第35卷 第23期 计 算 机 工 程 2009年12月
Vol.35 No.23 Computer Engineering December 2009
·软件技术与数据库· 文章编号：1000—3428(2009)23—0056—03 文献标识码：A 中图分类号：TP311

基于 PrefixSpan 的序列模式挖掘改进算法

汪林林[1,2]，范 军[1]
(1. 重庆邮电大学计算机科学与技术学院，重庆 400065；2. 重庆工学院，重庆 400050)

摘 要：针对序列模式挖掘算法 PrefixSpan 在挖掘过程中需要构造大量投影数据库的不足，提出 IPMSP 算法，在递归挖掘过程中，通过检查序列数据库关于前缀的前缀，避免对同一频繁前缀模式构造重复投影数据库，同时舍弃对非频繁项的存储并在投影序列数小于最小支持度时停止扫描投影数据库，从而提高 PrefixSpan 算法的时空性能。实验结果证明，IPMSP 算法在时间和空间性能上优于 PrefixSpan 算法。
关键词：序列模式；PrefixSpan 算法；投影数据库

Improved Algorithm for Sequential Pattern Mining Based on PrefixSpan

WANG Lin-lin[1,2], FAN Jun[1]
(1. College of Computer Science & Technology, Chongqing University of Post & Tele., Chongqing 400065;
2. Chongqing Institute of Technology, Chongqing 400050)

【Abstract】 Aiming at the PrefixSpan algorithm produce huge amount of project databases in mining sequence patterns, this paper proposes an Improved PrefixSpan algorithm for Mining Sequential Patterns(IPMSP) alaorithm. By avoid produce duplicated project databases with the same prefix pattern through checking the prefix with regard to prefix of the sequence database and abnegating the non-frequent items and project databases which sequential number is lower than minimum support in the recursive mining process, the performance of PrefixSpan is well improved. Experiment results shows that the time and space performance of IPMSP algorithm are better than that of PrefixSpan.
【Key words】 sequential pattern; PrefixSpan algorithm; project database

1 概述

序列模式挖掘作为重要的 KDD 研究课题有着十分广泛的应用前景。目前已提出许多序列模式挖掘算法，文献[1]提出基于 Apriori 特性和逐层的发现方法，包括 AprioriAll, AprioriSome, DynamicSome 3 种算法以及后来提出的泛化序列模式挖掘算法 GSP[2]；文献[3]提出了使用垂直数据格式的 SPADE 算法；文献[4-5]提出了一种称为基于序列模式增长的方法，包括 FreeSpan, PrefixSpan 算法，其中 PrefixSpan 算法因包含更少的投影库和子序列连接要比 FreeSpan 算法性能更优。目前而言，在需要对大型的序列数据库做序列模式挖掘时，Prefixspan 算法以其在性能及效率等方面的优势作为首选考虑。

Prefixspan 算法投影数据库的规模小于原始数据库，然而构造投影数据库的开销巨大，递归地构建大量投影数据库为该方法的主要开销。此外该算法在挖掘过程中需要多次反复扫描投影数据库也降低了算法的执行效率。针对 PrefixSpan 算法的不足，本文提出的 IPMSP(Improved PrefixSpan algorithm for Mining Sequential Patterns)算法主要对其进行以下 2 个方面的改进：(1)在构建投影数据库时，通过舍弃对非频繁项的存储以及对投影序列数小于最小支持度的投影数据库的扫描，减少不必要的存储空间和扫描不可能出现频繁序列的投影数据库的时间；(2)相对于序列模式挖掘中后缀的定义提出序列关于前缀的前缀概念，在递归挖掘过程中，通过检查序列数据库关于前缀的前缀，避免对投影数据库中同一频繁前缀重复投影以及对投影数据库的重复扫描，减少投影的次数和数量，从而缩短算法执行的时间，减小投影数据库占用的空间。

2 基于前缀投影的序列模式挖掘算法PrefixSpan

文献[5]提出了基于前缀投影的 PrefixSpan(Prefix-projected Sequential pattern mining)算法，该算法将前缀以及投影的概念引入到了序列模式挖掘领域。在介绍 PrefixSpan 算法之前，首先介绍前缀、后缀、投影等相关概念。

