间代价太大的问题。对此我们可以采用位图的压缩技术[6]。考虑到在实际应用中,1 位的数量是随着行增长的,而 0 位的数量是随着列增长的。在行不变的情况下,随着列的增长,0 会越来越

多。随着 0 数量的增多,可以使得压缩效果越来越好。

BBSP 算法是基于 bitmap 结构的逐层序列模式发现算法,它只需扫描一次交易数据库。先把数据库映射成一个二维 bitmap 结构表。然后对这张表做 AND 运算,生成频繁 1—序列表;再利用频繁 1—序列表,生成频繁 2—序列表,同时对频繁 1—序列表中不在频繁 2—序列表中出现的属性进行剪枝;由剪枝后的频繁 1— 序列表和频繁 $k-1$— 序列表做 OR 运算,生成频繁 k— 序列表,再对频繁 1— 序列表进行剪枝。一般情况下内存中只需保留一个频繁 1— 序列表,一个频繁 k— 序列表和索引表。同时,由于位操作的简单易行,使得支持度的计算更加方便和快捷(注:这里的频繁 k— 序列表都是指的 bitmap 表)。具体 BBSP 算法参见文献[6]。

3 LPI-SPM

3.1 LPI-SPM 算法介绍

尽管 BBSP 算法有效地计算了候选序列的支持度,但是还是发现在这个支持度计算的过程中还有更多的有效改进空间。

在 BBSP 中,判断一个候选序列是否是模式,它需要做和涉及到的顾客数量一样多的与运算。例如,若特定数据集中有 10000 个顾客,则每一个候选项测试就会花费 10000 个"与运算"的时间。考虑到其实施过程中的递归特性,这项开支太大了。因此怎样避免这些"与运算"是在所难免的步骤。

通过前面描述可知,若对特定顾客给出一个当前位置,就能知道哪些项在当前位置之后,哪些项不是基于它们的最终位置。因此,一个判别候选序列的大胆想法是把当前位置和它的最终位置作一比较。实际上,这和 BBSP 中"与运算"花费的开支同样大。为了避免 "比较操作"或者"与运算",当第一次扫描数据库时可以构建 ITEM_IS_EXIST_TABLE 表。在每一次重复的过程中,只需检查这个表,以获悉一个候选是否在当前位置之后即可。这样,通过避免"与运算"或"比较运算"可以节省大量时间。

表 3 描述的是某一个顾客的 ITEM_IS_EXIST_TABLE 表的一部分。左边一列是位置号,顶行是 ID。在表中,使用位矢量去代表候选项各自位置存在。位置是 1 表明该项存在,否则表明该项不存在。位矢量的大小和候选项的总数量相等。例如,若当前位置是 2,可把其相关位矢量看作 1111,这就意味着所有的候选项可以出现在当前前缀的后面。若当前位置是 4,把位矢量看作 1100,表明只有项 a 和 b 存在于位于当前前缀后的同一顾客序列中。为了累算候选序列的支持度,只需检查这个表,并且把相关项的矢量值加起来,同时避免了"比较运算"、"与运算"或在每个递归步中构建 S 矩阵,这在挖掘过程中就大大提高效率了。请注意,这里只讨论了 S 步过程,在相同策略下,读者可以较容易地把它扩展到 I 步过程。

表 3 ITEM_IS_EXIST_TABLE

Pos \ Item	a	b	c	d
1	1	1	1	1
2	1	1	1	1
3	1	1	1	0
4	1	1	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

3.2 空间优化

BBSP 是在假定整个数据库的矢量都能存放在主存中情况下实现的,然而"空间需要"经常是一个算法的关键。如表 3 所示。可以容易知道,LPI-SPM(即最终位置归纳序列模式挖掘)中使用的主存不超过 BBSP 中的 2 倍,因为对每一次交易来说每个项不论它是否存在 ITEM_IS_EXIST_TABLE 表中都只需一个 bit。经过研究发现,表中只有一部分数据是有用的,而大部分是无用的,例如见图 1,当当前位置小于 3 时,所有项都存在,而当位置大于 4 时,就没有项存在了。所以有用的信息存放在一些关键位置行中。

