基于频繁序列树的交互式序列模式挖掘算法

刘佳新
(燕山大学 图书馆,河北 秦皇岛 066004)

摘 要:为了减少在序列模式挖掘过程中由于重复运行挖掘算法而产生的时空消耗,提出了一种基于频繁序列树的交互式序列模式挖掘算法(ISPM)。ISPM算法采用频繁序列树作为序列存储结构,频繁序列树中存储数据库中满足频繁序列树支持度阈值的所有序列模式及其支持度信息。当支持度发生变化时,通过减少本次挖掘所要构造投影数据库的频繁项的数量来缩减投影数据库的规模,从而减少时空消耗。实验结果表明,ISPM算法在时间性能上优于PrefixSpan算法和IncSpan算法。

关键词:数据挖掘;序列模式;交互式挖掘;频繁序列树
中图分类号:TP311.131 **文献标识码**:A **文章编号**:1673-629X(2012)05-0064-03

An Interactive Sequential Patterns Mining Algorithm Based on Frequent Sequence Tree

LIU Jia-xin
(Library, Yanshan University, Qinhuangdao 066004, China)

Abstract: An interactive sequential patterns mining algorithm based on frequent sequence tree, called ISPM, is proposed in this paper in order to reduce the time and space consumption generated by repeatly running mining algorithm in the process of the sequential pattern mining. ISPM uses the frequent sequence tree as the storage sturcture of the algorithm. The frequent sequence tree stores all the sequential patterns with its support that meet the frequent sequence tree support threshold in the database. When the support is changed, ISPM can reduce the time and space consumption through by reducing the number of frequent items that construct the projected databases to reduce the size of the projected databases. Experiments show that ISPM outperforms PrefixSpan and IncSpan in time cost.

Key words: data mining; sequential patterns; interactive mining; frequent sequence tree

0 引 言

序列模式挖掘最早是由Agrawal和Srikant在1995年提出来的,是数据挖掘领域中的一个重要研究课题[1,2]。Agrawal和Srikant给出了AprioriAll, AprioriSome和DynamicSome三种算法。AprioriAll算法需要多次扫描数据库,并产生大量的候选序列,因此Srikant在文献[3]中提出了GSP算法,GSP引入了时间约束和滑动时间窗等技术提高了算法的性能。GSP算法需要扫描数据库多次,并且使用复杂的哈希结构,哈希结构有一定的缺陷。针对GSP算法的问题,Masseglia等提出了PSP算法,该算法采用前缀树结构解决了内存空间的浪费问题。基于Apriori的序列模式算法的缺点是需要产生巨大的候选序列,为了减少巨大的候选序列集,Pei[4]等人提出了一种通过模式增长进行挖掘序列模式的方法,称为PrefixSpan。PrefixSpan算法使用前缀投影减小了投影数据库的大小,在整个过程中不需要产生候选序列,大大减少了搜索空间。

目前序列模式挖掘方面的多数研究工作均在符合要求的最小支持度已知的假设下,集中于减小时间和空间消耗方面,没有考虑到需要重复挖掘的情况。在交互式序列模式挖掘研究领域,Parthasarathy[5]等人通过设置全过程最小支持度,重复执行挖掘算法来实现交互,但未能有效减少交互挖掘[6,7]的时间消耗。Lin[8]等人提出基于知识库的KISP算法,需多次扫描数据库,当数据库很大且最大序列模式长度很长时,算法时空消耗会很大。Lu[9]等人提出一种基于PrefixSpan的快速交互序列模式挖掘算法,通过建立序列模式库SPB,交互地挖掘不同最小支持度下的序列模式。

通常情况下,在挖掘过程中用户很难一次挖掘就得到满意的结果,因而需要以不同的参数值进行一系列的尝试挖掘。在某些条件下,这些挖掘任务甚至是

收稿日期:2011-10-10;**修回日期**:2012-01-15
基金项目:国家自然科学基金资助项目(61170190);秦皇岛市科学技术研究与发展计划项目(201001A018)
作者简介:刘佳新(1978-),女,吉林图们人,博士研究生,研究领域为数据挖掘。

