一种基于 FUP 的 TD-FP-Tree 并行快速更新算法

周爱武,王 琰,陈宝楼
(安徽大学 计算机科学与技术学院,安徽 合肥 230039)

摘 要:TD-FP-Growth 是对经典关联规则挖掘算法 FP-Growth 算法的改进,它采用新的数据结构 TD-FP-Tree。人们已经基于 Apriori 和 FP-Growth 算法提出了多种关联规则增量挖掘算法。文中讨论了在基于 TD-FP-Tree 的结构上如何进行增量挖掘,对批量挖掘算法的瓶颈进行分析,指出加快更新速度的策略。文中基于 FUP 思想提出了 TD-FP-Tree 的快速更新算法,重点研究了当有单个项在新增事务加入后由非频繁变为频繁时 TD-FP-Tree 的处理情况。通过将项分类处理降低更新时间,并部分采用并行处理进一步提高效率。实验表明,文中提出的算法不仅可以快速更新 TD-FP-Tree,而且在同基于 FP-Tree 结构的增量挖掘对比中也有更好的表现。

关键词:关联规则;TD-FP-Growth;增量挖掘;FUP;TD-FP-Tree 更新
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2013)04-0091-05
doi:10.3969/j.issn.1673-629X.2013.04.022

A Parallel Fast Update TD-FP-Tree Algorithm Based on FUP

ZHOU Ai-wu, WANG Yan, CHEN Bao-lou
(College of Computer Science & Technology, Anhui University, Hefei 230039, China)

Abstract: TD-FP-Growth is an improvement to the classical algorithm for mining association rules which called FP-Growth, and it uses a new data structure TD-FP-Tree. Many incremental mining algorithm of association rules have been proposed based on the Apriori and FP-Growth. It discusses how to do incremental mining based on the structure of TD-FP-Tree, analyzes the bottleneck of batch mining and points out the strategy of speeding up update rate. It proposes the fast update algorithm of TD-FP-Tree based on the thought of FUP, and puts the focus on researching how to handle the TD-FP-Tree with the situation that a single item becomes frequent by non-frequent when new transactions are added. It processes items classified to reduce the updated execution time, and adopts parallel processing partially to further improve efficiency. Experiments show that the proposed algorithm not only can quickly update TD-FP-Tree, but also has a better performance on the incremental mining compared with the FP-Tree structure.

Key words: association rules; TD-FP-Growth; incremental mining; FUP; TD-FP-Tree updating

0 引 言

关联规则挖掘是数据挖掘中的一个重要分支,意图发现一个事件和其他事件之间的依赖和联系,找出隐藏在数据库中的关联信息,尤其是基于事务数据库的关联规则挖掘对商业决策具有重要价值。关联规则挖掘首先要进行频繁项发现,Han 等提出的 FP-Growth 算法[1]是这一阶段的著名算法,又在 FP-Growth 基础上发展出 TD-FP-Growth 算法[2]。

然而上述算法均要在一个批次中处理所有事务项集,实际应用中,事务数据库总是不断有新事务插入。Cheung 提出 FUP 算法[3]以处理数据库的更新挖掘,它的重要影响在于给出了一种分类处理更新事务集中项的思想[4]。Hong 等人已经将其应用到 FP-Tree 结构的更新上,取得良好结果[5,6]。其他学者也提出一系列改进算法[7~10],或者通过更改 FP-Tree 结构加快增量挖掘速度[11,12]。TD-FP-Growth 在发现频繁项集时相对 FP-Growth 有较大优势,文中认为 TD-FP-Tree 结构的更新研究具有实践意义。文中将 FUP 思想应用到更新 TD-FP-Tree 结构上,并基于目前的并行环境,部分采用并行处理,进一步提高算法执行效率。