定义关键位置如下:给定一个位置,若它的相关位矢量的个数比该位置序号小 1(位矢量是 0 的除外)这一位置就叫关键位置。

例如,表 3 中,位置 3 和 4 是关键位置,而其它位置则不是(位置 5 不是关键位置是因为它的位矢量是 0)。我们会发现这些关键位置确实是候选项的最终位置(最后一个除外)。表 3 中的阴影部分描述的是 ITEM_IS_EXIST_TABLE 表的优化,它只存储了 2 个位矢量,而不是表 3 中的全部 8 个。对于长序列数据集而言,这一空间节省策略是非常有效的。第 3 部分会描述完整的实验,通过它可以看到,用于存放 ITEM_IS_EXIST_TABLE 表的存储空间少于 SPAM 中使用中的 10%,当比较 LPI-SPM 和 BBSP 有效性时,该空间大小是可以忽略的。

3.3 实 验

笔者在 Intel 奔腾(R)M,1.6GHz,内存 1G 的,系统用 Windows XP 的 PC 机上完成了实验。数据集采用由文献[1]描述的 IBM 数据产生器合成的数据库。

产生的数据集中不同参数的含义如表 2 所示。到目前为止,当挖掘数据库整个序列模式时,BBSP 是最快的算法。对大规模数据集而言,比 PrefixSpan[3] 和 SPADE[2]都快。所以文中只比较 BBSP 和 LPI－SPM 两种算法。所有方法都已用 Microsoft Visual C＋＋ 6.00 实施过。

其中:D － 顾客数量,用 000S 表示;C － 每个顾客的交易平均数量;T － 每个交易项的平均数量;N － 不同项的数量,用 000S 表示;S － 最大序列平均长度。

因为数据集的大小在算法的性能体现中扮演了一个重要角色,如图 1 所示,首先进行了 BBSP 和 LPI－SPM 的不同容量的比较。这组测试表明 LPI－SPM 在不同容量的数据集中其运行时间胜过 BBSP 大约 2～3 倍。

图 1 中:LPI－SPM1、BBSP1 采用数据集 Dataset D1C10T5S5N1; LPI － SPM2、BBSP2 采用数据集 Dataset D5C15T10S10N1;LPI－SPM3、BBSP3 采用数据集 Dataset D5C20T20S20N1。