很相似的。因此,设计有效的交互式序列模式挖掘方法,来充分利用先前的挖掘结果,避免总是从头进行重复挖掘,提高迭代挖掘过程的总体响应时间,是非常有意义的。

为了减少在序列模式挖掘过程中由于重复运行挖掘算法而产生的时空消耗,提出了一种交互式序列模式挖掘算法,称为 ISPM。ISPM 算法采用频繁序列树作为序列存储结构,频繁序列树中存储满足频繁序列树支持度阈值的所有序列模式及其支持度信息。ISPM 在第一次挖掘过程中,把所有满足最小支持度的序列模式及其支持度信息存储在频繁序列树中,其后的挖掘过程中,通过减少所要构造投影数据库[10~12]的频繁项的数量来缩减投影数据库的规模,从而减少时空消耗。

1　频繁序列树结构

频繁序列树是一棵前缀树,频繁序列树中存储数据库中满足频繁序列树支持度阈值的所有序列模式及其支持度信息。频繁序列树的构造过程与在数据库中使用 PrefixSpan 算法挖掘序列模式的过程是相似的。把每一次在投影数据库中挖掘出的所有频繁项作为孩子结点,插入到以投影数据库前缀的最后一项为父亲结点的频繁序列树中。下面给出频繁序列树的定义。

定义:频繁序列树的根结点包含一个属性,用于存储频繁序列树支持度阈值。除了根结点,频繁序列树中每个结点都包含 2 个属性,分别存储数据库中满足频繁序列树支持度阈值的序列模式及其支持度。从根结点的孩子结点到任何一个叶结点的路径都代表数据库中的一个序列模式,其支持度等于叶结点的支持度。频繁序列树中任何结点的支持度都不小于其子结点的支持度。

用一个实例来说明频繁序列树结构。简单起见,假设每一个元素中仅包含一个项,当元素包含多个项时,可以使用相似方法进行推论得到结果。序列数据库 DB 如表 1 所示。假设频繁序列树支持度阈值为 6。

表 1　序列数据库 DB

序列号	序列	序列号	序列
1	B C K E	6	F R K
2	B R C K	7	B R C J K
3	F B D R	8	B K C
4	R K	9	E J
5	B C R K	10	B R D C K

序列数据库 DB 的频繁序列树如图 1 所示。

由图 1 可以看出序列数据库 DB 中满足频繁序列树支持度阈值的序列模式及其支持度为{<B>:7, <C>:6, <K>:8, <R>:7, <BC>:6, <BK>:6, <RK>:6}。

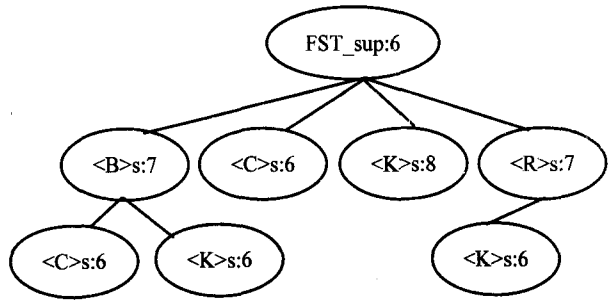

图 1　表 1 中序列数据库 DB 的频繁序列树

ISPM 算法在第一次挖掘过程中,把用户给定的最小支持度设为频繁序列树支持度阈值,在文献[13]中,给出了频繁序列树的构造算法 Con_FST,用 Con_FST 构造原有数据库的频繁序列树。由于频繁序列树中存储了序列数据库中满足频繁序列树支持度阈值的所有序列模式及其支持度信息,因此在最小支持度不小于频繁序列树支持度阈值的情况下,当支持度发生变化时,ISPM 无需对数据库进行重复挖掘,只需通过深度优先遍历频繁序列树就能得到满足支持度的所有序列模式。

2　基于频繁序列树的交互式序列模式挖掘算法

交互式挖掘的目的就是使每次挖掘过程耗时尽量少,从而使整个挖掘过程效率最高。为了减少在序列模式挖掘过程中由于重复运行挖掘算法而产生的时空消耗,提出了一种基于频繁序列树的交互式序列模式挖掘算法,称为 ISPM。ISPM 采用频繁序列树作为序列存储结构。