2.1 相关概念及术语

序列数据库 S 是元组 $<sid,s>$ 的集合，其中，sid 是序列 ID；s 是一个序列。如果 α 是序列 s 的子序列，则称元组 $<sid,s>$ 包含序列 α。序列 α 在序列数据库 S 中的支持度是指数据库中包含 α 的元组的个数，记作 $sup\,port(\alpha)$。给定一个正整数 min_sup 表示最小支持度阈值，序列 α 在数据库 S 中是频繁的，如果 $sup\,port(\alpha)\geqslant \min\_sup$。频繁序列叫做序列模式。一个长度为 l 的序列模式叫做 l 模式。

假设序列元素中的项目以字母序排列。序列 $\alpha=<e_1e_2\cdots e_n>$，序列 $\beta=<e'_1e'_2\ldots e'_m>$ $(m\leqslant n)$ 满足以下条件时，β 是 α 的前缀：(1) $e'_i=e_i$ $(i\leqslant m-1)$；(2) $e'_m\subseteq e_m$；(3) $e_m-e'_m$ 中的所有的项目排在 e'_m 中项目的后面。序列 $\gamma=<e''_m e_{m+1}\cdots e_n>$ 是 α 的关于前缀 β 的后缀，记作 $\gamma=\alpha/\beta$，其中，$e''_m=e_m-e'_m$，也记作 $\alpha=\beta\cdot\gamma$。

作者简介：汪林林(1945－)，男，教授，主研方向：数据挖掘，计算机网络，系统结构，图像处理；范 军，硕士研究生
收稿日期：2009-07-23 **E-mail：**fanjun1223@163.com

序列 α 和它的子序列 β，α 的子序列 α' 是其关于前缀 β 的投影当且仅当：(1) α' 以 β 为前缀；(2)不存在 α' 的真超序列 α'' 使得 α'' 也是 α 的以 β 为前缀的子序列。对序列数据库 S 中所有序列以序列 α 为前缀进行投影，得到的数据库叫做 S 的 α 投影数据库，记做 $S|_\alpha$ 。

2.2 PrefixSpan算法描述

模式增长是一种不需要产生候选的频繁模式挖掘方法，它的基本思想如下：先找出各个频繁项，然后产生投影数据库的集合，每个投影数据库关联一个频繁项，每个数据库进行单独挖掘。算法构造前缀模式，它与后缀模式相连得到频繁模式，从而避免生成候选。

Prefixspan 算法执行步骤描述如下：(1)对序列数据库扫描一次得到全部频繁项目 n，即所有长度为 1 的频繁序列集合；(2)将频繁序列完整的集合分为 n 个具有不同前缀的频繁序列的子集；(3)通过构造相应的投影数据库并在其中递归地挖掘发现频繁序列的子集。

下面以表 1 中的序列数据库 S 为例，给定最小支持度阈值 min_*sup*=2，用 Prefixspan 算法进行挖掘。

表 1　序列数据库

SID	序列
1	<*a*(*abc*)(*ac*)*d*>
2	<(*ad*)*c*(*bc*)*a*>
3	<*abdcb*>
4	<*acbc*>

S 中的序列模式用 Prefixspan 算法可以按照下面的步骤进行挖掘：

第 1 步 得到长度为 1 的序列模式。扫描 S 一次，找出所有的频繁项，每个频繁项都是一个长度为 1 的序列模式。找到的频繁项及其支持度为<*a*>:4, <*b*>:4, <*c*>:4, <*d*>:3。

第 2 步 划分搜索空间。序列模式的完全集可根据 4 个前缀划分成下面的 4 个子集：即前缀分别为<*a*>, <*b*>, <*c*>, <*d*>的子集。

第 3 步 找出序列模式的子集。在第 2 步中提到的序列模式的子集可以通过构建相应的投影数据库并递归地挖掘每一个来进行挖掘。投影数据库和序列模式如表 2 所示。

表 2　投影数据库和序列模式

前缀	投影数据库	序列模式
<*a*>	<(*abc*)(*ac*)*d*><(_*d*)*c*(*bc*)*a*> <*bdcb*><*cbc*>	<*a*><*aa*><*ab*><*ac*><*ad*> <*abd*><*aba*><*a*(*bc*)><*abc*> <*acb*><*aca*><*acc*><*a*(*bc*)*a*>
<*b*>	<(_*c*)(*ac*)*d*><(_*c*)*a*> <*dcb*><*c*>	<*b*><*ba*><*bc*><(*bc*)> <*bd*><(*bc*)*a*>
<*c*>	<(*ac*)*d*><(*bc*)*a*> <*b*><*bc*>	<*c*><*ca*><*cb*><*cc*>
<*d*>	<*c*(*bc*)*a*><*cb*>	<*d*><*db*><*dc*><*dcb*>

