基于垂直压缩格式的高效FP-STREAM算法的研究

唐耀红 魏慧琴
(北京交通大学计算机与信息技术学院 北京 100044)

摘 要 近年来由于信息的爆炸式增长,数据流频繁模式挖掘逐渐成为研究的热点。FP-Stream作为经典的数据流频繁模式的挖掘算法,实现了多时间粒度的挖掘,但是该算法并未对数据本身进行压缩,使其在一定时间内处理的数据量受到限制,存在有限内存和高速海量数据的矛盾。通过对数据流进行垂直和Dif-bits压缩变换来改进FP-Stream算法,大大降低了内存需求,提高了数据处理能力。经过实验证明,改进算法是有效的。

关键词 数据流,频繁模式,FP-Stream,垂直格式,Dif-bits数据压缩
中图法分类号 TP391 **文献标识码** A

Efficient FP-STREAM Algorithm Based on Vertical Compression Data Format

TANG Yao-hong WEI Hui-qin
(Institute of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract Along with the sharp increment of the information, mining frequent itemsets gradually becomes a hot point in recent years. FP-Stream is a classic algorithm for mining frequent itemsets at multiple time granularities. But the weakness is the contradiction of the massive data and the limited memory in a certain time which will lead to the result that the algorithm can not be used in high speed data stream mining. This article proposed a improved FP-Stream algorithm based on vertical and Dif-bits compression data format.

Keywords Data stream, Frequent itemsets, FP-Stream, Vertical format, Dif-bits data-compression

1 引言

近年来,随着信息的急剧增长,特别是高级数据系统技术和万维网的迅速发展,各种高速海量复杂的数据给传统的数据挖掘技术带来新的挑战,数据流挖掘技术逐渐成为热点问题。在网络监控、入侵检测情报分析、电子商务、生物医学、股票交易、卫星遥感等众多领域中,数据均以流的形式存在。

其中频繁模式挖掘是关联规则、相关分析、序列模式等许多重要任务的核心,也是其中开销最大的步骤,因此数据流挖掘技术的效率很大程度上取决于频繁模式挖掘的效率。许多针对静态数据集的经典算法都是建立在获取完整数据集并多遍扫描的基础上的,如Apriori[1]、FP-Growth[2]、CLOSET[3,4]、CHARM[5],它们难以进行增量式的更新,而数据流在存储预知潜在的数据和庞大的历史数据上存在很大困难,因此传统的频繁模式挖掘技术很难应用到数据流中。并且在数据流中,随着时间的推移,频繁和非频繁项集之间会存在一定程度的转化,频繁可能变为非频繁,相反亦之,这就要求存储结构能随时间动态调整,以反映数据的变化。在数据流挖掘中,大多数情况下数据的变化趋势比数据本身更重要。

FP-Stream[6]作为经典的数据流频繁模式挖掘算法实现了单遍扫描、多时间粒度的挖掘,并且提供了灵活的查询机制,以根据需要查询不同时间粒度的频繁项集。但是此算法仍然存在高速海量数据处理和有限内存的矛盾,当数据流速度很高时,到达的大量数据和中间结果存储在内存中是个挑战。本文提出一种基于垂直格式的Dif-bits[9]数据压缩算法,该算法可缩减数据对内存的占用,大大提高数据处理能力,进一步扩大应用范围。

2 相关工作分析

近几年,由于电子商务、卫星遥感、金融服务、生物医学等众多应用领域的需求,数据流的挖掘问题受到越来越多的关注。

在数据流的频繁项挖掘方面,许多学者提出了自己的算法,主要思路分为批处理和启发式两种。批处理就是对数据流分块处理,通过局部频繁项集来求解全局频繁项集。启发式则是通过已知的频繁项集按照某种策略逐渐生成新的频繁项集。其中比较经典的两个分别是Manku等人提出的Lossy Counting[7]算法和Chris Giannella等人提出的FP-Stream[6]算法,两种思想都是批处理方式。Lossy Counting算法相当简单并且非常有效,对数据流只进行一遍扫描,并将数据流分成事务相等的桶,每次处理尽可能多的桶。该算法保证最后的查询结果一定包含流数据中的所有频繁项集和一小部分非频繁项集,算法简单,容易实现。但是频度列表可能随数据流无限增长,很可能造成空间不足,而且没有考虑到不同时期的权重问题,时间敏感性不强。一种可行的改进算法就是将倾斜时间窗口整合到该算法中,以实现时间的敏感。FP-

