CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 41 卷第 5 期 2019 年 5 月
Vol. 41, No. 5, May. 2019

文章编号:1007-130X(2019)05-0780-08

稀疏数据频繁项集挖掘算法研究综述*

肖　文,胡　娟
(河海大学文天学院,安徽 马鞍山 243031)

摘　要:频繁项集挖掘 FIM 是最重要的数据挖掘任务之一,被挖掘数据集的特征对 FIM 算法的性能有着显著影响。在大数据时代,稀疏是大数据的典型特征之一,对传统 FIM 算法的性能带来严峻挑战。针对在稀疏数据中如何高效进行 FIM 的问题,从稀疏数据的特征出发,分析了稀疏数据对 3 种类型 FIM 算法性能的主要影响,对已经提出的稀疏数据 FIM 算法进行了综述,对算法中采用的优化策略进行了讨论,最后通过实验对代表性的稀疏数据 FIM 算法进行了性能分析。实验结果表明,采用伪构造策略的模式增长算法最适合用于稀疏数据的 FIM,在运算时间和存储空间上,相比其他算法该算法具有较大的优势。

关键词:大数据;稀疏数据;频繁项集挖掘;性能分析;综述
中图分类号:TP391　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2019.05.003

A survey of frequent itemset mining algorithms for sparse dataset

XIAO Wen, HU Juan
(Wentian College, Hohai University, Maanshan 243031, China)

Abstract: Frequent itemset mining (FIM) is one of the most important data mining tasks. The characteristics of datasets have a significant impact on the performance of FIM algorithms. In the era of big data, sparseness, a typical feature of big data, brings severe challenges to the performance of traditional FIM algorithms. Aiming at the problem of how to perform FIM in sparse datasets efficiently, based on the characteristics of sparse datasets, we analyze the main effects of sparse datasets on the performance of three FIM algorithms, summarize current sparse datasets FIM algorithms, discuss the optimization strategies used in these algorithms, and analyse the performance of the typical sparse datasets FIM algorithms through experiments. Experimental results show that the pattern growth algorithm with pseudo-structural strategy is most suitable for FIM in sparse datasets and outperforms the other two algorithms in both operation time and storage space.

Key words: big data; sparse data; frequent itemset mining (FIM); performance analysis; survey

1　引言

频繁项集挖掘 FIM(Frequent Itemset Mining)是最重要的数据挖掘任务之一,是关联规则挖掘、聚类、离群点分析等众多数据挖掘任务的基础,自从它被提出以来[1],受到了越来越多的关注。主要的 FIM 算法可以分为 3 类:第 1 类为产生测试类算法,如 Apriori 算法及其众多优化算法。第 2 类为模式增长类算法,如 FPGrowth 算法及其一系列优化算法。上述 2 类算法都是对水平格式的数据进行挖掘。第 3 类算法是基于数据垂直格式的

* **收稿日期**:2018-08-10;**修回日期**:2018-10-18
基金项目:安徽省高校优秀青年人才支持计划(gxyq2018139)
通信地址:243031 安徽省马鞍山市河海大学文天学院
Address: Wentian College, Hohai University, Maanshan 243031, Anhui, P. R. China

挖掘算法,如 Eclat 算法及其一系列优化算法。

FIM 问题的完整搜索空间是一棵由数据集中频繁项的集合构成的枚举树[2-4],可以利用广度优先搜索 BFS(Breadth First Search)或深度优先搜索 DFS(Depth First Search)的方法在枚举树进行查找。所有的 FIM 算法都包含 2 个基本计算步骤:产生候选项集和计算候选项集的支持度。具体来说:产生测试类算法通过 BFS 查找枚举树产生候选项集,通过迭代扫描数据集确定候选项集的支持度;模式增长类算法通过 DFS 查找枚举树产生候选项集,产生项集的同时即完成计数;基于垂直数据格式的算法通过 BFS 查找枚举树产生候选项集,通过对项集事务标识集进行交运算得到候选项集的支持度。

被挖掘数据的特征对 FIM 算法性能有着显著的影响。数据集的基本特征有事务数、项数、平均事务长度等,有研究关注了不同类型 FIM 算法受上述特征性能影响的问题[5]。随着计算机技术和通讯技术的快速发展,人类已经进入"大数据时代",稀疏是大数据的主要特征之一,具体表现为数据事务数极大、数据项(频繁项)的数量极大、事务之间的差异度极大等,且在 FIM 任务背景下,数据集稀疏度与最小支持度呈反比等[6]。为了提高稀疏数据 FIM 问题的效率,很多研究对上述 3 类 FIM 算法进行了优化,提高了挖掘稀疏数据时 FIM 算法的性能。

本文主要贡献为:

(1)对已提出的稀疏数据 FIM 算法进行了综述,对算法中的优化策略进行了讨论。

(2)提出了适用于稀疏数据 FIM 的产生测试类算法 FastL2Apriori,运行效率比其他同类算法提高了一个数量级。

(3)通过实验对代表性稀疏数据 FIM 算法性能进行了实证分析,对不同类型的算法进行了横向比较。

2　符号及相关定义

设 $I=\{i_1,i_2,\cdots,i_n\}$ 为项的集合;X 称为一个项集,即 $X=\{i_1,i_2,\cdots,i_k\}\subseteq I$,若项集包含 k 个项则称该项集为一个 k-项集;一个事务 T 的定义为 $T=(tid,X)$,其中 tid 为事务标识,X 为一个项集;一个事务数据集 $D=\{T_1,T_2,\cdots,T_n\}$ 是 T 的集合;D 的垂直格式 D' 由项集和含有该项集所有事务的事务标识组成,具体定义如下:

$$D'=\{(i_j,C_{ij}=\{tid\mid i_j\in X,(tid,X)\in D\})\}$$

其中,C_{ij} 也称为项 i_j 的事务标识集合 $tidset$。

项集 Y 的支持度定义为事务数据集 D 中包含项集 Y 的事务的数量。正式定义为:

$$Support(Y)=|\{tid\mid Y\subseteq X,(tid,X)\in D\}|$$

在垂直数据库 D' 中,项集 Y 的支持度为其 $tidset$ 的长度。

如果一个项集被称为是频繁的,则其支持度不小于用户给定的一个阈值,这个阈值也可以称为最小支持度 $minsup$(minimum support)。

事务数据集 D 频繁项的集合 $FList$ 定义为:

$$FList=\{i\mid Support(i)\geqslant minsup\ \&\&\ i\in I\}$$

D 中一个事务 T 的频繁项投影 T' 定义为:

$$T'=\{\ i\mid i\in T\ \&\&\ i\in FList\ \}$$

频繁项集具有单调特性:如果一个项集是频繁的,则其所有子集都是频繁的。相应地,如果一个项集不是频繁的,则其所有的超集都是不频繁的。

FIM 的整个搜索空间是一棵由 D 中 $FList$ 构成的枚举树。设 $FList=\{i_1,i_2,\cdots,i_n\}$,一棵枚举树定义为:

(1)在树中每一个节点对应一个候选项集;

(2)树的根节点为 null;

(3)设 $I=\{i_1,i_2,\cdots,i_k\}$ 是一个频繁项集,则 I 的父节点对应的频繁项集为 $\{i_1,i_2,\cdots,i_{k-1}\}$。

枚举树的所有节点即为 FIM 的整个搜索空间,可以从定义中看出,若 $|FList|=n$,则整个搜索空间的尺寸为 2^n-1。若有 $FList=\{a,b,c,d\}$,则它的枚举树如图 1 所示。

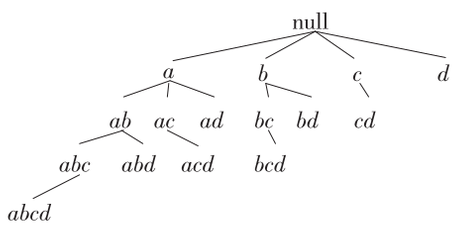


Figure 1　An example of FIM search space

图 1　FIM 搜索空间示例

3　已提出算法综述

所有 FIM 算法都包含 2 个基本的计算任务:产生候选项集和对候选项集进行计数。FIM 的基准算法如算法 1 所示。

算法 1　FIMBaseLine(D,$minsup$)

输入:事务数据集 D,最小支持度 $minsup$。

输出:所有频繁项集。

步骤:

1. $FrequentItemsets=\{\}$;
2. 将 $FList$ 中的项插入 $FrequentItemsets$ 中;
3. until 所有频繁项集都插入 $FrequentItemsets$ 中 do
   产生候选项集 p(从 $FrequentItemsets$ 已有项集中产生);
   if($Support(p)\geqslant minsup$)
   将 p 插入 $FrequentItemsets$ 中;
4. end

设 D 中$|FList|$的值为 n,则 FIM 基准算法的时间复杂度为 $O(2^n)$。虽然不同类型的 FIM 算法在设计思路上有所不同,但稀疏数据集频繁项数量极大、事务之间差异度大、平均事务长度较短等特征在性能上对所有类型的算法都带来了挑战。针对稀疏数据,需要分析不同类型算法在稀疏数据中进行 FIM 的性能瓶颈,有针对性地进行优化,提高算法性能。

3.1 产生测试类算法

此类算法的 2 个基本计算步骤为:从已产生的 k-频繁项集的集合 L_k 产生 $k+1$ 候选项集的集合 C_{k+1};对 C_{k+1} 中的所有候选项集进行计数产生 L_{k+1}。2 个步骤都需要多次迭代执行。由于候选项集的数量与$|FList|$呈指数关系,稀疏数据集的$|FList|$一般都十分大,会导致产生海量的候选项集,对算法性能产生显著影响,主要体现在候选项集数量增多会导致计数的成本显著增加。因此,针对稀疏数据集的优化方法可以从 2 个方面来考虑:(1)通过频繁项集单调性等属性对候选项集进行剪枝,以减少候选项集数量;(2)在迭代过程中对 D 进行剪枝,加快候选项集的计数速度,提高算法整体效率。