在第二组实验中,在性能效果的基础上考虑了用于产生数据集的参数不同的情况(如图 2 所示)。

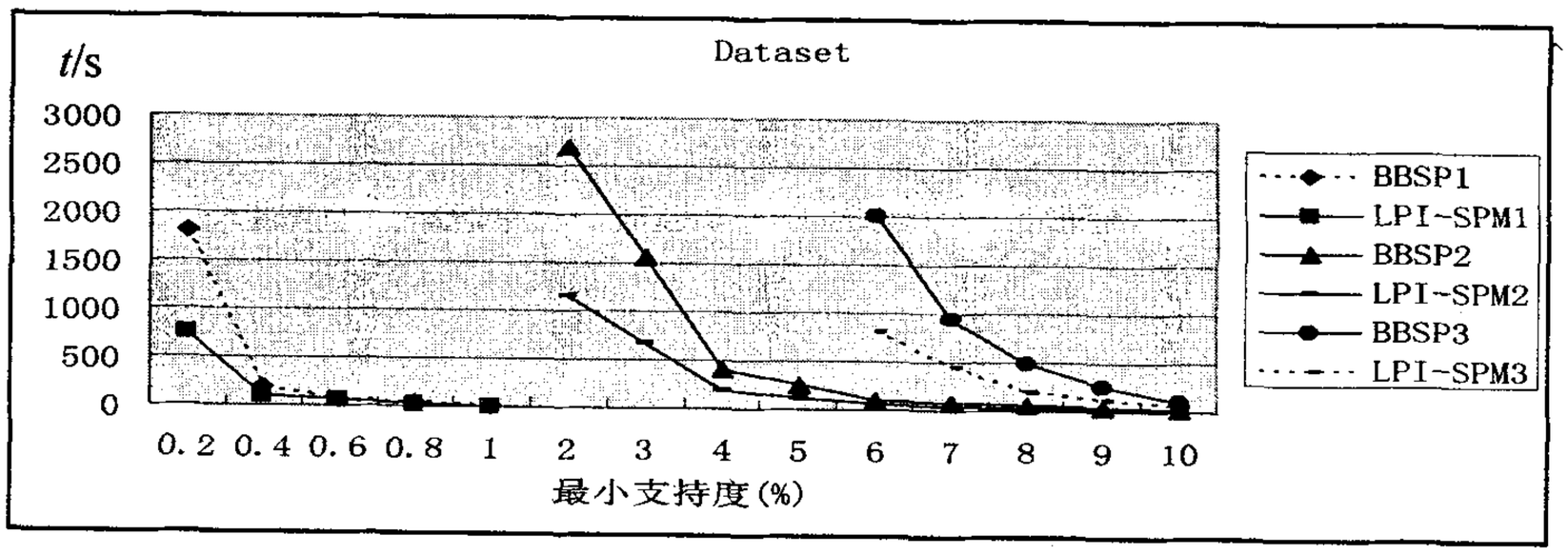

图 1　数据集不同容量的支持度变化

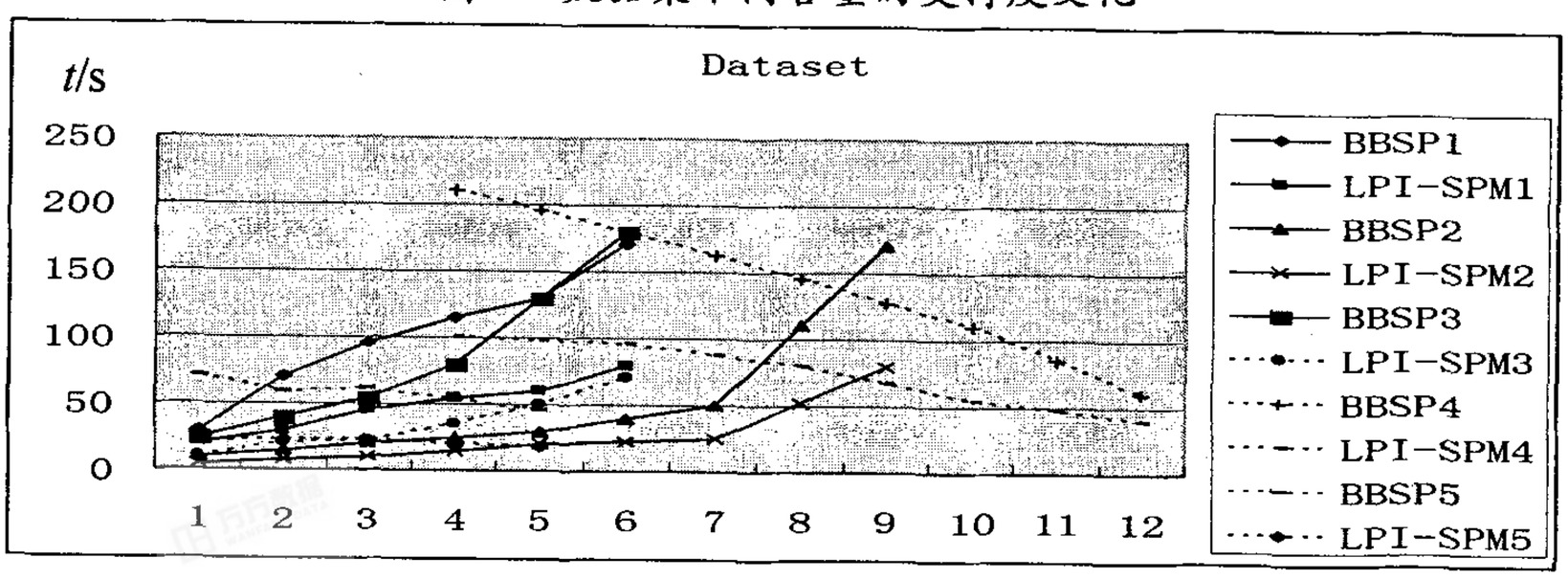

图 2　数据集的参数变化图

图 2 中:LPI－SPM1、BBSP1 曲线横坐标为顾客数量(sup＝10%),数据集为 D?C20T20S20N1;