到稿日期:2012-11-12 返修日期:2012-04-08
唐耀红 女,硕士,主要研究方向为网络与数据库,E-mail:tangyaohong11@126.com;魏慧琴 女,副教授,主要研究方向为网络与数据库。

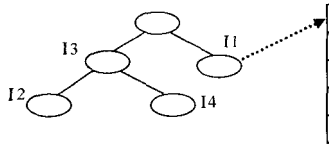
Stream[6]作为数据流挖掘的经典算法，将倾斜时间窗口融入到频繁模式树(Pattern-Tree)中，从而使算法可以实现多时间粒度，满足对时间敏感的查询。实际中，连续的一段时间内频繁项集存在很大程度上的交集，发生概念漂移的现象并不多，大部分具有代表意义的频繁集保持着它的属性，所以使得存储多时间粒度的频繁模式树在一段时间内具有很好的稳定性，最主要的开销就是更新节点处的时间窗口，其他的操作比如创建新的分支等并不多，这给 FP-Stream 算法提供了时间和空间上的可能性。Chris Giannella 等人在文献[6]中已经通过实验证明，该算法具有很好的时间效率和稳定性。但是 FP-Stream 算法在接收分块数据时并未对其进行压缩处理并且算法会产生大量的中间结果而使算法对空间的要求比较高，从而限制了其处理数据能力，使其不适用于高速数据流的处理。如何进一步压缩算法对内存的需求，是算法改进的一个方向。

3 FP-Stream 改进算法

3.1 FP-Stream 算法描述

Jiawei Han 等人提出的 FP-Growth[2]算法中使用一种紧缩的数据结构来存储查找频繁项目集所需要的全部信息，将提供的频繁项集压缩到一棵 FP tree 中，并且保存关联信息，这彻底地脱离了 Apriori[1]算法必须产生候选项集的传统方式，大大地提高了时间和空间的效率。FP-Stream 是基于 FP-tree 结构的算法，此结构成功地应用到了数据流挖掘中，并且提供了多时间粒度的保存，满足了时间敏感的查询需求。

FP-Stream 数据结构主要包含两个部分：(a)FP-tree；(b) Pattern -tree。Pattern-tree 实际上就是嵌入倾斜时间窗口的 FP-tree，其数据结构如图 1 所示。



Time window	support
t3	75
t2	69
t1	36
t0	20

图 1 Pattern Tree

其中 I1，I2，I3，I4 是事务中的 item，每个节点存储的时间窗口代表的是从根到该节点路径上的 item 组成的事务集在不同时间粒度上的绝对支持度。倾斜时间窗口常见的有自然倾斜时间窗口、对数尺度倾斜时间窗口和渐进型的对数尺度倾斜时间窗口，详见文献[16]。

FP-Stream 采用分批处理方式，到达的每块数据用 Bi 表示，算法流程如下：

1. 第一块数据 B1 单独处理，作为初始化，扫描数据块，创建一个按照 item 支持度递减的 list，并利用支持度 σ 和最小误差 ε 去掉非频繁的 item，然后创建 FP-Tree，并应用 FP-Growth 算法产生频繁数据集，构建初始的 Pattern-Tree。