收稿日期:2012-07-29;**修回日期**:2012-10-31
基金项目:安徽省教育科研重点项目(KJ2009A57)
作者简介:周爱武(1965-),女,副教授,研究方向为数据库与 web 技术、数据仓库与数据挖掘、信息系统安全;王 琰(1987-),男,硕士研究生,主要研究方向为数据库与 web 技术、数据挖掘。

1 相关工作回顾

FP-Growth 算法见文献[1],采用 FP-Tree 压缩存

储事务数据库中数据。TD－FP－Growth 是对 FP－Growth 的改进,修改 FP-Tree 为 TD-FP-Tree,Tree 中存在 HeaderTable 以及连接所有同项节点的 NodeLink 以遍历整棵 TD-FP-Tree。TD-FP-Tree 同 FP-Tree 的主要区别在于 HeaderTable 中项排序方式,从而避免条件模式基的不断生成。TD-FP-Growth 算法流程见文献[2],分为两个阶段,第一阶段两次扫描事务数据库生成 TD-FP-Tree,第二阶段通过模式增长在 TD-FP-Tree 上挖掘频繁项集。实验表明,相同数据下,TD-FP-Growth 时间效率相对基于 Apriori 的算法提高了一个数量级,相对 FP-Growth 算法发现频繁项时间消耗减少 50% ~70%,高峰内存占用更小[2]。

FUP 本身是基于 Apriori 的增量挖掘算法,主要思想是根据项在原始事务数据库和新增事务集中是否频繁将其分成四类,然后分别处理。令原始事务数据库为 DB,新增事务集为 db,四类分别为:

BothLarge:在 DB 和 db 中均频繁;

OldLarge_NewSmall:在 DB 中频繁,db 中不频繁;

OldSmall_NewLarge:在 DB 中不频繁,db 中频繁;

BothSmall:在 DB 和 db 中均不频繁。

2 并行 TD-FP-Tree 快速更新算法(PFU-TDFP)

2.1 算法思想

FP-Growth 和 TD-FP-Growth 两种算法在 Tree 生成阶段均要两次扫描事务数据库:第一次扫描,获得所有单个项目的计数和排序,得到 HeaderTable 表,表中频繁项降序或升序排列;第二次扫描,按照 HeaderTable 表中项的顺序重新排列各事务中项,剔除非频繁项,生成 Tree 结构。这一阶段涉及 2 次 IO 和重排事务中项操作, HeaderTable 长度越长,重新排序事务时要处理的项的数目就越多,时间耗费越大,在稀疏事务集中生成 Tree 时间甚至可能超过挖掘时间,达到总时间的 80%[12]。因此提高效率的关键在于:

(1)减少扫描事务集数量;

(2)减少扫描次数;

(3)减少重新排序事务中项时依赖项的数目。

FUFP 就是通过将项分类处理,满足了上述策略,减少了更新 FP-Tree 结构的时间。但注意到,新事务集到来时,总的时间耗费由更新 Tree 结构时间和挖掘时间共同决定,而在挖掘阶段有更快的 TD-FP-Growth 算法可用。

若使用 TD-FP-Growth 算法,则更新的 Tree 结构应为 TD-FP-Tree。其中 BothLarge 的项只需更新计数,BothSmall 的项被忽略。OldLarge_NewSmall 的项支持度减少,若变为非频繁,则从 TD-FP-Tree 中删除。从 HeaderTable 开始,删除这一项 NodeLink 上的所有节点,并将被删除节点的父子直接相连。删除完成后,某些节点下可能存在表示同一项的子节点:如 ABC 和 AC 两个路径,A 节点下原有 B、C 两个节点,删除 B,则有两个 C 节点同 A 直接相连。应在删除完成后合并相同子节点。OldSmall_NewLarge 的项支持度增加,可能变频繁而要加入 TD-FP-Tree 结构。此时需再次扫描一遍原始事务数据库,因为它先前是非频繁的,HeaderTable 中没有其计数信息。

在 FUFP 和本算法中,均有如下假设:

假设一 新增事务集与原始事务数据库相比总是较小的,因此从非频繁变为频繁的项也是少数的,多数情况下可能没有,它们的计数也是少量的。

这种假设具有现实背景。实际应用中,如超市销售记录,周期内增加的事务数量 d_i 趋于稳定,总的事务数量 D 不断增加。$D = d_1 + d_2 + \cdots + d_{i-1}$。当一批新的事务集 d_i 到来时,原有的事务数据库总比它大($D = d_1 + d_2 + \cdots + d_{i-1} > d_i$),且比例越来越悬殊($d_i/D = d_j/(d_1 + d_2 + \cdots + d_{i-1})$)。

所以将新出现的频繁项直接添加到 HeaderTable 末端,实际处理方法因排序方式的差异而不同:FP-Tree 中,末端在 HeaderTable 最下方,TD-FP-Tree 中,末端在 HeaderTable 最上方。假设存在频繁项集 ABC,计数依次递减,更新后有新项 D 加入,变为 $ABCD$。在 FP－Tree 上代表一条 $A－B－C$ 路径,将 D 节点直接加在 C 之后即可。但在 TD－FP－Tree 中则代表一条 $C－B－A$ 路径,D 则要插入 C 前,直接同 root 节点相连。需先将 $C－B－A$ 路径删除,再加入 $D－C－B－A$ 路径。可见,虽是同一事务,但在新事务集到来后,表示路径发生了改变。将这种需要修改后再加入 Tree 的事务称作 Rescan_Transaction。所有 Rescan_Transaction 同新增事务集合并,得到事务集 Scan_ Transactions,即将要在 TD－FP－Tree 中更新的全部路径。原 TD－FP－Tree 中旧路径的删除和新路径的加入可并行处理,先将所有的 Scan_Transactions 生成一棵新树,再将其合并到删除完成后的原 TD－FP－Tree。

2.2 算法流程

由于在处理 Scan_Transactions 时采用了并行思想,此算法被称作 PFU-TDFP(Parallel Fast Update TD-FP-Tree)算法,它的具体流程如下:

输入:原始事务数据库 DB,根据 DB 生成的 HeaderTable 和 TD-FP-Tree,最小支持度 min_sup,新增事务集 db。

输出:更新后的 TD-FP-Tree。

Step1:扫描新增事务集 db,获得所有单个项的计数和排序。

Step2:根据项在 DB 和 db 是否频繁,将项分到 BothLarge、OldLarge _ NewSmall、OldSmall _ NewLarge、BothSmall 4 个集合中。

Step3:处理 BothLarge 中的项:

for BothLarge 中的项 I:

(1)计算全局支持度 $S^U(I) = S^D(I) + S^d(I)$。

(2)更新 HeaderTable 中 I 的计数。

(3)将 I 放入 insert_items 中。

Step4:处理 OldLarge_NewSmall 中的项。

Step4-1:for OldLarge_NewSmall 中的项 I:

(1)计算全局支持度 $S^U(I) = S^D(I) + S^d(I)$。

(2)若 $S^U(I)$ <min_sup,则要将 I 从 TD-FP-Tree 中删除。

a. 根据 HeaderTable 中 I 的 NodeLink,删除 NodeLink 上的所有 I 节点,将 I 节点原先的父子节点直接相连。

b. 删除 HeaderTable 中的 I 节点。

(3)若 $S^U(I) \geqslant$min_sup,更新 HeaderTable 中 I 的计数,将 I 放入 insert_items 中。

Step4-2:合并删除后的 Tree:

for HeaderTable 中的项 I:

遍历 I 的 NodeLink,若 NodeLink 上某两个节点有相同的父节点,则这两个节点合并(计数合并,指向共同的父节点)。

Step5:处理 OldSmall_NewLarge 中的项:

for OldSmall_NewLarge 中的项 I:

(1)扫描 DB 一次,得到 $S^D(I)$ 和各自的 Rescan_Transactions(I)(只要事务中含有 OldSmall_NewLarge 中的项就放入)。

(2)若 $S^U(I) = S^D(I) + S^d(I) \geqslant$min_sup,将 I 放入 new_insert_items 中,并将其 Rescan_Transactions(I)合并至 Rescan_Transactions,合并时不得有重复的事务。

Step6:将 Rescan_Transactions 同 db 合并为 Scan_Transactions。

Step7:将 new_insert_items 中的项升序排列,将 insert_items 中的项升序排列,并将 new_insert_items 整体放入 insert_items 的前端,组成新的 insert_items。

Step8:for Scan_Transactions 的每个事务项集:

(1)按 insert_items 中项顺序重新排列,不在其中的项删除。

(2)并行执行以下 2 步:

a. 去除原 TD-FP-Tree 中的一条相同路径:

* 去除处理后事务中属于 new_insert_items 的项(即属于 OldSmall_NewLarge 的项)。

* 按照事务中项的顺序,从 root 节点向叶子节点进行,在原 TD-FP-Tree 中找到这样的一条路径——路径末端无节点或无同其计数相同的子节点,说明原 TD-FP-Tree 中存在表示同一事务的路径。

* 将此条路径上所有节点计数均减去 1,若减为 0,则删除此节点,并连接前后节点为 NodeLink。

b. 以 root 为根节点,生成一个新的路径,最终得到一个 new_TD-FP-Tree。

Step9:将 new_TD-FP-Tree 同经过裁剪后的原 TD-FP-Tree 合并(算法同 OldLarge_NewSmall 中合并步骤),按照 insert_items 中项的顺序维护 HeaderTable 和 NodeLink。

事实上,insert_items 是更新后 HeaderTable 的一个子集,减少了重排序事务中项时依赖项的个数,符合前文提出的策略,因此可加快 TD-FP-Tree 的更新速度。

2.3　示例说明

通过一个示例进一步说明 OldSmall_NewLarge 中项的处理。示例中数据来自文献[5],是一个假设的事务数据库,共有 10 条事务,单个项目数量为 9,新增事务集包含 5 条事务,如表 1 和表 2 所示:

表 1　原始事务数据库

Transaction No.	Items
1	a,b,c,d,e,g,h
2	a,b,f,g
3	b,d,e,f,g
4	a,b,f,h
5	a,b,f,i
6	a,c,d,e,g,h
7	a,b,h,i
8	b,c,d,f,g
9	a,bf,h
10	a,b,g,h

表 2　新增事务集

Transaction No.	Items
1	a,b,d,e,g
2	c,e,i,f
3	a,b,d,h
4	a,b,c,d,g
5	b,c,d,i

设最小支持度为 50%,原始事务数据库的频繁项是 a、b、f、g、h。新增事务集到来,经过 Step1 到 Step5,f、h 由频繁变为非频繁,删除后得到如图 1 所示的 TD-FP-Tree:

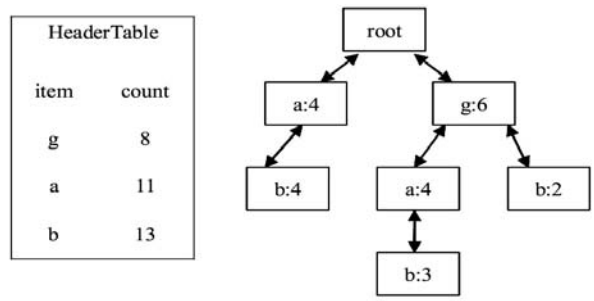

图 1　删除 f、h 节点的 TD-FP-Tree

OldSmall_NewLarge 中的项是 c 和 d,需重新扫描一次原始事务数据库,以获得它们在原始事务数据库中的计数。经过扫描并计算计数总和,发现只有 d 由非频繁变成了频繁,同时得到与 d 有关的事务集合,分

