第 36 卷 第 14 期 计 算 机 工 程 2010 年 7 月
Vol.36 No.14 Computer Engineering July 2010
·软件技术与数据库· 文章编号：1000—3428(2010)14—0032—02 文献标识码：A 中图分类号：TP311.13

一种单遍扫描频繁模式树结构

谭 军[1,2]，卜英勇[2]，杨 勃[2]
(1. 中南林业科技大学计算机学院，长沙 412006；2. 中南大学机电工程学院，长沙 410083)

摘 要：针对频繁模式增长算法无法适应数据流的无限性和流动性的特点，提出一种新颖的 FP-tree 的变形结构-SP-tree，只需单遍扫描便能容纳全部数据库信息。为使 SP-tree 具有与 FP-tree 一样良好的压缩性能，给出一种有效的动态重构树的方法，称为宽度排序方法，该方法能够在挖掘过程中动态地逐条分支地重构树，最终产生一棵频繁递减的前缀树。实验结果表明，SP-tree 的压缩性能优于其他单遍扫描的前缀树结构。

关键词：数据流；频繁模式增长算法；单遍扫描模式树；宽度排序方法

Single-pass Frequent Pattern Tree Structure

TAN Jun[1,2], BU Ying-yong[2], YANG Bo[2]
(1. College of Computer Science, Central South University of Forestry and Technology, Changsha 412006;
2. College of Mechanical Electrical Engineering, Central South University, Changsha 410083)

【Abstract】 Aiming at the problem that FP-growth algorithm requires two database scans, which are not consistent with efficient data stream processing, this paper presents a novel tree structure which is a variation of FP-tree, called SP-tree, which captures database information with one scan. For making SP-tree have the same compact performance, it presents an efficient dynamic tree restructuring method, called the breadth sorting method, which restructures a frequency-descending prefix-tree branch-by-branch. Experimental results show that compact performance of the SP-tree is better than other prefix-tree structure with one scan.

【Key words】 data stream; FP-growth algorithm; single-pass pattern tree; breadth sorting method

1 概述

近年来，从数据流上发现频繁模式已成为一个具有挑战性的课题。FP-growth[1]是一种高效的经典算法，然而 2 遍数据库扫描不能适应数据流的无限性和流动性的特点。为了克服该局限性，需要设计一种有效的全局数据结构，通过单遍扫描就能包含所有数据库信息。文献[2-4]提出了几种前缀树数据结构，只需单遍扫描就能把全部数据库信息存储在一棵树结构上，其中，CanTree[4]结构最具代表性。由于按照频繁无关的顺序插入项到树结构中，因此 CanTree 的压缩性能和挖掘性能比 FP-tree 要差。

本文提出一种新颖的类似于 FP-tree 的单遍扫描前缀树结构，称为 SP-tree，其主要思想是按频繁递减的项目顺序定期重构树，通过反复重构，使 SP-tree 具有尽可能多的前缀共享，提供更好的压缩性能。

2 SP-tree 的构造

SP-tree 的构造有 2 个关键过程：(1)插入过程：扫描事务，根据项目表 *I*-list 的项目顺序逐一将它们插入到树，并更新 *I*-list 中项目的频繁计数。(2)重构过程：按频繁递减的项目顺序重新安排 *I*-list，根据最新的 *I*-list 重构树。首先执行插入过程，根据定义好的顺序将数据库中的事务逐条插入到初始为空的 SP-tree 中。然后以交互的方式动态地执行重构和插入这 2 个过程。最后再执行一次重构过程。

举例说明 SP-tree 的构造。图 1 为一个事务数据库 DB，为简单起见，假定每插入 3 个事务就重构树。SP-tree 初始为空，*I* 和 I_{sort} 分别表示频繁无关和按频繁递减的项目表(图 2)。首先，执行插入阶段，插入前 3 个事务后树结构如图 3 所示。然后，执行重构阶段。先把 *I*-list 的项目顺序按频繁递减的顺序重新排列，再根据新的 *I*-list 重构树，重构后的树如图 4 所示。接着，按 *I*-list 的项目顺序执行插入阶段，插入后的树如图 5 所示。插入 3 个事务后，又执行重构阶段，重构后的树如图 6 所示。如此反复直到所有的事务都插入到 SP-tree 中。

