计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):78−85 [doi: 10.15888/j.cnki.csa.006166]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于序列前缀技术的 XML 频繁路径挖掘算法①

张　洁, 毛国君

(中央财经大学 信息学院, 北京 100081)

摘　要: XML 文档是半结构化数据, 对其进行频繁路径挖掘可以分为两步: XML 文档序列化和序列挖掘阶段. 现有的序列化方式将 XML 文档表示为 Xpath 路径集合, 其中有大量的节点冗余; 序列挖掘阶段采用的类 Apriori 算法需要多次扫描数据库并产生大量的候选集, 采用的 PrefixSpan 算法会产生大量的投影数据库, 占用较大的内存. 针对以往 XML 频繁路径挖掘算法存在的不足, 本文提出一种高效的挖掘算法——基于序列前缀技术的 XML 频繁路径挖掘算法 (PXFP, Prefix-based XML Frequent Path Mining Algorithm). PXFP 算法以广度优先方式遍历 XML 文档树并将每个节点表示为“节点: 父节点”的形式, 这种序列化的方式减少了节点冗余. 在序列挖掘阶段借鉴 PrefixSpan 算法中前缀的概念, 但不产生投影数据库, 仅得到直接后缀 (即前缀的子节点), 通过记录频繁子路径的位置信息逐渐扩大频繁模式的长度, 位置信息的引入减少了对数据库的扫描. 实验结果表明, PXFP 算法取得了比 PrefixSpan 算法更高的时间和空间效率.

关键词: XML 频繁路径挖掘; 序列化; 位置信息; 前缀

引用格式: 张洁,毛国君.基于序列前缀技术的 XML 频繁路径挖掘算法.计算机系统应用,2018,27(1):78–85. http://www.c-s-a.org.cn/1003-3254/6166.html

Prefix-Based XML Frequent Path Mining Algorithm

ZHANG Jie, MAO Guo-Jun

(School of Information, Central University of Finance and Economics, Beijing 100081, China)

Abstract: XML documents are semi-structured data, and XML frequent path mining can be divided into two steps: XML document serialization and sequence mining. The existing serialization method expresses the XML document as a set of Xpath paths with a plenty of node redundancy. Algorithms based on Apriori require multiple scanning of the database and can generate a large number of candidate sets. The PrefixSpan algorithm generates a large number of projection databases, occupying a lot of memory space. In view of the shortcomings of the existing algorithms used in XML frequent path mining, this paper proposes an efficient mining algorithm called Prefix-based XML Frequent Path Mining Algorithm (PXFP). The PXFP algorithm traverses the XML document tree in a breadth-first manner and represents each node as “node: parent node”, which reduces the node redundancy. The PXFP does not generate the projection database, but only gets the sub-node of the prefix, and then increases the length of the frequent pattern by the position information of the frequent sub-path, which reduces scanning the database. The experimental results show that the PXFP algorithm achieves higher time and space efficiency than the PrefixSpan algorithm.

Key words: XML frequent path mining; serialization; location information; prefix

① 基金项目: 国家自然科学基金 (61273293)
收稿时间: 2017-04-09; 修改时间: 2017-05-09; 采用时间: 2017-05-16; csa 在线出版时间: 2017-12-22

XML 是可扩展标记语言 (eXtensible Markup Language) 的简称, 已经成为许多领域中数据交换、数据表示、数据存储的事实标准. 面对海量且不断产生的 XML 文档, 人们迫切需要从中发现有价值的信息和知识, 研究人员正在探索寻找合适的技术来解决与 XML 文档存储和分析相关的问题.

XML 数据挖掘可分为 XML 文档结构挖掘和 XML 文档内容挖掘. 结构挖掘的主要目标是挖掘 XML 模式, 可以分为结构内挖掘 (挖掘一篇 XML 文档内的结构) 和结构间挖掘 (挖掘多篇 XML 文档之间的结构). 挖掘出的频繁模式, 可在 XML 分类、XML 聚类、XML 索引、XML 数据存储、XML 数据压缩、XML 查询、XML 数据预测等多个 XML 相关领域获得应用. 例如: 通过用户访问模式挖掘改善网站架构; 将频繁查询作为结果缓存加快 XML 查询效率; 通过挖掘 XML 格式的订单数据进行消费者行为分析; 使用频繁路径作为特征向量来估量 XML 文档相似性从而用于 XML 文档分类、聚类. XML 文档的结构信息可以通过解析表示在标签序列中, 因此, 我们可以将 XML 文档树序列化, 借鉴序列挖掘算法对 XML 文档进行相应的频繁模式挖掘.