2. 对于之后到达的每块数据则按照以下流程处理：

(1)清空 FP-Tree。

(2)将 Bi 中的事务按照步骤 1 中生成的 list 排序，插入到 FP-tree 中，并且不进行裁剪。

(3)运用 FP-Stream 算法挖掘频繁项集 I，并分两种情况处理：

①如果 I 在 Pattern -Tree 中：

i)将 $f(B)$加入到倾斜时间窗口中；

ii)对时间窗口剪枝；

iii)如果 I 的倾斜时间窗口为空则不再挖掘 I 的超集(反单调性)；

iv)如果 I 的倾斜时间窗口不为空则继续挖掘 I 的超集。

②如果 I 不在 Pattern-Tree 中：

i)项集 I 满足 $f_I(B)\geqslant\varepsilon|B|$，将 I 插入到 Pattern-Tree 中，此时 I 的倾斜时间框架只有一个时间框；

ii)如果 $f_I(B)<\varepsilon|B|$，则直接丢弃，并且 FP-Growth 算法停止挖掘 I 的超集(反单调性)。

3)借用深度优先搜索算法扫描 FP-Stream，对于每一个未更新的项集在时间窗口中插入 0，并裁剪窗口。

4. 当扫描到叶子节点时，如果出现倾斜窗口为空，则删除叶子节点，接着检查其兄弟节点，完成后返回父节点，如果父节点的所有孩子节点都被删除了，则父节点成为叶子节点，进行递归操作，直到根节点为止。

经过以上的步骤，多时间粒度的频繁项集就被存储在 Pattern-Tree 中，用户可以查询自己感兴趣的时间粒度，更详细的算法介绍参见文献[2]。

3.2 FP-Stream 算法改进

通过前面对 FP-Stream 算法的分析和描述得知该算法对到达的数据直接存储而未经压缩，虽然后来采用 FP-Tree 进行压缩存储，但在插入之前，将每块数据读入到内存中时已经占用了较大的内存空间，这就限制了处理数据块的大小，进而限制了该算法在高速数据流中的应用。另外在 FP-Growth 算法中，递归的调用会产生大量的条件模式树，也会产生很大的内存消耗，在存储倾斜时间窗口时，存在类似的问题，因此该算法仍然存在高速海量数据和有限内存之间的矛盾。本文对 FP-Stream 算法进行改进，主要是对初始到来的数据进行垂直格式变换并且应用 Dif-bits 进行数据压缩，大大减少内存的使用量，从而使该算法更高效，以得到更广泛的应用。

3.2.1 基本概念

定义 1(水平数据格式) 水平数据格式是二元组(TID，项目集)。TID 是一个事务的标识符，项目集是一个 TID 中的项目集合。

定义 2(垂直数据格式) 垂直数据格式是二元组(项目，TID 集合)，项目是一个项目的名字，TID 集合是包含该项目的事务标识符的集合。

Dif-bits 算法：Dif-bits 采用垂直数据格式，用垂直格式列表中的当前事务标志减去前一个事务标志得到事务标识差 Di，然后将 Di 转化为二进制格式，再对二进制格式进行压缩处理。以 TID 集合{2,4,5,8}为例，算法先计算事务差 Di，得到差值的集合为{2,2,1,3}，差值集合转化为比特格式，同时进行压缩去掉高位 0 得到 1111010，仅用了 7bits。但是这种记录方式给还原带来了困难，因此采用添加填充位的方式，插入 n 的二进制比特作为填充位，n 代表后面要读取的二进制位的长度，但是为了还原方便各个差值的 n，必须采用固定的位数、实验证明，5bits 的 n 几乎能满足大部分的应用需求，因为 5bits 能表示 n 的最大值为 31，其所能表示的整数大于 2 亿，在这样大的间隔内出现一次几乎不可能，即使出现也不可能成为频繁项集，直接忽略即可。按照上述陈述，则转化后的 Dif-bits 形式即为 11000101000001000101000010，共 27bits，如

果按照整形存储，则 4 * 32＝128bits，显然 Dif-bits 节省了大量的存储空间，另外由于 Dif-bit 采用的是差值存储，使得差值大小与事务集中事务总数无关，从而进一步减少了所占用的空间。至于该算法的解压算法只需要做相反操作即可。