3　序列模式挖掘PrefixSpan改进算法

3.1　相关定义及定理

定义 给定序列 $\alpha=<e_1e_2\cdots e_n>$，序列 $\beta=<e'_1e'_2\ldots e'_m>$ $(m\leqslant n)$，β 是 α 的前缀：即满足 (1) $e'_i=e_i$ $(i\leqslant m-1)$；(2) $e'_m\subseteq e_m$；(3) $e_m-e'_m$ 中的所有的项目排在 e'_m 中项目的后面。序列 $\chi=<e_1e_2\cdots e'''_m>$ 是 α 的关于前缀 β 的前缀，其中，e'''_m 为 $e_m-e'_m$ 中所有项目排在 e'_m 中项目之前的项目。

例如：对于序列 *s*=<*a*(*abc*)(*ac*)*d*>，<*c*>是 *s* 的前缀，则<(*ac*)*d*>是 *s* 的关于前缀<*c*>的后缀，<*a*(*ab*_)>是 *s* 的关于前缀<*c*>的前缀。注意：在包含<*c*>的序列中，仅需要考虑以<*c*>的第 1 次出现为前缀的子序列。

设 α 是一个长度为 l 的序列模式，$\{\beta_1,\beta_2,\cdots,\beta_m\}$ 是所有以 α 为前缀的长度为 $(l+1)$ 的序列模式的集合。除 α 本身外，以 α 为前缀的序列模式的完全集可划分成 m 个不相交的子集，第 j 个子集 $(1\leqslant j\leqslant m)$ 是以 β_j 为前缀的序列模式集。

定理 设序列数据库 $S=\{s_1,s_2,\cdots,s_n\}$，α 和 β 为频繁模式。(1)对 $\forall s_i\in S$ 关于 β 的前缀 χ_i $(i=1,2,\cdots,n)$，$\alpha\subseteq\chi_i$ 并且 α 在 β 所属项集之前。(2)部分 $s_i\in S$ 关于 β 的前缀 χ_i，$\alpha\subseteq\chi_i$ 并且 α 在 β 所属项集之前，其余的所有 s_j 与 β 不构成前后缀关系，则序列数据库在 α 为前缀的序列模式中，以 β 为前缀的序列模式集与序列数据库中以 β 为前缀的序列模式集相同。

证明：当满足条件(1)时，序列数据库中对序列 α 与对序列 β 的支持数是相同的，同时在所有序列中 β 都出现在序列关于 α 的后缀中的单独项集中，当满足条件(2)时，序列数据库中对 β 的支持度有贡献的序列都在序列关于 α 的后缀中的单独项集中，由于原数据库和以 α 为前缀的投影数据库对 β 支持数以及 β 所在项集都没有改变，因此挖掘序列数据库中以 β 为前缀的序列模式就是挖掘以 α 为前缀的序列模式中的以 β 为前缀的序列模式。

例如：在表 1 所示的序列数据库 S 中，在前缀<*a*>的序列模式中以<*c*>为前缀的序列模式集为<*c*><*ca*><*cb*><*cc*>，和序列数据库 S 以<*c*>前缀的序列模式集是相同的。这是因为在序列数据库 S 中，所有序列关于前缀<*c*>的前缀<*a*(*ab*_)>, <(*ad*)>, <*abd*>, <*a*>中，<*a*>都出现在<*c*>所属项集之前。

3.2　序列模式挖掘PrefixSpan改进算法描述

从 2.2 节中介绍的 Prefixspan 算法中可以看到，虽然 PrefixSpan 投影数据库的规模小于原始数据库，然而构造投影数据库的开销巨大，多次扫描数据库也降低了算法执行效率。因此，本文提出基于 PrefixSpan 的序列模式挖掘改进算法 IPMSP(Improved PrefixSpan algorithm for Mining Sequential Patterns)，主要根据 3.1 节中的定理，在构建投影数据库时，通过检查序列数据库关于前缀的前缀，避免对投影数据库中同一频繁前缀重复投影，减少投影的次数和数量，并且节省重复投影数据库的构造时间和在投影数据库上挖掘模式浪费的时间。同时，在 PrefixSpan 算法执行过程中，由于在投影序列数小于最小支持数的投影数据库中，将不会出现超过最小支持数的频繁项，因此在 IPMSP 算法中只保存扫描投影数据库时得到的频繁项，通过比较投影数据库所包含的序列数和最小支持数，放弃挖掘序列数已小于最小支持数的投影数据库，提前退出对投影数据库的进一步挖掘，减少扫描不可能出现频繁序列的投影数据库的时间，从而提高算法执行效率。