由于 XML 特殊的结构特点和表达含义, 对 XML 数据的挖掘具有更多的挑战性和创新性. 首先, XML 本质上是基于树结构的, 传统的关系型数据库的数据挖掘方法不能直接被利用; 其次, XML 的结构和文本内容一体化, 现有的方法还是以单独挖掘结构或者文本内容信息为主, 缺乏一体化的解决方案; 最后, 随着大数据时代的到来, 从因特网上获取 XML 文档成为主流应用, 因此找到代价 (如内存资源) 和精度平衡的解决方法成为重要任务之一. 本文提出一种 XML 文档序列的频繁路径挖掘算法 PXFP(PrefixSpan-based mining for XML Frequent Paths). 它通过对 XML 文档的序列化来形成基于前缀技术的序列模式挖掘工作, 期望获得更高的时间和空间效率.

1 相关工作

序列模式挖掘问题是最初由 Agarwal 等人[1]开创性提出, Agarwal 等人借鉴频繁项目挖掘算法提出了 AprioriAll 算法, 该算法基于先产生后测试的策略, 依据的原理是所有频繁序列的子序列也都是频繁的. 1996 年, Srikant 和 Agrawal 又提出了 AprioriAll 算法的扩展算法 GSP 算法[2], 它考虑了时间约束、滑动时间窗口和分类层次技术, 从而大大地减少了产生的候选序列的数量, 减少了时间和空间开销. 然而由于类 Apriori 算法自身存在的局限性, 学者们纷纷开始寻找更高效的方法. 2000 年, Han 等人提出了基于模式增长的 FP-growth 算法, 该算法不产生候选集, 而是通过 FP-tree 的树状结构压缩数据库, 然后再利用 FP-tree 从下向上挖掘频繁序列, 加快了挖掘过程[3]. 其后, 学者们又研究并提出了一系列基于前缀思想的算法, 包括 Han 和 Pei 于 2000 年提出的 FreeSpan 算法[4]和 2001 年提出的 PrefixSpan 算法[5], 其基本思想是: 递归地将当前挖掘的频繁序列作为前缀, 计算每一前缀的后缀, 生成投影数据库, 在每个投影数据库上进行子序列的增长. 这一过程将每一次的检验范围缩小到更小的投影数据库中, 降低了搜索代价. 这些都是经典的序列挖掘算法.

近年来, 在经典序列挖掘算法的基础上, 学者们相继提出了优化的序列挖掘算法. 2007 年, 张坤等人针对 PrefixSpan 算法存在大量重复投影数据库的问题, 提出一种名为 SPMDS 的算法[6]运用哈希值判断投影数据库是否重复, 通过建立索引加快了检索速度, 进而提高了算法的性能. 2009 年, 张利军等提出一种基于位置信息的序列模式挖掘算法 PVS[7], 在 PrefixSpan 算法的基础上, 通过记录每个已产生的投影数据库的位置信息降低投影数据库的冗余, 提高了算法效率. 2012 年, 刘栋等提出了基于 Map Reduce 的序列挖掘模式, 采用 Hadoop 分布式平台, 将 PrefixSpan 算法扩展到大数据集[8]. 2013 年, 吴信东等设计了一种带有通配符的模式挖掘算法, 通配符的加入使模式的形式更加灵活[9]; 2015 年, 刘端阳等首次在频繁序列模式挖掘算法中引入逻辑的思想, 提出了一种基于逻辑的频繁序列挖掘算法 LFSPM, 通过逻辑规则对中间结果进行过滤, 该算法较好地解决了支持度阈值的设定问题, 并提高了挖掘结果的可理解性[10]. 2015 年, Kaustubh Beedkar 等提出了用于挖掘具有层次结构的频繁序列的并行算法[11], 并将其设计为可以扩展到大数据集.