3.2.2 改进算法描述

改进算法的主要思想就是将垂直格式的 Dif-bits 算法应用到 FP-Stream 中来压缩事务集的内存占用量，从而提高算法的数据处理能力，满足高速数据的要求。基本思路：首先扫描到来的数据块，将数据块由水平格式转化为垂直格式，并建立类似文献[8]中的垂直项目头表，主要区别就是融入了 Dif-bits 算法对项目的事务列表进行压缩，事务列表不采用链表格式而是采用 Dif-bits 算法处理后的二进制的 bit 串，垂直项目头表是一个四元组(Item，Count，Tids，N_link)。这 4 个域的含义分别是：项目名称、支持度计数、项目对应的事务集比特串、FP-Tree 项目链头。将 Dif-bits 压缩算法应用于项目的事务列表中，不但便于数据增量更新，而且可以大大减少传统垂直数据格式中随着事务集增大而使项目的事务列表急剧扩张的情况。事务集{*I1 I2 I3*}、{*I2 I3*}、{*I3 I4 I5*}、{*I1 I2*}、{*I3*}、{*I2 I3 I4 I5*}、{*I2 I3*}、{*I2 I3 I4*}、{*I2 I3 I4 I5*}初始的状态如图 2 所示。

Item_name	Count	Tids	N_link
I3	8	110001……	^
I2	7	110001……	^
I4	4	110001……	^
I5	3	1100010……	^
I1	2	1100010……	^

图 2 扫描数据库形成的初始垂直项目表

建立 FP-tree 的过程和文献[8]中的过程类似，只是在处理的过程中要加入数据的解压算法，以确定属于同一事务中的项目集。由于压缩和解压算法简单、时间效率较高，因此只要稍微牺牲时间效率就可以换取大幅度的空间效率。插入过程的基本思路：纵向每次扫描 Tids 的低位，形成当前最小事务号的项目集。然后将所得的项目集插入到 FP-Tree 中，接着释放掉这个项目集的存储空间，因为存储的事务号是按照升序存储的，所以每次找到的当前最小事务号都是在 bit 串的低位，这便于随时释放已经插入到 FP-tree 的 bit 串的存储空间。插入所有事务之后的垂直项目头表如图 3 所示。

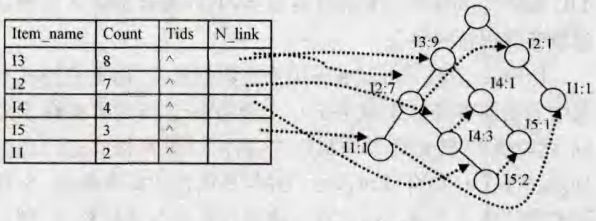


图 3 插入所有事务后的垂直头目表

该改进算法将垂直格式的事务列表和 FP-tree 的垂直头目表合二为一，并将 Dif-bits 压缩算法应用到项目事务列表中，其优势在于：一方面垂直项目表的前两列由两份缩减成一份，在 item 数量巨大的环境中达到了较好的压缩效果；另一方面，将项目事务列表进行压缩，可以防止随着事务数目的增加对内存的需求急剧增长的情况。该改进算法通过对经过横向和纵向两个方向进行压缩达到了较高的内存利用率，提升了算法处理数据的能力。

通过前面的分析可知另外一个空间需求较大的就是倾斜时间窗口的存储，同样可以将其 Count 转化为二进制比特，但是会涉及到时间窗口的更新算法，第一个基础数据会频繁地变化，不利于采用差值的方式存储，所以直接使用带填充位的二进制串即可，这样在用户查询的时候解压算法也相对简单，提高了反应速度，提供了较好的用户体验。

总之，通过将垂直数据格式和 Dif-bits 算法融入到 FP-stream 算法中，大大压缩了算法的空间需求，提高了算法的数据处理能力，这为其在高速数据流中的应用提供了可能。