ISPM 是一种基于投影的交互式序列模式挖掘算法。其主要思想是通过把每次挖掘的序列模式信息存储在频繁序列树中,当下次挖掘时通过减少所要构造投影数据库的频繁项的数量来缩减投影数据库的规模,从而减少时空消耗。ISPM 算法在第一次挖掘过程中,把用户给定的最小支持度设为频繁序列树支持度阈值,构造频繁序列树,最后通过遍历频繁序列树来找到满足最小支持度的所有序列模式。在交互挖掘过程中,当用户指定的最小支持度小于频繁序列树支持度阈值时,有可能产生新的序列模式,而这些序列模式不在频繁序列树中。ISPM 算法需要对数据库构造投影数据库,找到支持度大于等于最小支持度且支持度小于频繁序列树支持度阈值的序列模式。然后,把得到的序列模式存储到频繁序列树中,并把频繁序列树支持度阈值设为用户指定的最小支持度。再通过遍历频繁序列树找到满足最小支持度的所有序列模式。

算法 1:ISPM (DB, min_sup, FST)

输入:原有数据库 DB,最小支持度 min_sup,频繁

序列树 FST。

输出:频繁序列树 FST,满足最小支持度的频繁序列集 FS。

1:If FST 为空

2:Con_FST(DB, min_sup, FST);

3:遍历 FST,找到满足最小支持度的 FS;

3:Else If FST 不为空

4:If min_sup >= FST_sup

5:遍历 FST,找到满足最小支持度的 FS;

6:Else If min_sup < FST_sup

7:对数据库构造投影数据库,找到支持度大于等于最小支持度,并且支持度小于频繁序列树支持度阈值的序列模式的集合 FS_Tem;

8:把 FS_Tem 存储到 FST 中;

9:FST_sup = min_sup;

10:遍历 FST,找到满足最小支持度的 FS;

11: Return;

频繁序列树结构在 ISPM 算法中的引用,使 ISPM 算法能够充分利用先前的挖掘结果。在交互挖掘过程中,当用户指定的最小支持度不小于频繁序列树支持度阈值时,ISPM 算法不需要对数据库进行重复挖掘,无需构造投影数据库,只需通过深度优先遍历频繁序列树就能得到满足最小支持度的所有序列模式;当用户指定的最小支持度小于频繁序列树支持度阈值时,ISPM 算法不需要对原有数据库构造投影数据库,只需要对局部的频繁项构造投影数据库,大大减小了投影数据库的规模。

3 实验结果和性能分析

实验中,对 ISPM、IncSpan 和 PrefixSpan 算法的性能进行比较。运行环境为 1.5GHz 奔腾 CPU、1G 内存,操作系统为 Windows XP。所有算法使用 Microsoft Visual Studio 2005 和 Microsoft SQL Server 2005 实现。采用的数据集是人工合成数据,用字母标记项目。项目集包含 26 个元素,数据集大小为 200K。

图 2 给出当支持度发生变化时,三种算法的运行时间。这里假设序列数据库 DB 的频繁序列树已经存在,并且频繁序列树支持度阈值为 10。从图中可以看出,当 min_sup = 0.4% 时,由于 PrefixSpan 算法和 IncSpan 算法需要构造大量的投影数据库,因此算法运行的时间较长。ISPM 算法不需要对原有数据库构造投影数据库,只需要对局部的频繁项构造投影数据库,大大减小了投影数据库的规模。因此,ISPM 算法运行速度比 PrefixSpan 算法和 IncSpan 算法快。当支持度变大时,由于 PrefixSpan 算法和 IncSpan 算法不具有交互式挖掘功能,需要重新构造投影数据库,但是由于投影数据库的规模大大减小,因此算法的运行速度有了明显的提高。ISPM 算法采用频繁序列树结构作为序列存储结构,频繁序列树中存储数据库中满足频繁序列树支持度阈值的所有序列模式,由于最小支持度大于频繁序列树支持度阈值,因此 ISPM 算法无需对数据库进行重新挖掘,只需要遍历频繁序列树就能得到所有的序列模式,算法的运行时间只需要几秒钟。因此,ISPM 算法的运行速度最快。