对候选项集剪枝最基础的策略是利用频繁项集的单调特性。Agrawal 等[1]提出非频繁项集所有的超集都不可能是频繁的(频繁项集所有的子集都是频繁的),通过检查 C_{k+1} 中候选项集的所有 k 子项集是否存在于 L_k 中来实现,若 $k+1$ 候选项集存在一个 k 子项集不属于 L_k,则该候选项集不可能是频繁的,可以从 C_{k+1} 中剪枝。这种剪枝策略虽然容易实现,但对每一个候选项集进行子集检查会带来 $O(n^2)$ 计算开销,特别是对稀疏数据集挖掘时选项集的数量极大,剪枝过程对算法整体性能有十分严重的影响。

Chen 等[7]提出一种简易的剪枝策略:在产生 C_k 的过程中,候选项集中所有的项 i 支持度 $Support(i)$大于 k,否则将该候选项集进行剪枝。虽然这种剪枝策略十分简单且易于实现,但稀疏数据中的频繁项集一般长度都很短(k 的值少且偏小),这种剪枝策略对于稀疏数据来说效果十分有限。

由于产生测试类算法是迭代进行的,可以充分利用上一次迭代中候选项集的计数结果来对候选项集进行剪枝,实际上是对单调性的进一步使用。Essalmi 等[8]提出了一种利用所有 $k-1$ 子集支持度最小值 $SupportMin$ 来确定 k 项集支持度最大值的方法进行剪枝。设 P 是 C_k 中的一个候选项集,则 P 的 $SupportMin$ 满足如下关系:

$SupportMin(P)=min(\{Support(q)\mid q\in P$ 的所有 $k-1$ 子集$\})$

$Support(P)\leqslant SupportMin(P)$

$Support(P)$的最大可能值可以从 $k-1$ 次迭代的计数结果中计算得到,如果该值小于 $minsup$,则项集 P 不可能是频繁的,可以从 C_k 中进行剪枝。利用 $SupportMin$ 进行剪枝同样需要检查候选项集的所有 $k-1$ 子项集,计算开销为 $O(n^2)$。

除了通过对频繁项的枚举树进行剪枝产生每一次迭代的候选项集之外,文献[9,10]提出了直接从事务中产生候选项集的方法。这种方法的优点是直接产生所有的候选项集,无需迭代,但其缺陷也十分明显:由于稀疏数据的频繁项数量极大且事务之间差异度极大,直接从事务产生的候选项集是海量的,其中多数候选项集都是不频繁的,增大了存储需求及计数成本。

针对稀疏数据,产生测试类方法除了通过对候选项集进行剪枝之外,还可以通过优化计数过程来加快运算速度。对事务进行剪枝最基础的策略就是将所有非频繁的项全部删去,根据频繁项集的先验性质,频繁项集中不可能出现非频繁项的项。文献[11]中提出了一种简单的事务剪枝策略,若事务中的一个项 i 在所有 k-频繁项集中从未出现过,则它在所有$(k+1)$-频繁项集中不可能出现,可以从事务中删去;文献[9]中同时提出了一种加快候选项集计数的策略:使用一个数组 $C(tid)$保存每一条事务的频繁项投影 T'的长度,在对 k-候选项集进行计数的过程中,所有 T'长度小于 k 的事务不用检查。Li 等[12]也提出一种对事务进行剪枝的策略,若事务 T 在第 k 次迭代的计数过程中未被检查,则在之后所有的迭代过程中均可以不检查进行剪枝。Deng 等[13]提出一种剪枝策略,同样是利用了上一轮迭代中的计数信息。设有项集 P 和项 i $(i\notin P)$,若有 $Support(P)=Support(P\cup\{i\})$,则对于任何项集 $A(A\cap P=\varnothing$ 且 $i\notin A)$,$Support($

$A\cup P)=Support(A\cup P\cup\{i\})$。对于项集$\{A\cup P\}$和$\{A\cup P\cup\{i\}\}$,只需要进行一次计数。

产生测试类算法的计数需要判断每一个候选项集在每一个事务频繁项投影中是否存在。设候选项集总数为n,事务数为m,则总共需要$n*m$次判断。虽然可以采用上述策略加快计数速度,但D中事务数总是不变的,候选项集的数量对该类算法的整体性能有着决定性的影响。