XML 频繁模式挖掘作为 XML 数据挖掘的重要研究方向之一, 也获得了许多学者的关注. 2005 年, Leung Ho-pong 等人提出的 PBClustering 算法将频繁路径作为特征向量来计算 XML 文档的相似度[12], 从而进行聚类. 其中频繁路径挖掘的具体方法是将每个 XML 文档树表示为 Xpath 路径集合, 然后利用 AprioriAll 序列挖

掘算法来挖掘 XML 文档集的频繁路径. 2008 年, 贝毅君提出了基于等价类的频繁标签序列挖掘算法 XSM[13], 该算法将 XML 序列构建成垂直数据库, 通过应用概念格理论、以不同前缀为基础将标签序列分成互不相交的前缀等价类, 通过合并等价类产生频繁标签序列, 递归地从等价类中挖掘频繁标签序列, 该算法在实验数据集中取得了较好的时间性能. 2012 年, 雷向欣等提出了数据流分页频繁子树挖掘模型 Tmlist[14], 对 XML 数据流进行分页, 管理跨页节点及频繁候选子树的跨页增长, 逐页挖掘频繁子树, 在可控误差范围内降低了空间消耗, 提高了挖掘效率. 2013 年, 李巍等提出了一种根据 XML 数据变化过程挖掘 XML 空间频繁变化结构 SFCS 的方法和发现 SFCS 的数据模型 SC-DOM[15], 实验证明了算法的有效性和可扩展性.

2 相关的定义

定义 1. XML 树模型. 一个 XML 文档的结构使用树 XT(r, V, E, L, f) 来表示. 其中, r 是树的根节点; V 是树的节点集合; E 是树的边集合, 每条边 $(v_1, v_2)\in E$ 表示节点 v_1 是节点 v_2 的父节点; L 表示标签集合; f 是从节点集合到标签集合的一个函数映射 f: V->L.

例子 1. 图 1 给出了一个 XML 文档对应的树结构, 其中: 它的根节点 $r=n_1$; 树的节点集合 $V=\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$; 边集合 $E=\{(n_1, n_2), (n_2, n_3), (n_2, n_4), (n_2, n_5), (n_3, n_6), (n_4, n_7), (n_5, n_8), (n_5, n_9)\}$; 树的标签集合 L={a, b, c, d, e, f, g}; 节点集合到标签集合的一个函数映射 f: $\{n_1$-> a, n_2-> b, n_3-> c, n_4-> e, n_5-> e, n_6-> d, n_7-> f, n_8-> f, n_9-> g$\}$.

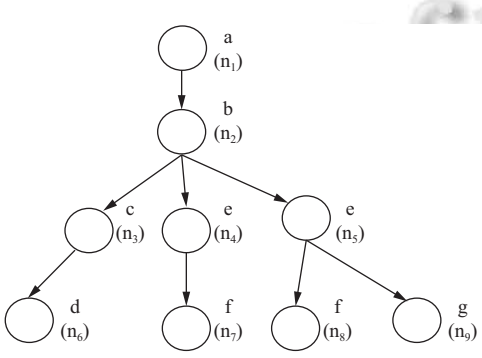

图 1 XML 文档树模型示例

定义 2. XML 树的标签序列表示. 对 XML 文档树进行广度优先遍历, 即按层序从左到右输出 XML 树的每个节点的标签, 得到 XML 树的标签序列被称为该 XML 树的标签序列表示.

例子 2. 图 1 中 XML 树的标签序列表示为<a, b, (c, e, e), (d, f, f, g)>. 注: 同层如果有多个标签用括号括起来, 如果只有一个标签, 括号可以省略.

定义 3. XML 树的标签路径. XML 树的一个标签序列被表示为 $A=<a_1, a_2, \ldots, a_n>$. 假如在 A 中 a_i 均是 a_{i+1} 的父节点 $(0<i<n)$, 则称序列 A 为该 XML 树的一条标签路径.

例子 3. 对应图 1 中 XML 树, <a, b, c, d>, <a, b, e, f>和<a, b, e, g>都是它的标签路径.

定义 4. 前缀路径. 对于路径 $A=<a_1, a_2,\ldots a_n>$和序列 $B=<b_1, b_2,\ldots b_m>$, $n\le m$, 满足 $a_1b_1, a_2b_2,\ldots, a_nb_n$, 则称 A 是 B 的一个前缀路径, 简称前缀.