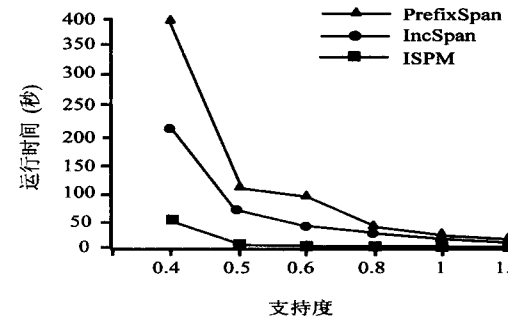

图 2 性能分析

4 结束语

文中提出了一种基于频繁序列树的交互式序列模式挖掘算法,称为 ISPM。ISPM 算法使用频繁序列树结构作为序列存储结构,在交互式挖掘过程中,ISPM 算法把满足最小支持度的所有序列模式都存储在频繁序列树中。频繁序列树结构在 ISPM 算法中的引用,使 ISPM 算法能够充分利用先前的挖掘结果。在交互挖掘过程中,当用户指定的最小支持度不小于频繁序列树支持度阈值时,ISPM 算法不需要对数据库进行重复挖掘,无需构造投影数据库,只需通过深度优先遍历频繁序列树就能得到满足最小支持度的所有序列模式;当用户指定的最小支持度小于频繁序列树支持度阈值时,ISPM 算法不需要对原有数据库构造投影数据库,只需要对局部的频繁项构造投影数据库,大大减小了投影数据库的规模。实验结果表明,ISPM 算法在时间性能上优于 PrefixSpan 算法和 IncSpan 算法。

参考文献:

[1] 陈 卓,杨炳儒,宋 威,等.序列模式挖掘综述[J].计算机应用研究,2008,25(7):1960-1963.

[2] 吴 楠,胡学钢.基于聚类分区的序列模式挖掘算法研究[J].计算机技术与发展,2010,20(6):109-112.

[3] Srikant R, Agrawal R. Mining sequential patterns: generalization and performance improvements[J]. Lecture Notes in Computer Science, 1996, 1057: 3-17.

[4] Pei J, Han Jiawei, Mortazavi-asl B, et al. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix - Projected Pattern Growth[C]//Proceedings of 17th International Conference on

(下转第 70 页)

在 3.3 节所述，这是因为在 DServices 架构中，优先选取离目前节点“最近的”节点执行查询，并且这个节点拥有最好的计算资源或最少的负载。

4.2 系统吞吐量分析

本节主要研究提交查询的客户端的数量与系统吞吐量的关系。这里吞吐量的定义为单位时间内整个网络内查询的数量。如图 3 所示，DServices 框架的系统吞吐量随着客户端数量的增多呈现指数性增加，当增长到该网络内所允许的饱和值时，额外的客户端不再影响其吞吐量，也就是新客户端不能获得任何资源。DServices 的吐吞量远大于无优化架构，这是由于在 DServices 系统中建立了响应块，这样就可以返回大量查询结果。

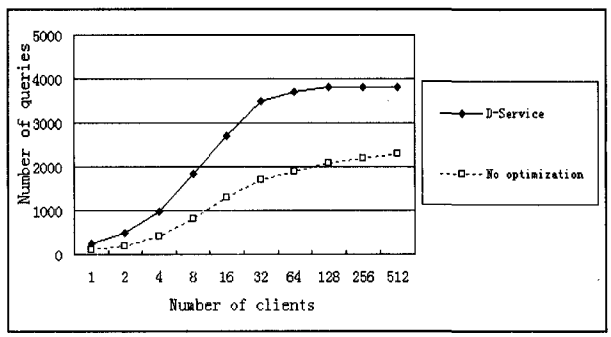

图 3 系统吞吐量比较曲线

5 结束语

在文中，为基于数据联邦服务的 P2P 提出了动态查询处理机制 DServices，它采用了动态查询执行引擎，这种引擎包含了可以适应动态网络和节点状况的数据查询基础设施。文中重点放在了查询处理引擎的扩展方面。最后进行了仿真实验，结果表明 DServices 系统在响应查询时间和系统吞吐量方面均具有高效性。

参考文献：

[1] 李 乔，郑 啸. 云计算研究现状综述[J]. 计算机科学，2011,38(4):32-37.