别是{a,b,c,d,g,h,e}、{b,d,e,f,g}、{a,c,d,e,g,h}、{b,c,d,f,g}。这也是最终的 Rescan_Transactions。算法从 Step6 继续。

Step6:将 Rescan_Transactions 同 db 合并为 Scan_Transactions。

Step7:升序排列 new_insert_items 和 insert_items 中的项,并将 new_insert_items 整体放入 insert_items 前端,组成新的 insert_items。得到 insert_items 为{d,g,a,b}。

Step8:根据 insert_items 中项的顺序重排序 Scan_Transactions 事务中项,不在 insert_items 的项删除。处理后的事务为{d,g,a,b}、{d,g,b}、{d,g,a}、{d,g,b}、{d,g,a,b}、{d,a,b}、{d,g,a,b}、{d,b},共 8 个。

按照算法中的处理步骤,对每一个事务,并行地执行下面两步:

(1)去除原 TD-FP-Tree 中表示相同事务的路径。需先去除每个事务中的 d 项,因为它属于 OldSmall_NewLarge 集合。

(2)依照 TD-FP-Tree 的生成算法[2],生成一棵 new_TD-FP-Tree。

最终得到的两棵树如图 2 所示:

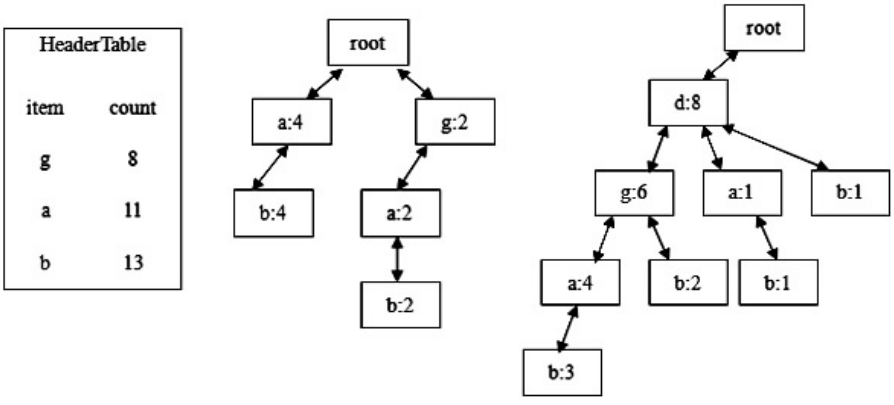

图 2 生成的两棵(左边是删除相同事务的 TD-FP-Tree,右边是 new_TD-FP-Tree)

注意原 TD-FP-Tree 的 gb 路径中 b 节点由于计数减为 0 而移除,HeaderTable 中的计数经过更新,同 Tree 中节点计数之和并不完全对应。

Step9:将 new_TD-FP-Tree 与原 TD_FP_Tree 合并。这里产生的 new_TD-FP-Tree root 节点下只有一个 d 节点,是新增的频繁项,原 TD-FP-Tree 中不存在,所以将两棵树的 root 节点直接合并即可。维护 HeaderTable 和 NodeLink 后得到更新后的 TD_FP_Tree。至此,算法完成。

3 实验结果及分析

希望通过实验验证 PFU-TD-FP 算法在两方面的优势,一是同新事务集到来时完全重建 TD-FP-Tree 的效率对比,二是同基于 FP-Tree 结构的增量挖掘的效率对比,后者需要原始事务数据库建立 FP-Tree。因此进行以下 2 个实验:

(1)实验一:(更新算法同完全重建的比较)当新增事务集到来时,使用 PFU-TD-FP 更新 TD-FP-Tree 与完全重建 TD-FP-Tree 的耗费时间比较。