3.2　模式增长类算法

模式增长类算法通过DFS方式探索频繁项的枚举树,通过构造条件数据库实现项集的计数。设有前缀项集p,其条件数据库为D_p(由数据集中所有包含项集p的事务组成),找到D_p中i个频繁项$\{x_1,x_2,\cdots,x_i\}$,可以直接得到i个频繁项集$p\cup\{x_1\},p\cup\{x_2\},\cdots,p\cup\{x_i\}$。最经典的模式增长算法是Han等[14]提出的FPGrowth,使用一种前缀树FP-Tree来表示条件数据库。当挖掘的数据集十分稀疏时,事务之间的差异度极大,挖掘过程中会创建大量的条件FP-Tree,且条件FP-Tree中共享的前缀节点极少而节点很多,过大的条件FP-Tree构造成本使得经典的FPGrowth算法不适用于对海量稀疏数据进行挖掘[15,16]。针对稀疏数据,对经典FPGrowth算法的优化主要集中在降低条件数据库构造成本及提高遍历效率这2个方面。

Tan等[17]提出一种FPGrowth的优化算法FPGrowth#,其主要目标是提高构造条件FP-Tree的效率。在FPGrwoth算法中,构造项集$X\cup\{i\}$的条件频繁模式树$T_{X\cup\{i\}}$需要遍历2次项集$X\cup\{i\}$的条件数据库,第1次遍历找到条件数据库中所有的频繁项构造$T_{X\cup\{i\}}$的$Header_Table$,第2次遍历构造$T_{X\cup\{i\}}$。而算法FPGrowth#在构造T_X时使用一个数组A_X用于保存项集$X\cup\{j\}$的支持度,可以从数组中直接得到$\{j\}$中所有支持度大于$minsup$的项$\{i\}$,从而省去构造过程中的第1次遍历,相对于FPGrwoth提高了效率。

Pyun等[18]提出的算法进一步提高了条件数据库的构造速度。他提出了一种新的树结构LP-Tree(Linear Prefix Tree),项集使用一个多维数组来进行存储,减少了FP-Tree中节点之间的大量链接信息,加快了条件数据库的构造速度。由于数据是一种顺序结构可以进行直接访问,遍历LP-Tree的速度比通过链接遍历FP-Tree要快得多,进一步提高了构造条件数据库的效率。

Pei等[19]提出了一种表示条件数据库的数据结构H-Struct,并提出了在此结构上进行挖掘的HMine算法。H-Struct将条件数据库中的频繁项投影按固定顺序排序,使用链接实现投影之间的关系,在递归构造条件数据库时不用实际创建对应的FP-Tree而只需要修改投影的链接即可。实际上这是一种条件数据库的伪构造模式,用修改链接来取代了实际构造FP-Tree,提高了构造条件数据库的效率,同时也节约了大量的内存空间。

Guo等[20]提出了一种新的数据结构FPL(Frequent Pattern List)来表示稀疏的数据集。设有数据集D,i为其$FList$中的项,若i在事务T的频繁项投影T'中出现,则用1表示,否则用0表示,每一个事务均用一个长度为$|FList|$的位向量表示。使用事务的位字符串构造每一个频繁项的条件数据库,通过对位字符串的位计数和移位产生频繁模式。FPL这种数据结构由于采用了位向量来表示事务,对其进行计数和移位的效率很好,但由于数据集是稀疏的,代表每一个事务位向量中存在着大量0元素,存储效率比较低下。

上述优化策略中,HMine算法与其他算法有着本质的不同,将实际构造条件数据库的对应数据结构改为伪构造模式,只需要修改事务频繁项投影之间的指针即可,避免了递归构造大量条件数据库的计算成本,4.2节的实验结果表明,对于稀疏数据HMine算法在模式增长类算法中具有最佳的性能[19]。

3.3　垂直挖掘算法

垂直挖掘算法通过BFS方式探索频繁项的枚举树,通过对项集的$tidset$运行交运算直接得到项集的支持度。设有项集X和Y,它们的$tidset$分别为$\{1,2,3\}$和$\{2,3,4,5\}$,则可以直接得到$Support(X)=3$,$Support(Y)=4$,计算项集$\{X\cup Y\}$的支持度只需要将$\{1,2,3\}\cap\{2,3,4,5\}$得到$\{2,3\}$,则$Support(\{X\cup Y\})=2$。由于项集的支持度可以通过$tidset$的交运算直接得到,垂直挖掘算法在效率上一般比产生测试类算法要好。可采用$bitmap$表示项集的$tidset$[3],通过二进制运算与进一步加快$tidset$的交运算速度,提高了算法的效率。设有数据集$D=\{T_1,T_2,\cdots,T_n\}$,则项集X的$tidset$用$bitmap$表示为$\{b_1,b_2,\cdots,b_n\}(b_i=1,X\in T_i\mid b_i=0,X\notin T_i)$,项集的支持度等于$bitmap$中1的个数,2个项集$tidset$的交运算转换为将2个项集的$bitmap$按位与运算。由于计算机可直接执行二进制的与运算,因此极大提高了项集支持度计算的速度。