[2] 何 伟,李庆忠,郑永清,等. 社区云计算环境中的一种数据分布与搜索策略[J]. 计算机研究与发展,2010,47(S):407-413.

[3] 毛军鹏,崔艳莉,黄建华,等. 大规模应用 p2p 网络的特定资源搜索模型[J]. 计算机工程,2009,35(12):95-96.

[4] Kemper A, Wiesner C. HyperQueries: Dynamic Distributed Query Processing on the Internet[C]//VLDB Conference. [s. l.]:[s. n.],2001.

[5] Abiteboul S, Segoufin L. Active XML, Security and Access Control[D]. [s. l.]:Tel Aviv University,2001.

[6] Huebsch R, Hellerstein J M, Lanham N, et al. Querying the internet with pier[C]//VLDB. [s. l.]:[s. n.],2003.

[7] Kurzyniec D, Wrzosek T, Drzewiecki D, et al. Towards self-organizing distributed computing frameworks: the H2O approach [J]. Parallel Processing Letters,2003,13(2):273-290.

[8] Kossmann D. The state of the art in distributed query processing[J]. ACM Comput. Surv.,2000,32(4):422-469.

[9] Mortier R, Narayanan D, Donnelly A, et al. Seaweed: distributed scalable ad hoc querying[C]//ICDE Workshops. [s. l.]:[s. n.],2006.

[10] Pietzuch P R, Ledlie J, Shneidman J, et al. Network-aware operator placement for stream-processing systems[C]//ICDE. [s. l.]:[s. n.],2006.

[11] 汤克明,王创伟,陈 峻. P2P 模拟器的比较研究[J]. 微电子学与计算机,2008,25(9):105-108.

[12] 黄烟波,张 凌,张国华. 一种通用的 p2p 模拟器[J]. 计算机技术与发展,2008,18(10):120-122.

(上接第 66 页)

Data Engineering. Heidelberg: Institute of Electrical and Electronics Engineers Computer Society,2001:215-224.

[5] Parthasarathy S, Zaki M J, Ogihara M, et al. Incremental and interactive sequence mining[C]//Proceedings of 8th International Conference on Information and Knowledge Management. New York: ACM,1999:251-258.

[6] 任家东,宗俊省. 一种基于规则表达式约束的序列模式增量式挖掘算法[J]. 燕山大学学报,2007,31(5):402-409.

[7] 牛兴雯,杨冬青,唐世渭,等. OSAF-tree-可迭代的移动序列模式挖掘及增量更新方法[J]. 计算机研究与发展,2004,41(10):1760-1767.

[8] Lin M Y, Lee S Y. Improving the efficiency of interactive sequential pattern mining by incremental pattern discovery [C]//Proceedings of 36th Annual Hawaii International Conference on System Sciences. United States: IEEE Computer Society,2003:68-75.

[9] 陆介平,刘月波,倪巍伟,等. 基于 PrefixSpan 的快速交互序列模式挖掘算法[J]. 东南大学学报,2005,35(5):692-696.

[10] 汪林林,范 军. 基于 PrefixSpan 的序列模式挖掘改进算法[J]. 计算机工程,2009,35(23):56-58.

[11] 张 坤,朱扬勇. 无重复投影数据库扫描的序列模式挖掘算法[J]. 计算机研究与发展,2007,44(1):126-132.

[12] 陆介平,刘月波,倪巍伟,等. 基于投影数据库的序列模式挖掘增量式更新算法[J]. 东南大学学报,2006,36(3):457-462.

[13] Liu J X, Yan S T, Ren J D. The Design of Frequent Sequence Tree in Incremental Mining of Sequential Patterns[C]//Proceedings of 2th IEEE International Conference on Software Engineering and Service Science. United States: IEEE Computer Society,2011:679-682.