LPI－SPM2、BBSP2 曲线横坐标为每个顾客的交易平均数(sup＝0.3%),数据集为 D15C?T20S20N1;LPI－SPM3、BBSP3 曲线横坐标为每个交易的项集平均数(sup＝1%),数据集为 D10C15T?S20N1;LPI－SPM4、BBSP4 曲线横坐标为最大序列的平均长度(sup＝7%),数据集为 D10C15T10S?N1;LPI－SPM5、BBSP5 曲线横坐标为数据集中的不同项集数量(sup＝2.2%),数据集为 D8C15T10S20N?。

在图 2 中 LPI－SPM1、BBSP1 曲线表明当改变顾客数量时的结果;LPI－SPM2、BBSP2 表明当改变每一个顾客交易的平均数量变化时的结果;LPI－SPM3、BBSP3 表明当改变每一个交易参数的项的平均数量时的结果;LPI－SPM4、BBSP4 显示的是改变了最大序列的平均长度时的情况;而 LPI－SPM5、BBSP5 中的变量是数据集中不同项的数量。可以看到,不论哪个参数改变,LPI－SPM 总是比 BBSP 快大约 2～3 倍。其主要原因在于 LPI－SPM 对所有数据集而言,性能如此之好,是由于在有效计算时避免了“与运算”和“位图的比较”。这一过程是临界的,因为它在每一个递归步骤中完成了许多次,而且 LPI－SPM 和 BBSP 相比可节省大量时间。

4　总　结

文中给出了新的策略,一个叫 LPI－SPM 的挖掘序列模式的算法。其关键思想在于:若给定一个当前位置,基于项的最终位置,可立刻知道哪一项将出现在当前前缀项之后。LPI－SPM 在每一次重复的支持度统计过程中会避免“与运算”或“比较运算”,可大大提高效率。实际上,对于任一时间序列数据库而言,不同项的最终位置应被加倍关注,因为,在每一个递归步骤中,它们可被当作项是否存在的判断依据。在不同种类数据集中比较 LPI－SPM 和 BBSP 时,我们也有一套完整的实验。结果表明,改进的算法胜过当前最快算法大约 2～3 倍。

未来,想把该策略用到更多方面,如动态数据库(数据流等)。另外,对于大型数据库而言,位图的容量将会很大,应考虑其简化问题。

参考文献:

[1]　Agrawal R,Srikant R. Mining Sequential Patterns[C]//In:

(下转第 91 页)

4.3 基于构件的开发的缺点

虽然 CBSD 有许多优点,但也存在不少缺点,这些缺点给软件开发带来一定的风险。基于构件的开发具有以下缺点[1,5]:

①开发者无法得到构件的源码,也就无法通过修改源码来改变构件的功能,这就意味着分析、调试以及测试构件必须完全以黑盒子的形式进行。

②构件的使用者无法控制构件的升级。

③在构件集成时,所选用的构件之间可能不匹配或者是作为单独应用设计的构件与软件的其他部分不易交互。在有些时候这些问题在开发阶段的后期才会暴露出来。

5 结　论

基于构件的开发方法由于对软件功能性、开发效率、质量、可靠性、可移植性等方面的良好支持而受到越来越多软件开发组织的重视[1,5]。作为一种新的软件开发方法,它在参照准则、质量控制、开发过程等方面使软件工程的研究与实践产生了极大的更新和改进,相对传统开发方法体现出许多新优势,同时随着实施的进展也暴露出许多问题,有待研究和开发人员在更多的工作中进行分析总结、充实和完善 CBSD 的理论体系。

参考文献:

[1] Sommerville I. 高级软件工程[M]. 第 7 版. 北京:机械工业出版社,2004:440 - 459.