文献[21]中提出的BitTableFI算法与文献[3]的类似,采用$bitmap$表示$tidset$,大大提高了

通过 *tidset* 交集求支持度的速度。BitTableFI 采用 BFS 探索频繁项集枚举树,虽然候选项集的产生也通过使用 *bitmap* 加快了速度,但是仍然会产生海量候选项集,影响算法性能。Index-BitTableFI 算法[22]是对 BitTableFI 算法的进一步优化,通过在 BitTable 的垂直方向上引入 index-array,将候选项集的产生方式由 BFS 转为 DFS,从而提高算法性能。Matrix 算法[23]引入了 Matrix 作为剪枝的策略,进一步减少了 Index-BitTableFI 算法中候选项集的数量。

将 *tidset* 转换为 *bitmap* 可以将求集合的交集转换为位向量的按位求与,极大地提高了项集支持度的计算效率,但同时也带来了一个十分严重的问题——极大的内存需要。项集的 *bitmap* 中需要一个二进制位来表示该项集在对应的事务中是否存在,则每一个项集的 *tidset* 都需要 $|D|/8$ 个字节的存储空间,最大存储空间需求为 $2^{|FList|} * |D|/8$,在挖掘大数据时,内存空间的需求将变得无法忍受,使得基于 *bitmap* 的算法实际上不能用于挖掘任务。对于稀疏数据而言,项集 *bitmap* 中 0 的数量远远大于 1 的数量,也带来了很多无效的与操作。因此,需要进一步压缩 *bitmap* 存储空间以及避免无效与操作的问题。

文献[24]提出了一种将整个数据集对应的 BitTable 水平分割为若干个分区的方法,对每个分区进行编号,并使用一个 Hash 表来保存 *FList* 中的项所在的分区号。在计算项集支持度时,需要通过 Hash 表查询项集所出现的分区号,通过分区号找到项集的 *bitmap*。这种方法虽然通过水平划分 BitTable 降低了对一台计算机的内存需求量,但极大地增加了计算复杂度和成本。文献[25]中提出了一种方法,对每一个项集建立一个索引数组 *PBR*,用于指向 *bitmap* 中的有效区域(值为 1 的位),在计算支持度时只访问有效区域而跳过无效区域,加快与运算速度。文献[26]中提出了一种称为 Primal Block Encoding 的实用 *bitmap* 压缩方法,它将 *bitmap* 划分为若干个连续的块,每一块都用一个质数序列的幂集来表示,将 *bitmap* 的按位与运算转换为求对应幂的最大公约数,极大地减少了 *bitmap* 的存储空间,同时保证了较快的运算速度。

只从运算速度上来考虑,直接采用 *bitmap* 表示 *tidset* 的 Eclat 算法在此类算法中无疑具有最佳的效率,但如果挖掘海量稀疏数据,则内存的需求是不可忍受的。对 *bitmap* 进行各种压缩存储的策略可以节省相当部分存储空间,但对算法运行速度有一定的影响。

4 算法性能分析

实验平台配置为 Intel CPU i7-4790 3.60 GHz 8 核、内存 8 GB、操作系统为 Red Hat Enterprise Linux Server 6.6;采用 Java 语言实现有关算法,Java 平台为 Java 1.6;稀疏数据集采用 FIM 标准数据集[27]中的稀疏数据集 retail 和 T10I4D100K,2 个数据集的基本属性如表 1 所示。根据文献[6]的研究,数据集稀疏度与最小值尺度呈反比,retail 数据集的稀疏度要大于 T10I4D100K 数据集的。对具有代表性的稀疏数据 FIM 算法进行时空性能评价。

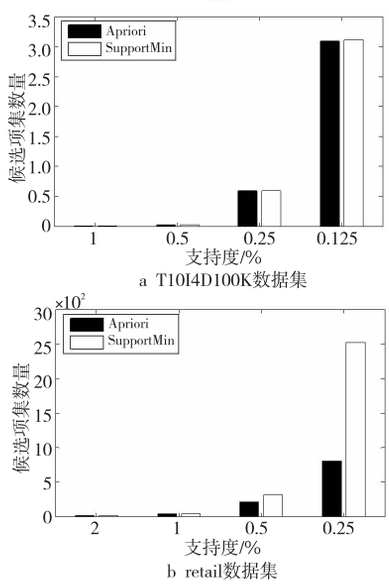
Table 1 Basic properties of the two test datasets

表 1 2 个测试数据集的基本属性

数据集	事务数	项数	平均事务长度	性质
retail	88 162	16 470	10	真实数据
T10I4D100K	100 000	870	10	合成数据

4.1 剪枝策略比较

稀疏数据集由于 $|FList|$ 数量极大,会产生海量的候选项集,对产生测试类 FIM 算法的性能带来极大影响。在 *minsup* 确定的情况下,稀疏数据的 *FList* 和 C_2 都是确定的,且任何候选项集剪枝策略都无法对 C_2 进行剪枝。实验选择 Apriori 和 SupportMin 2 种剪枝策略,比较它们在 2 个稀疏数据集不同支持度情况下剪枝后候选项集的数量(不包含 *FList* 和 C_2 的数量),结果如图 2 所示。



a T10I4D100K数据集

b retail数据集

Figure 2 Comparison of candidates on two test datasets using two pruning policies