例子 4. 对于标签序列数据 B=<a, b, (c, e, e), (d, f, f, g)>. 依据定义 4 可知: 路径 A=<a, b, c, d>是 B 的前缀, 但是 C=<a, c, d>不是 B 的前缀. 当然 B 的前缀不止一个, 比如<a>, <a b>, <a b c>, <a b e>, <a b e f>, <a b e g>也都是 B 的前缀.

定义 5. 直接后缀. 路径 $A=\{a_1, a_2,\ldots a_n\}$是序列 $B=\{b_1, b_2,\ldots b_m\}$的前缀, 当 $n<m$ 时, b_{n+1} 中 a_n 的子节点的集合被称为 A 在 B 上的直接后缀; 当 n=m 时, A 在 B 上的直接后缀为空.

例子 5. 在图 1 中, 令 A=<a, b, c>, B= <a, b, (c, e, e), (d, f, f, g)>, 则 A 在 B 上的直接后缀为<d>. 这是因为在图 1 中, <(d, f, f, g)>中只有 d 是 c 的子节点.

定义 6. 位置信息. 一个路径 $A=\{a_1, a_2,\ldots a_n\}$在一个 XML 树中的位置信息用一个 3 元组来表示, 记为 (s, v, m), 其中 s 表示 XML 标签序列数据库 S 中包含该路径的序列的序号; v 表示对应的序列中该路径的结束位置为 XML 树的第几层; m 表示该路径的结束位置在 XML 树该层的第几个. 当一个路径在数据库中多次出现时, 这个路径的位置信息表示为向量 $((s_1, v_1, m_1), (s_2, v_2, m_2), \ldots, (s_k, v_k, m_k))$.

例子 6. 对于图 1(假设数据库中只有图 1 所示一篇 XML 文档), 路径 A=<a, b, c>的位置信息可以表示为 (1, 3, 1); 路径 B=<a, b, e, f>的位置信息可分别表示为 (1, 4, 2), (1, 4, 3).

3 PXFP 算法

3.1 从 XML 文档集中挖掘频繁路径的基本步骤

XML 文档是半结构化数据, 对其进行频繁路径挖

掘可以分为两步: XML 文档序列化和序列挖掘, 基于此将从 XML 文档集中挖掘频繁路径的过程划分为五个阶段, 如图 2 所示.

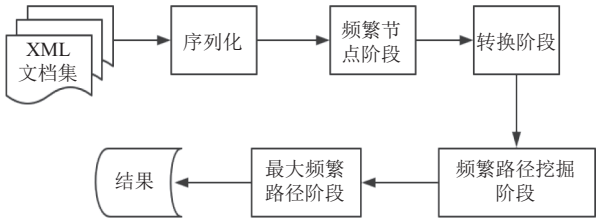

图 2 从 XML 文档集中挖掘频繁路径的基本步骤

(1) 序列化阶段: 依据 XML 文档的树形结构和定义 2 得到 XML 标签序列. 此外为了不遗漏 XML 文档树的结构信息, 将每个节点表示为“节点: 父节点”的形式 (根节点除外), 最终实现 XML 文档的序列化, 作为数据挖掘的格式化数据来用于分析.

(2) 频繁节点阶段: 找出支持度不小于阈值的频繁节点, 构造 Hash 映射表, 将频繁节点标签与数字一一对应.

(3) 转化阶段: 将频繁节点根据 Hash 表映射为数字同时去掉不频繁的节点, 得到转化后的序列数据库作为频繁路径挖掘阶段算法的输入.

(4) 频繁路径挖掘阶段: 本文将设计 PXFP 算法来完成这一工作. 算法的整体思想是: 首先扫描数据库来获取 1 阶频繁序列 L_1, 假设 L_1={1, 2, 3, 4}, 然后依次以 L_1 中的元素为前缀递归挖据频繁序列…深度优先直到不再有更长的前缀产生时, 再以 2 为前缀…以此类推, 直到遍历过 L_1 中所有元素后停止. 规范化的算法描述见 4.2 节.

(5) 最大频繁路径阶段: 去除频繁路径中包含于其他频繁路径中的子路径, 得到最大频繁路径.