(2)实验二:(基于 TD-FP-Tree 结构的增量挖掘同基于 FP-Tree 结构的增量挖掘的比较)为原始数据库建立 FP-Tree 和 TD-FP-Tree,使用 FP-Growth 和 TD-FP-Growth 算法挖掘原始数据库,加入新增事务集,分别使用 PFU-TD-FP 和 FUFP 更新 Tree 结构后再次挖掘的总时间比较。

实验在 Windows 7 操作系统、i3 2.53GHz CPU、2G 内存的环境下进行,使用 VS2010 编写 C++程序实现,采用文献[2]中的 Connetct-4 数据集。这是一个来自 UC_Irvine 机器学习库的分类数据集,经过转换可得到 3 个事务集。3 个事务集均有 67557 个事务,单个项目数量 128,区别在于事务长度。实验一从中抽取事务平均长度为 4 的一类,可看做稀疏矩阵。实验二抽取事务平均长度为 34 的一类,这是稠密的事务集,意味着将有很多长频繁项,挖掘过程也会有更多的迭代。选择不同分类的原因在于实验的关注点不同。

实验一设置最小支持度为 10%,先选取 2000 个事务作为原始事务数据库,每次新增 2000 个事务,并将更新后的数据库作为下一次的原始数据库。图 3 展示了完全重建 TD-FP-Tree 和使用 PFU-TD-FP 更新 TD-FP-Tree 的执行时间。

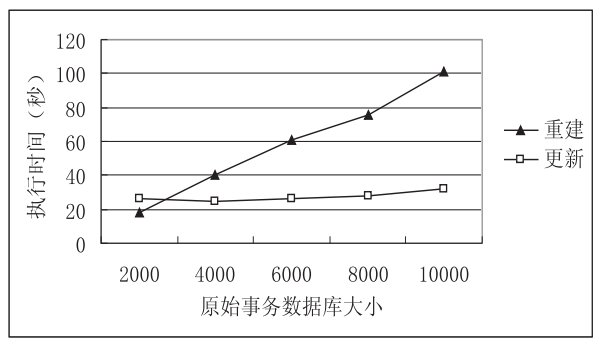

图 3 更新与重建的执行时间比较

可以观察到在实验刚开始时,更新时间甚至多于重建,因为此时新增事务集与原始数据库的大小是一

致的,但更新的处理步骤比重建要繁琐。而后两者的时间耗费逐渐接近,随着事务数据库越来越大,更新的优势越来越明显。这是因为总的事务数量不断增多,重建 TD-FP-Tree 时要两次扫描大量事务,时间耗费也越来越大,而每次新增事务集的数量则是固定的,时间耗费也趋于稳定。

实验二设置最小支持度为 80%,原始事务数据库 2000 个事务,新增事务集 500 个事务。实验二模拟实际应用中增量挖掘的全过程,比较的是基于不同 Tree 结构的效率,前者是 PFU-TD-FP 同 TD-FP-Growth 的组合,后者是 FUFP 同 FP-Growth 的组合。最后的执行时间包含三部分,一是原始数据库挖掘时间,二是更新 Tree 结构时间,三是更新后数据库挖掘时间。图 4 显示了这些时间。

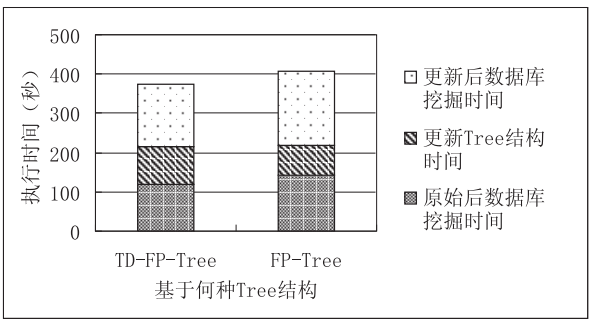

图 4　基于不同 Tree 结构的增量挖掘比较

从图 4 中观察到在更新 Tree 结构时 TD-FP-Tree 可能花费更多的时间,这是因为两者处理 Rescan_Transactions 的策略不同,文中提出的 PFU-TD-FP 算法需要先删除原 Tree 中表示相同事务的路径,而 FUFP 则只要在 Tree 的末端添加新节点即可。但是由于 TD-FP-Growth 的挖掘速度更快,所以基于 TD-FP-Tree 的增量挖掘在总时间上比基于 FP-Tree 的要少。