图 2 2 个测试数据集上 2 种剪枝策略候选项集数量比较

从图 2 可以看到,在 T10I4D100K 上 2 种剪枝策略的效果相当,在稀疏度更高的 retail 数据集上,Apriori 剪枝策略的效果更好,但 2 种剪枝策略都需要 $O(n^2)$的时间复杂度,对算法性能有着严重的影响。对于稀疏数据集而言,候选 2-项集 C_2 在整个候选项集中占据绝对的比重,且远远大于频繁 2-项集 L_2 的数量。表 2 为 2 个测试数据集 C_2 占候选项集总数的比重。因此,加快候选 2-项集产生和计数可以大大提高产生测试类算法对稀疏数据挖掘的效率。

Table 2　Comparison of numbers of C_2 and L_2 on two sparse datasets

表 2　2 个稀疏数据集 C_2 和 L_2 数量比较

数据集	*minsup* %	候选项集总数	C_2	L_2
retail	2.000	211	190	22
	1.000	2 499	2 415	58
	0.500	25 758	24 310	237
	0.250	266 365	253 828	764
T10I4D100K	1.000	70 127	70 125	9
	0.500	162 183	161 596	342
	0.250	275 656	256 686	2 661
	0.125	411 670	302 253	6 926

可以直接使用文献[9]中提出的产生候选项集的方法直接产生频繁 2-项集 L_2,避免对海量候选 2-项集进行剪枝和计数。使用上述剪枝策略的 FastL2Apriori 算法与经典 Apriori 算法在 2 个数据集上的运行时间比较如图 3 所示。

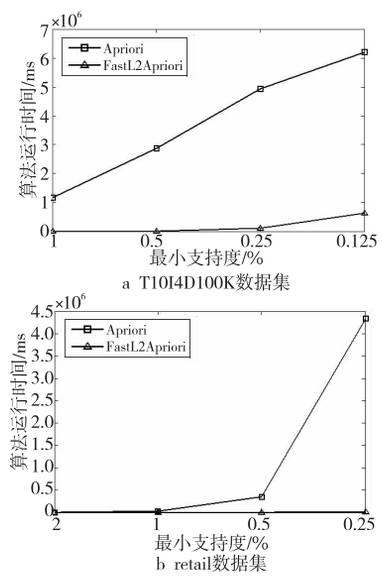


Figure 3　Running time comparison of the two algorithms on two test datasets

图 3　2 个测试数据集上 2 种算法运行时间比较

从图 3 可以看到,使用直接产生 L_2 剪枝策略的 FastL2Apriori 算法在运行时间上远远低于经典的 Apriori 算法,且数据集稀疏度越高,性能提升效果越好。

4.2　条件数据库构造方式比较

稀疏数据对模式增长类算法带来的主要挑战是产生海量的条件数据库,影响算法性能,因此优化策略主要从 2 个方面来考虑:(1)通过各种途径提高构造条件数据库的速度;(2)将实际构造改为伪构造,不实际构造条件数据库,通过修改事务频繁项投影之间的链接来实现构造条件数据库的目的。采用条件数据库实际构造策略的 FPGrowth 算法和伪构造策略的 HMine 算法在 2 个测试数据集上的运行时间如图 4 所示。

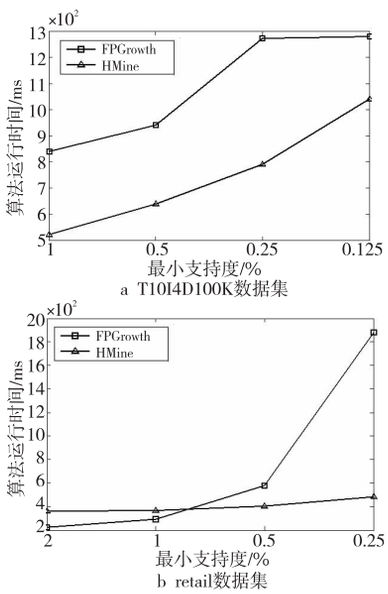


Figure 4　Performance comparison of the two pattern growth algorithms on two test datasets