4 实验结果分析

为了进一步测试改进算法的性能，我们用 QT4.7.4，在内存 1.99GB、CPU 为 Intel(R) Core(TM)2 Duo CPU E7200、操作系统为 Windows XP 的 PC 机上实现改进算法，并与同环境下的原始算法进行对比。实验的数据集来自 http://fimi.cs.helsinki.fi/data/，选择了其中的 accidents、pumb_star、T10I4D100K、T40I10D100K 4 个数据集进行测试，accidents 和 pumb_star 分别是交通事故和人口普查的数据集，详细介绍参见网站链接，T10I4D100K 和 T40I10D100K 是用 IBM 的数据产生器产生人造数据，IBM 的数据产生器可以在官方网站上下载，默认下载版本是 linux 下：http://www.almaden.ibm.com/cs/quest/syndata.html/#assocSynData/，相关使用参数说明参见链接 http://bbs.pinggu.org/forum.php?mod=viewthread&tid=900057&page=1&fromuid=2877370。由于原始算法和改进算法在 FP-Tree 和 Pattern-tree 的存储上所占用的存储空间是一样的，这里只对比事务集存储上的差别，实验结果如图 4 所示。

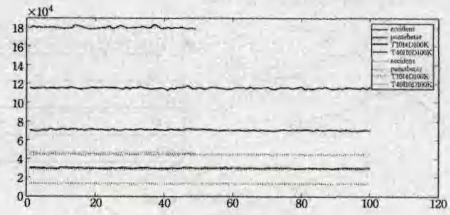


图 4 FP-Stream 与改进算法内存占有量对比

为了模拟数据流的分批次处理和体现时间敏感的特性，实验中将数据集分隔成事务数目相等的数据块，每 1000 条数据作为一个数据块，图中横坐标表示的数据块号，取前一百块作为参考，纵轴表示两种算法中每块事务集占用的内存的字节数目，实线表示原始算法的内存占用值，虚线表示改进算法的内存占用值，因为数据集 pumb_star 的数据量较小，不满一百块数据，其形成的曲线较短，同一颜色的实线和虚线代表同一数据集在原始算法和改进算法中的结果。

然而通过实验证明，改进的算法对时间效率的影响是细微的，下面通过表 1 加以说明。

表 1 改进算法与原始算法时间效率对比

数据集	时间差(ms)
T40I4D100K	10359
T10I4D100K	875
accidents	30313
pumb_star	469

表中的第一列代表数据集的名称，是实验中选取的 4 个数据集，第二列时间差的含义是改进算法在挖掘频繁项集耗费的总时间减去原始 FP-Stream 算法耗费的总时间，单位是 ms。表中的时间差是相对于整个数据集而言的，当平均到每批数据上，其值是很小的，几乎可以忽略，以 pumb_star 为例，每批数据的处理时间差值是 8ms 左右，这个时间差值完全在可接受的范围之内，因此算法在时间上是可行的。

通过结果集可以看出，改进算法在事务集读入内存后占用的存储空间较原始算法有很大程度上的提高，且稳定性并没有受到影响。显然改进算法可以提高数据处理能力，扩展 FP-Stream 算法的应用。

结束语 通过实验结果分析可知，改进的 FP-Stream 算法具有更高的数据处理能力，但是仍然存在需要改进的地方，对于 FP-Tree 和 Pattern-tree 这两个主要的数据结构来说，对内存的需求仍然很大，因此，提出针对该数据结构的进一步压缩算法，将会进一步改善算法的效率。前面提出的对倾斜时间窗口存储的压缩是一个方面。此外，在 FP-Tree 产生频繁项集的算法中，大量的中间结果也会造成内存紧张，这方面需要进一步改进。也有很多学者提出了相关的改进算法，如文献[11]的 LR-Trie 和文献[10]的 NBFP-Tree 等。提出更高效的算法将是我们下一步的工作。

参 考 文 献

[1] Agrawal R, Srikant R. Fast Alogoritm for mining Association Rules[C]// Proc 20th Int Conf Very Large Data Bases VLDB (1994). Volume 1215, Citeseer, 1994: 487-499

[2] Han Jia-wei, Pei Jian, Yin Yi-wen. Mining Frequent Patterns without Candidate Generation[C]// SIGMOD '00 Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, 2000

[3] Pei Jian, Han Jia-wei. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets[Z]. DMKD. May 2000

[4] Wang Jian-yong, Han Jia-wei, Pei Jian. CLOSET+: searching for the best strategies for mining frequent closed itemsets[Z]. ACM SIGKDD, August 2003