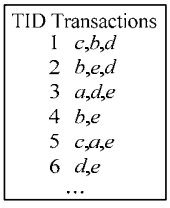

图 1 事务数据集

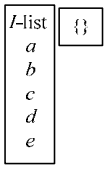

图 2 初始 SP-tree

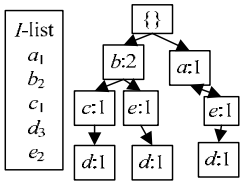

图 3 插入 TID1-3 后的 SP-tree

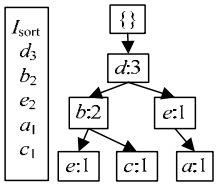

图 4 重构 TID1-3 后的 SP-tree

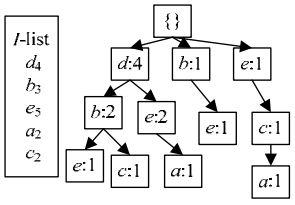

图 5 插入 TID4-6 后的 SP-tree

基金项目：国家自然科学基金资助项目“深海钻结壳微地形检测技术及最佳采集深度建模研究”(50474052)

作者简介：谭 军(1971－)，男，讲师、博士研究生，主研方向：数据库技术，数据挖掘；卜英勇，教授、博士生导师；杨 勃，讲师、博士研究生

收稿日期：2009-12-30 **E-mail：**tanjun007@vip.sina.com

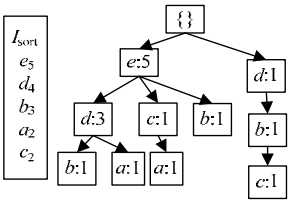

图 6　重构 TID4-6 后的 SP-tree

3　动态重构树的方法——宽度排序法(BSM)

影响SP构造性能的一个关键因素是重构SP-tree的方法。一种有效的重构机制对于减少整个构造代价是一个很重要的因素。本文提出一种新的技术，称为宽度排序方法，简记 BSM。

BSM 的基本思想：首先按频繁递减顺序重新安排 I-list 的项目，得到 I_{sort}，然后执行重构操作。从 T 的根节点逐条分支地执行重构过程。以根节点的孩子节点为根节点的子树就是一条分支。每条分支由几条路径构成。当重构一条分支时，所有路径的节点要重新排序。排序的方法是先把路径从树移开，放到一个临时数组，按 I_{sort} 的顺序重新安排路径的节点顺序，再将它插入到树。当处理一条路径时，如果已排好序，就不处理。当所有分支都处理完，重构过程就完成。

定义 1(排序的路径) 假设 P_1：如果所有项都按 S_{ort} 的项目顺序排列，则称 P_1 为已排序的路径。

定义 2(分支节点) 假设 a 是树 T 的任意节点，a_{lest} 指的是它的孩子列表。假设 $size(a_{lest})$是孩子列表的大小。节点 a 被定义为一个分支节点，如果 $size(a_{lest})>1$。

推论 假设 P_1:{a_1, a_2,⋯,a_k, a_{k+1},⋯,a_n}是树 T 的路径，a_1 和 a_n是根节点的孩子节点和叶子节点，a_k是分支节点，P_2: {a_1, a_2,⋯,a_l,a_{l+1},⋯,a_m}是一条路径，和 P_1 共享前缀{a_1,a_2,⋯,a_k}(k=l)，如果 P_1 是排序的路径，则 P_2 的子路径{a_1,a_2,⋯,a_l}也是排序的当且仅当在 a_{l+1} 和 a_m 之间没有比 a_l 更频繁的项。

证明：因为 P_1 是排序的路径，所以 a_1,a_2,⋯,a_k 是按频繁递减的顺序排列。如果在 a_{l+1} 和 a_m 之间有比 a_l 更频繁的项，那么它的位置应该在根节点和 a_l 之间。因此，子路径{a_1,a_2,⋯,a_l}没有排序。

基于以上的定义和推论，本文提出 BSM 算法：

输入 T, I

输出 T_{sort}, I_{sort}

```
1  按频繁递减顺序采用合并排序技术将 I 转换成 I_sort
2 For  树 T 的每个分支 B_i
3 For B_i 的每条未处理的路径 P_j
4   If P_j  已排序
5     Process_Branch(P_j)
6   Else Sort_Path(P_j)
7  当所有分支都已排序，输出 T_sort, I_sort
8 Process_Branch(P){
9 For  从 leaf_p 节点的 P 的每个分支节点 n_b
10 For  从 leaf_k  到 n_b 的每条子路径，k≠p
11 If  在 n_b 和 leaf_k 之间的所有节点的项目排位比 n_k 更高
12     P=从 n_b 到 leaf_k 的子路径
13     If P 已排序
14       Process_Branch(P)
15   Else P=从根节点到 leaf_k
16        Sort_Path(P)}
```