改进 PrefixSpan 算法按照以下步骤进行挖掘：

第 1 步 得到长度为 1 的序列模式。扫描 S 一次，找出所有的频繁项，每个频繁项都是一个长度为 1 的序列模式，并将频繁项按其支持度从大到小排序。

第 2 步 根据频繁项支持度的大小，依次对频繁项进行投影，通过构建相应的投影数据库并递归地挖掘每一个来进行挖掘。

其中对每一个频繁项，构建相应的投影数据库，对投影数据库扫描一次，得到其局部频繁项。如果局部频繁项与第 1 步得到的除当前频繁前缀 α 以外的频繁项相同，假设为 β，对第 1 步中的 S 作关于 β 的前缀，根据 3.1 节中定理判断是否可以减少 β 的重复投影及其挖掘。如果满足 3.1 节中

定理条件，则只挖掘 β 为前缀的序列模式一次，即得到 β 为前缀的序列模式的同时可以得到 α 连接 β 序列模式。在找到原投影数据中对应于该频繁项的所有后缀集合后，保存该频繁项的投影数据库时，只保存在扫描原投影数据库时得到的支持数大于 min_*sup* 的项，若该频繁项的投影数据库所含序列数小于 min_*sup*，则结束在投影数据库中继续挖掘。

算法描述如下：

输入 序列数据库 *S* 和最小支持度 min_*sup*

输出 频繁序列的集合

方法：调用 IPMSP(<>, 0, *S*)

函数 IPMSP(α, *L*, $S|_\alpha$)，其中参数：α：一个序列模式；*L*：α 的长度；$S|_\alpha$：如果 $\alpha \neq$ <>则是 α 的投影数据库，否则是序列数据库 *S*，List：局部频繁项列表

方法：

(1)扫描 $S|_\alpha$ 找到频繁项，将频繁项按照支持度降序排列放入 List 中，标识未处理。

(2)对 List 中的每一个频繁项 *b*，把它连接到 α 形成频繁序列 α'，构造 α' 投影数据库 $S|_{\alpha'}$，扫描 $S|_{\alpha'}$ 得到频繁项列表 List’：

1)对在 List’, List 中的除 *b* 自身以外相同的项 b_i 做 $S|_\alpha$ 关于频繁项 b_i 的前缀，对满足 3.1 节中定理条件的频繁项 b_i，如果 List 中标识已处理，则输出 b_i 为前缀的序列模式 β_i，同时将 β_i 连接到 α 并输出；否则对 b_i 转 2)，得到序列模式 β_i 并标识已处理，输出序列模式 α 联接 β_i 以及 β_i。

2)对 List’其他的频繁项，调用 IPMSP(α', *L*+1, $S|_{\alpha'}$)。

3.3 实例说明

对表 1 中的序列数据库，可以得到前缀<*b*>, <*c*>, <*d*>的前缀如表 3 所示。注：如果 β 不是 α 的子序列，则 α 和 β 不存在前、后缀关系，在表中以“-”表示。如果 β 是 α 的前缀并且 α 的关于 β 的前缀为空，表中以 ∅ 表示。

表 3　序列关于前缀的前缀

	<*b*>	<*c*>	<*d*>
1	<*a*(*a*_)>	<*a*(*ab*_)>	<*a*(*abc*)(*ac*)>
2	<(*ad*)*c*>	<(*ad*)>	<(*a*_)>
3	<*a*>	<*abd*>	<*ab*>
4	<*ac*>	<*a*>	-