[5] Zaki M J, Hsiao C-J. CHARM: An efficient algorithm for closed association rule mining[C]// 2nd SIAM International Conf. on Data Mining. 1999

[6] Giannella C, Han Jia-wei, Pei Jian, et al. Mining frequent Patterns in Data STream at Multiple Time Granularities[J]. Next generation data mining, 2006, 35(1)

[7] Xenfontas, Hurley P, Kind A. Probabilistic lossy counting: an efficient algorithm for finding heavy hitters[J]. ACM SIGCOMM Computer Communication, 2008, 38(1)

[8] 周莉，张吉赞. 用垂直格式构建 FP 增长树的算法[J]. 计算机科学与工程，2009，45(8)：161-164

[9] Ashrafi M, Taniar D, Smith K. An efficient compression Technique for vertical Mining Methods[M]. Research and Trends in Data Mining technologies and applications, 2007: 151-163

[10] 吕橙，郝莹，张翰韬. 基于垂直二进制位图的频繁模式挖掘算法[J]. 山东大学学报，2007(5)：24-29

[11] 徐艳红. 基于倾斜时间窗口的频繁项集挖掘算法研究[D]. 哈尔滨：哈尔滨工程大学，2010

[12] 孟彩霞. 面向数据流的频繁项集挖掘研究[J]. 计算机工程与应用，2010，46(24)

[13] Manku G S, Motwani R. Approximate frequency counts over data streams[C]// Proceedings of the 28th VLDB Conference. HongKong, China, 2002: 346-357.

[14] Carmona J M, Baena-García M, Campo-Avila J, et al. Using GNUsmail to Compare Data Stream Mining Methods for On-line Email Classification[C]// Journal of Machine Learning Research-Proceeding Track. 2011: 12-18

[15] 孙志长，冯祖洪，王沛栋. 一种高效的混合压缩数据挖掘算法[J]. 计算机应用研究，2009(10)：3738-3742

[16] Han Jia-wei, Kamber M. Data Mining Concept and Techniques (Second Edition)[M]. Morgan Kaufmann, 2006

[17] Ren J, Li K. Online data stream mining of recent frequent itemset based on sliding window model[J]. Proceedings of 2008 International Conference on machine and Cybernetics, 2008, 1(7)

[18] Leung C K-S, Brajczuk D A. Efficient Mining of frequent itemsets from data streams[J]. Information and Knowledge, LNCS, 2008, 5071: 2-14

[19] Jea K-F, Li Chao-wei, Chang T-P. An efficient approximate approach to mining frequent itemsets over high speed transactional data streams[C]// ISPA 2008: English International Conference on intelligent Systems Design and Applications. 2008, 3: 275-280

(上接第 159 页)

算法的时空分析模型。

参 考 文 献

[1] Gaede V, Gunther O. Multidimensional Access Methods [J]. ACM Computing Surveys, 1998, 30(2)

[2] 卢凤顺，宋君强，银福康，等. CPU/GPU 协同并行计算研究综述[J]. 计算机科学，2011，38(3)：5-9

[3] 杨珂等. 图形处理器在数据库技术中的应用[J]. 浙江大学学报：工学版，2009，43(8)：1349-1360

[4] Litwin W. Linear hashing: a new tool for file and table addres-sing [C]// VLDB1980. Montreal, 1980: 212-223

[5] Lehman T J, Michael J. A Study of Index Structures for Main Memory Database Management Systems[C]// Proceedings of the 12th International Conference on Very Large Data Bases. August 1986: 294-303

[6] Ouksel A M, Scheuermann P. Storage Mappings for Multidimensional Linear Dynamic Hashing[C]// Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. 1983: 90-105

[7] Ouksel A M, Abdul-Ghaffar J. Concurrency In Multidimensional Linear Hashing[C]// Proceedings of FODO. 1989: 233-240

[8] 周伟明. 多核计算与程序设计[M]. 武汉：华中科技大学出版社，2009：19-25

[9] NVIDIA CUDA (Compute Unified Device Architecture)[OL]. http://developer.nvidia.com/object/cuda.html