算法过程：首先按 I 的频繁递减顺序构造 I_{sort}(第 1 行)，然后逐一检查树 T 的分支，是否有没排序的路径(第 2 行~第 4 行)。如果有，则对该路径排序(第 6 行)并将它再插入到树 T。当没有分支被处理时，树 T 的重构就完成了(第 7 行)。

BSM 的一个重要特点是处理含有排序路径的分支。在重构树过程中，如果任何路径被发现已排序(第 4 行)，不但省略排序操作，而且把路径的状态信息传播给在整个分支范围内该路径上的所有分支节点。因此，当在同样分支上排序其他路径时，只需检查从叶子节点到已排序路径的分支节点之间的子路径，看是否有比分支节点更频繁的项(第 11 行)。如果有，根据推论，从叶子到根节点的整个路径将重新排序(第 16 行)。当所有子路径都已调整(第 10 行)时，就处理下一个分支。

4　实验分析

实验分析采用 5 个数据集，包括 1 个仿真数据集和 4 个真实数据集。仿真数据集是 T100I20N1KP5KC0.25D200K。4 个真实数据集分别是 chess, connect-4, mushroom 和 pumsb*。实验环境：Windows 2003 Server 操作系统，CPU 为 Intel 2.8 GHz，内存 1 GB。算法用 C++实现。

本文比较了 SP-tree 和 CanTree 的压缩性能。因为 CanTree 的性能与事务中项目的顺序有很大关系，所以分别比较了 SP-tree 和 CanTree 的 3 种版本：字母顺序(CT_I)，字母反向顺序(CT_R)，项目出现顺序(CT_A)。图 7 和图 8 分别显示在 3 个大数据集(T100I20N1KP5KC0.25D200K, pumsb*, connect-4)和 2 个小数据集(mushroom, chess)上 SP-tree 和 CanTree 的 3 种版本的内存比较结果。

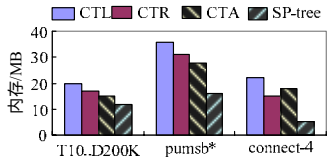

图 7　大数据集上的内存比较

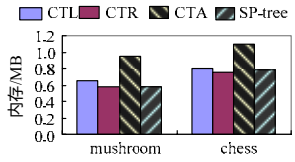

图 8　小数据集上的内存比较

结果表明 CanTree 大小的变化主要取决于数据集上数据的分布和项目的顺序。但是，SP-tree 的大小与这些参数无关，在大多数情况下，比 CanTree 的 3 种版本都要小得多。在 T100I20N1KP5KC0.25D200K，所有树需要的内存空间差不多，但 SP-tree 所需最小。然而，在真实数据集 pumsb*, connect-4，与其他数据集相比，SP-tree 取得更好的压缩性。在 chess 数据集，CT_R 的大小比 SP-tree 稍微小些。这说明并不能保证在所有情况下频繁递减的树结构的压缩性能都是最好的。

5　结束语

通过动态应用一种新的树重构方法得到频繁递减的单遍扫描前缀树-SP-tree，减少了挖掘时间并且具有良好的压缩性能。如何在数据流上基于滑动窗口模型重构 SP-tree，有待进一步研究。

参考文献

[1] Han Jiawei, Pei Jian, Yin Yiwei. Mining Frequent Patterns Without Candidate Generation[C]//Proc. of the ACM SIGMOD Conference on Management of Date. New York, USA: ACM Press, 2000: 1-12.

[2] Cheung W, Za O R. Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint[C]//Proc. of the 10th International Database Engineering and Applications Symposium. Alberta, Canada: ACM Press, 2003: 345-351.

[3] Lee A J T, Wang C S. An Efficient Algorithm for Mining Frequent Inter-transaction Patterns[J]. Information Sciences, 2007, 177(1): 3453-3476.

[4] Leung C K, Khan Q I, Li Z. A Canonical-order Tree for Incremental Frequent-pattern Mining[J]. Knowledge and Information Systems, 2007, 11(3): 287-311.

编辑　索书志