4　结束语

文中基于 FUP 的思想提出快速更新 TD-FP-Tree 结构的算法,根据项在原始数据库和新增事务集中是否频繁将其分类处理,可以减少重新扫描原始数据库的次数,并减少重排序事务中项时依赖项的个数,从而达到快速更新 TD-FP-Tree 的目的。文中针对 TD-FP-Tree 的结构特点提出了可行的更新方案,并采用并行处理的方法进一步提高效率。实验表明,文中提出的算法不仅可以快速更新 TD-FP-Tree,而且在同基于 FP-Tree 结构的增量挖掘中也有更好的表现。今后将继续在 TD-FP-Tree 快速更新算法上的研究,研究方向是将其应用到实际系统中,如个性化推荐、元搜索引擎等。

参考文献:

[1] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]//Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, TX: [s. n.], 2000: 1-12.

[2] Wang K, Tang L, Han J, et al. Top Down FP-Growth for Association Rule Mining[C]//Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Taipei, Taiwan: [s. n.], 2002: 334-340.

[3] Cheung D W, Han J, Ng V T, et al. Maintenance of discovered association rules in large databases: An incremental updating approach[C]//The Twelfth IEEE International Conference on Data Engineering. New Orleans, La: [s. n.], 1996: 106-114.

[4] 杨学兵,安红梅. 一种高效的关联规则增量式更新算法[J]. 计算机技术与发展,2007,17(1):108-113.

[5] Hong T P, Lin J W, Wu Y L. A fast updated frequent pattern tree[C]//The IEEE international conference on systems, man, and cybernetics. Taiwan: [s. n.], 2006: 2167-2172.

[6] Lin C W, Hong T P, Lu W H. The Pre-FUFP Algorithm for Incremental Mining[J]. Expert Systems with Applications, 2009, 36(5): 9498-9505.

[7] Li C X, Zhao L. Improved Incremental Mining Algorithm[J]. Computer Engineering, 2010(24): 42-44.

[8] Zou H, Zhu S H. Research on Incremental Mining Algorithm Based on HFUFP-Tree[J]. Computer Applications and Software, 2011(9): 102-105.

[9] 邹　海,朱四红. 基于 HFUFP-tree 的增量挖掘算法研究[D]. 合肥:安徽大学,2010.

[10] 钱　峰,张　蕾. 一种以 FP-tree 为基础的增量式挖掘算法的研究[J]. 科技信息,2009(24):386-389.

[11] Leung C K, Khan Q I, Li Z, et al. CanTree: a canonical-order tree for incremental frequent-pattern mining[J]. Knowledge and Information Systems, 2007, 11(3): 287-311.

[12] 邹力鹍,张其善. 基于 CAN-树的高效关联规则增量挖掘算法[J]. 计算机工程,2008(3):29-31.

(上接第 90 页)

[10] 陈飞宏. 基于向量空间模型的中文文本相似度算法研究[D]. 成都:电子科技大学,2011.

[11] 周　博,刘奕群,张　敏,等. 锚文本检索有效性分析[J]. 软件学报,2010,22(8):1714-1724.

[12] Eiron N, Ks M. Analysis of anchor text for Web search[C]//Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Toronto, Canada: [s. n.], 2003.

一种基于FUP的TD-FP-Tree并行快速更新算法

作者： 周爱武， 王琰， 陈宝楼
作者单位： 安徽大学 计算机科学与技术学院,安徽 合肥 230039
刊名： 计算机技术与发展
英文刊名： Computer Technology and Development
年，卷(期)： 2013(4)