[2] Souza D D, Wills A. UML 对象、组件和框架——CATALYSIS 方法[M]. 王　慧,等译. 北京:清华大学出版社,2004:3 - 32.

[3] Szyperski C, Pfister C. WCOP'96 Summary in ECOOP'96 Workshop Reader[M]. Heidelberg: Verlag, 1997:127 - 130.

[4] Bachman F, Bass L, Buhman C, et al. Volume Ⅱ: Technical Concepts of Component - based software Engineering [R/OL]. 2nd Edition. Pittsburgh: Carnegie Mellon Software Engineering Institute, 2000. Chapter 4. http://www.sei.cmu.edu/publications/documents/00. reports/00tr008/00tr008chap04.html.

[5] 杨芙清,梅　宏,李克勤. 软件复用与软件构件技术[J]. 电子学报,1999(2):68 - 75.

[6] Brow A W. 大规模基于构件的软件开发[M]. 赵文耘,张志,等译. 北京:机械工业出版社,2003.

[7] 梅　宏,陈　锋,冯耀东,等. ABC:基于体系结构面向构件的软件开发方法[J]. 软件学报,2003,14:721 - 732.

(上接第 83 页)

[D]. New Mexico: New Mexico Institute of Technology, 1997.

[8] Oikawa S, Rajkumar R. Linux/RK: A Portable Resource Kernel in Linux[C]// In 19th IEEE Real - Time Systems Symposium. Madrid, Spain:[s. n.],1998.

[9] Liu S, Rajkumar R, Lehoczky J. Priority Inheritance Protocols: An Approach to Real - Time Synchronization[J]. IEEE Transaction on Computers,1990,39(9):1175 - 1185.

[10] Lin Kwei - Jay, Wang Yu - Chung. The Design and Implementation of Real - Time Schedulers in RED - Linux[J]. Proceedings of the IEEE, 2003, 91(7):1114 - 1129.

[11] Oikawa S, Rajkumar R. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior[C]// In Proceedings of the IEEE Real - Time Technology and Applications Symposium. Vancouver:[s. n.],1999:111 - 120.

[12] 陈向群,杨芙清. 面向 Aspect 的操作系统研究[J]. 软件学报,2006,17(3):620 - 627.

(上接第 87 页)

Proc. of the 11th Int. Conf. on Data Engineering. Taipei:[s. n.],1995:3 - 14.

[2] Srikant R, Agrawal R. Mining Sequential Patterns: Generalizations and Performance Improvements[C]//In: Proc. of the Fifth Int. Conf. on Extending Database Technology (EDBT). Avignon, France:[s. n.],1996.

[3] Pei J, Han J, Mortazavi - Asl B, et al. PrefixSpan: Mining sequential patterns efficiently by prefix - projected pattern growth[C]//In: Proc. 2001 Int. Conf. Data Engineering (ICDE'01). Heidelberg, Germany:[s. n.],2001:215 - 224.

[4] Han J, Pei J, Mortazavi - Asl B, et al. FreeSpan: Knowledge Discovery and Data Mining (KDD'00). Boston, MA:[s. n.], 2000:355 - 359.

[5] 孙　莹,胡学钢. 基于 Bitmap 的序列模式研究[D]. 合肥:合肥工业大学,2004.

[6] Lin T Y, Hu X. Louie E. A Fast Association Rule Algorithm Based on Bitmap and Granular Computing[C]//The Proceedings of the IEEE International Conference Fuzzy Systems. St Louis, Missouri:[s. n.],2003:678 - 683.

[7] Morzy T, Zakrzewicz M. Group Bitmap Index: A Structure for Association Rules Retrieval[C]//Proc. of the 4th Int'l Conf. on Knowledge Discovery and Data Mining (KDD - 98). US: American Association for Artificial Intelligence, 1998.

[8] Amer - Yahia S, Johnson T. Optimizing Queries on Compressed Bitmaps[C]//Proc. of the 20th VLDB Conf. Cairo, Egypt:[s. n.],2000.