图 4　2 个测试数据集上 2 种模式增长算法性能比较

从图 4 可以看到,对于稀疏数据而言,伪构造策略比实际构造策略的效率要高得多,且随着数据集稀疏度的不断增高,伪构造策略的性能优势更加明显。

4.3　*tidset* 和 *bitmap* 及跨类横向比较

垂直挖掘算法通过 BFS 方式产生候选项集,通过对项集 *tidset* 运行交运算直接得到项集的支持度,将 *tidset* 转换为 *bitmap* 可以进一步加快交运算计算的速度,但会带来较大的内存需求,可以进一步将 *bitmap* 转换为对应的整数序列来节省内存空间。选择使用 *tidset* 表示的 Eclat 算法 EclatIntergerSet、使用 *bitmap* 的 Eclat 算法 EclatBitSet 在 2 个测试稀疏数据集上运行,算法运行时间

如图5所示,所需内存空间如图6所示。由于产生测试类算法的性能要远远低于其他2类算法的,因此只加入模式增长类效率高的伪构造模式增长算法HMine进行横向比较。

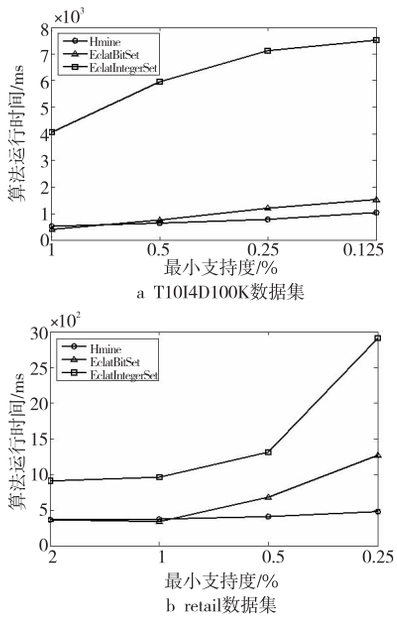


Figure 5 Performance comparison of the three algorithms on two test datasets

图5 2个测试数据集上3种算法的性能比较

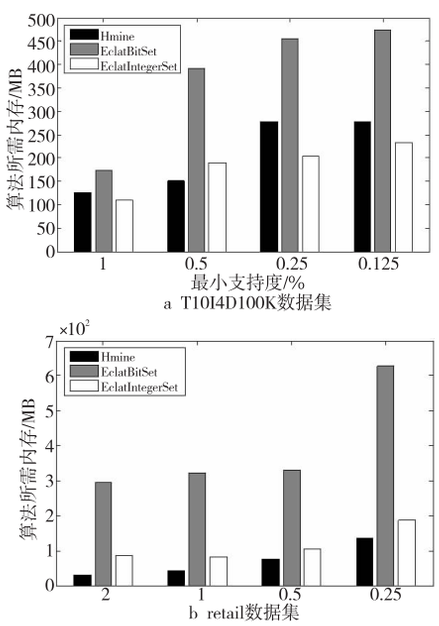


Figure 6 Memory requirements of the three algorithms on two test datasets

图6 2个测试数据集上3种算法的内存需求

从图5和图6可以看出,EclatBitSet算法的性能要比EclatIntegerSet的好很多,且数据集稀疏度越大,性能优势越明显。但是,使用BitSet表示*tidset*造成了更大的内存需求,数据集稀疏度越高,内存需求的增长也就越大。从横向比较的结果来看,无论从算法运行速度还是内存需求来说,对于稀疏数据的FIM任务,采用伪构造策略的模式增长方法是最好的选择。

5 结束语

稀疏数据集具有频繁项数量多、事务之间差异度大的特征,对传统FIM算法的性能带来了严峻的挑战。频繁项数量多的特点给采用BFS方式探索频繁项枚举树的产生-测试类、垂直挖掘类算法带来了极大的候选项集,事物之间差异度大使得采用诸如FP-Tree等前缀树的频繁模式增长类算法需要构造大量的条件数据结构,对FIM算法的性能产生了很大的影响。本文对已提出的针对稀疏数据集的FIM算法的优化策略进行了综述,对具有代表性的算法进行了实验评价。实验结果表明,在稀疏数据集中进行FIM挖掘,采用伪构造策略的频繁模式增长算法具有最佳的时间和空间性能。

参考文献:

[1] Agrawal R C, Imielinski T, Swami A N. Mining association rules between sets of items in large databases[C]// Proc of the ACM SIGMOD Conference on Management of Data, 1993:207-216.

[2] Agarwal R C, Aggarwal C C, Prasad V V V. Depth first generation of long patterns[C]// Proc of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000:108-118.

[3] Burdick D, Calimlim M, Flannick J, et al. MAFIA: A maximal frequent itemset algorithm[J]. IEEE Transactions on Knowledge & Data Engineering, 2005, 17(11):1490-1504.

[4] Gouda K, Zaki M J. Efficiently mining maximal frequent itemsets[C]// Proc of IEEE International Conference on Data Mining, 2001:2405-2409.

[5] Goethals B, Zaki M J. Advances in frequent itemset mining implementations: Report on FIMI'03[J]. ACM SIGKDD Explorations Newsletter, 2003, 6 (1):109-117.