第 1 步 对 *S* 扫描得到长度为 1 的序列模式：<*a*>:4, <*b*>:4, <*c*>:4, <*d*>:3。

第 2 步 找出以<*a*>为前缀的序列模式。对<*a*>的投影数据库扫描一次，它的局部频繁项是 *a*:2, *b*:4, *c*:4 和 *d*:2。所有前缀为<*a*>的序列模式可划分成 4 个子集：前缀分别为<*aa*>, <*ab*>, <*ac*>, <*ad*>的子集。由于局部频繁项<*b*>与第 1 步得到的 1 模式<*b*>相同，在 *S* 中作序列关于<*b*>的前缀，如表 3 所示，满足 3.1 节中定理，因此只需对<*b*>为前缀的序列数据库做一次挖掘，将<*b*>为前缀的所有模式记录。在 *S* 中对<*b*>构建投影数据库，并挖掘得到模式<*b*>,<*ba*>,<*bc*>,<(*bc*)>, <*bd*>,<(*bc*)*a*>，此时可以直接得到前缀为<*ab*>的模式集为<*a*>连接<*b*>的模式，即<*ab*>, <*aba*>, <*abc*>, <*a*(*bc*)>, <*abd*>, <*a*(*bc*)*a*>。同理，可以得到 *c* 的模式<*c*>, <*ca*>, <*cb*>, <*cc*>后直接得到<*ac*>的模式<*ac*>, <*aca*>, <*acb*>, <*acc*>。子集<*aa*>的频繁项只有 *a*:2，得到模式<*aa*>。因为 *S* 中关于<*d*>的前缀(如表 3 所示)不满足 3.1 节中定理，所以单独对其进行挖掘。

第 3 步 发现以<*d*>为前缀的序列模式。对<*d*>的投影数据库挖掘得到模式<*d*><*db*><*dc*><*dcb*>。

4 实验分析对比

本文实验环境为 CPU Celeron 1.70 GHz，384 MB 内存，Windows XP 操作系统，算法代码采用 C++编写在 Visual C++ 6.0 环境下编译。实验测试数据用 IBM Quest Synthetic Data Generator 生成(产生过程同文献[1])，对数据集 C10T2.5S4I1.25 进行测试，测试数据集的有关参数为：|*D*|表示顾客数，设为 1 000；|*C*|表示顾客平均事务数，设为 10；|*T*|表示事务的平均项数，设为 2.5；|*N*|表示总项数，设为 10 000；|*S*|表示事务的平均数量，设为 4；|*I*|表示最长频繁序列的平均长度，设为 1.25。在数据库 C10T2.5S4I1.25 中，分别取最小支持度为 1%, 2%, 3%, 4%, 5%, 6%, 7%, 8%, 9%, 10%，运行 IPMSP 和 PrefixSpan 算法，对 2 个算法的时间和空间性能进行对比，实验结果如图 1 和图 2 所示。

从图 1 中可以看出，支持度比较低时，在时间性能上 IPMSP 具有明显优势，这是由于此时 PrefixSpan 在重复投影和重复扫描花费的了较多的时间，同时 IPMSP 算法在投影序列数小于最小支持度时，放弃投影数据库的生成及扫描也使得算法效率有所提高。在空间性能比较上(如图 2 所示)，由于 IPMSP 算法避免了重复投影数据库的构建以及放弃了对非频繁项的存储，从而使得 IPMSP 算法占据的内存空间小于 PrefixSpan 算法。实验证明，IPMSP 算法在各个支持度上的时空性能优于 PrefixSpan，提高了 PrefixSpan 的执行效率。

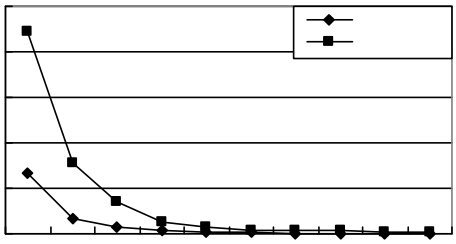

图 1　执行时间比较

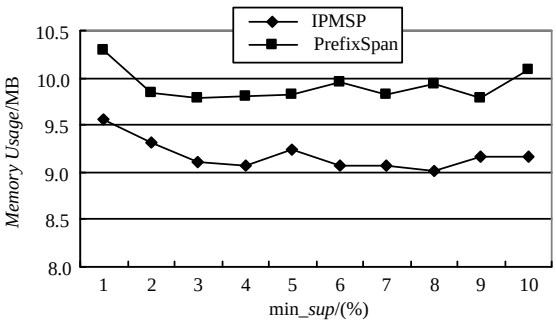

图 2　空间性能比较

5 结束语

序列模式挖掘是近几年越来越受关注的研究方向。本文在分析当前主要序列模式挖掘算法的基础上，重点研究了 PrefixSpan 算法，并提出 IPMSP 算法，通过构建投影数据库时舍弃非频繁项存储以及在投影序列数小于最小支持度时结束扫描投影数据库，同时避免序列数据库中重复投影数据库的产生以及对投影数据库进行的重复扫描 2 个方面对 PrefixSpan 作出改进，减少了投影的次数和数量，从而缩短

了算法的执行时间，减小了算法产生投影数据库占用的空间，提高了算法执行效率。分析和实验表明，IPMSP 算法在时空

（下转第 61 页）