[6] Xiao Wen, Hu Juan. Performance analysis of frequent itemsets mining algorithms based on sparseness of dataset [J]. Journal of Computer Applications, 2018, 38(4):995-1000. (in Chinese)

[7] Chen Z, Cai S, Song Q, et al. An improved Apriori algorithm based on pruning optimization and transaction reduction[C]// Proc of International Conference on Artificial Intelligence, Management Science and Electronic Commerce, 2011:1908-1911.

[8] Essalmi H, elFar M E, Mohajir M E, et al. A novel approach for mining frequent itemsets: AprioriMin[C]// Proc of IEEE International Colloquium on Information Science and Technology, 2017:286-289.

[9] Ye F Y, Wang J D, Shao B L. New algorithm for mining frequent itemsets in sparse database[C]// Proc of International Conference on Machine Learning and Cybernetics, 2005: 1554-1558.

[10] Liu Yong, Hu Yun-fa. Mining frequent patterns based on inverted list[C]// Proc of International Conference on Machine Learning and Cybernetics, 2009: 1320-1325.

[11] Park J S, Chen M S, Yu P S. An effective hash-based algorithm for mining association rules[J]. ACM Sigmod Record, 2016, 24(2): 175-186.

[12] Li L, Li Q, Wu Y, et al. Mining association rules based on deep pruning strategies[J]. Wireless Personal Communications, 2018, 102(3): 2157-2181.

[13] Deng Z H, Lv S L. PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning[J]. Expert Systems with Applications, 2015, 42(13): 5424-5432.

[14] Han J, Pei J, Yin Y, et al. Mining frequent patterns without candidate generation: A frequent-pattern tree approach [J]. Data Mining & Knowledge Discovery, 2015, 8(1): 53-87.

[15] Chen X, Li L, Ma Z, et al. F-miner: A new frequent itemsets mining algorithm[C]// Proc of IEEE International Conference on E-Business Engineering, 2006: 466-472.

[16] Rana D P, Mistry N J, Raghuwanshi M M. Memory boosting for pattern growth approach [C] // Proc of International Conference on Information Science, Electronics and Electrical Engineering, 2014: 1941-1945.

[17] Tan J, Bu Y, Yang B. An efficient frequent pattern mining algorithm[C]// Proc of the 2009 6th International Conference on Fuzzy Systems & Knowledge Discovery, 2009: 47-50.

[18] Pyun G, Yun U, Ryu K H. Efficient frequent pattern mining based on linear prefix tree[J]. Knowledge-Based Systems, 2014, 55: 125-139.

[19] Pei J, Han J, Lu H, et al. H-Mine: Hyper-structure mining of frequent patterns in large databases[C]// Proc of IEEE International Conference on Data Mining, 2002: 441-448.

[20] Guo H. A novel frequent item-set mining method based on frequent-pattern list [J]. International Journal of Digital Content Technology & Its Applications, 2011, 5(11): 182-188.

[21] Dong J, Han M. BitTableFI: An efficient mining frequent itemsets algorithm[J]. Knowledge-Based Systems, 2007, 20 (4): 329-335.

[22] Song W, Yang B, Xu Z. Index-BitTableFI: An improved algorithm for mining frequent itemsets[J]. Knowledge-Based Systems, 2008, 21(6): 507-513.

[23] Xu Z, Gu D, Wei S. An efficient matrix algorithm for mining frequent itemsets[C]// Proc of International Conference on Computational Intelligence and Software Engineering, 2009: 1-4.

[24] Fakhrahmad S M, Jahromi M Z, Sadreddini M H. Mining frequent itemsets in large data warehouses: A novel approach proposed for sparse data sets[C]// Proc of International Conference on Intelligent Data Engineering and Automated Learning, 2007: 517-526.

[25] Bashir S, Baig A R. Ramp: High performance frequent itemset mining with efficient bit-vector projection technique[C] // Proc of Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2006: 504-508.

[26] Gouda K, Hassaan M. Efficiently using prime-encoding for mining frequent itemsets in sparse data[J]. Computing & Informatics, 2013, 32(5): 1079-1099.

[27] IEEE computer society. Frequent itemset mining dataset repository[EB/OL]. [2018-04-21]. http://fimi.ua.ac.be/data/.

附中文参考文献:

[6] 肖文,胡娟.基于数据集稀疏度的频繁项集挖掘算法性能分析[J].计算机应用,2018,38(4):995-1000.

作者简介:

肖文(1984-),男,安徽歙县人,硕士,讲师,CCF会员(83513M),研究方向为数据挖掘和分布式计算。**E-mail**: cyees@163.com

XIAO Wen, born in 1984, MS, lecturer, CCF member (83513M), his research interests include data mining, and distributed computing.

胡娟(1985-),女,江苏海门人,硕士,讲师,研究方向为计算机软件应用。**E-mail**: hujuan19851985@163.com

HU Juan, born in 1985, MS, lecturer, her research interest includes computer software application.