下面以一个实例来介绍 XML 频繁路径的挖掘过程. 给定 3 个待挖掘的 XML 文档树, 如图 3 所示, 令支持度阀值为 0.5.

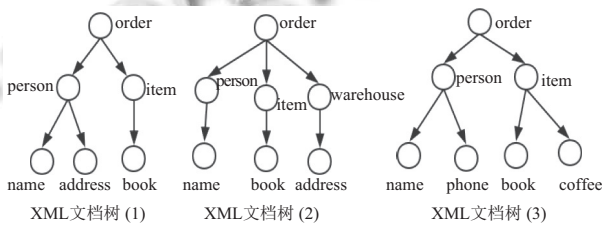

图 3 待挖掘的 XML 文档树

(1) 序列化阶段

对各个 XML 文档树进行层序遍历, 即按照广度优先的策略按层序从左到右输出 XML 树的每个节点. 如图 3 中的 XML 树 (1), order(第一层)(person, item) (第二层) (name, address, book) (第三层). 得到的 XML 标签序列集合如表 1 所示, 该集合表示出了 XML 树的层级关系.

表 1 XML 文档的序列化表示

XML文档编号	标签序列表示	序列化结果
1	<order, (person, item), (name, address, book)>	<order, (person:order, item:order), (name:person, address:person, book:item)>
2	<order, (person, item, warehouse), (name, book, address)>	<order, (person:order, item:order, warehouse:order), (name:person, book:item, address:warehouse)>
3	<order, (person, item), (name, phone, book, coffee)>	<order, (person:order, item:order), (name:person, phone:person, book:item, coffee:item)>

此外, 为了不遗漏 XML 文档树的结构信息, 需要将 XML 文档树的边, 即父子节点间的对应关系表示在序列中. 由于每一个节点都有且只有一个父节点 (根节点除外), 因此把每个节点表示为“节点: 父节点”的形式 (根节点除外). 例如: 对于 XML 树 (1), 其标签序列表示为<order, (person, item), (name, address, book)>, 序列化的结果为 <order, (person: order, item: order), (name: person, address: person, book: item)>. 可以用同样的方式处理另外两个 XML 文档树, 表 1 给出了序列化的结果.

(2) 频繁节点阶段

很显然, 一个频繁路径中的所有节点必须频繁, 所以发现频繁节点是挖掘频繁路径的基础性工作. 这个阶段相对比较简单, 只需要找出所有支持度不小于 50% 的节点即可. 这里的支持度是指出现该路径的文档数与总文档数的比值. 实际操作中, 将频繁节点映射成连续的整数, 结果如表 2 所示, 这样的映射纯粹是为了处理的方便和高效, 减少内存空间的占用.

表 2 支持度不小于 50% 的频繁节点

频繁节点	映射数值
order	1
person	2
item	3
name	4
address	5
book	6

(3) 转换阶段

根据前一阶段得到的频繁节点集, 构造 Hash 映射表, 将频繁节点的标签与数字一一对应, 将频繁节点根据 Hash 表映射为数字同时去掉不频繁的节点得到转换后的序列, 如表 3 所示. 这样做可以减少 XML 文档标签序列集的表示空间, 并加快后面的挖掘速度.

表 3 转换后的序列数据库

XML文档编号	序列化结果
1	<1, (2:1, 3:2), (4:2, 5:2, 6:3)>
2	<1, (2:1, 3:1) (4:2, 6:3, 5)>
3	<1, (2:1, 3:1) (4:2, 6:3)>

(4) 频繁路径挖掘阶段

将上一阶段得到的转换后的序列数据库作为频繁路径挖掘阶段算法的输入, 调用序列挖掘算法找出所有可能的频繁路径. 本文将设计 PXFP 算法来完成这一工作. 我们先通过本例来说明 PXFP 的基本思想, 规范化的算法描述将在 4.2 节来完成.

频繁路径挖掘阶段采用的算法是基于前缀的思想, 首先找到长度为 1 的前缀, 包括<1>, <2>, <3>, <4>, <5>, <6>, 我们需要对这 6 个前缀分别递归搜索各个前缀对应的频繁路径. 依据表 3 的序列数据库, 表 4 是长度为 1 的前缀对应的直接后缀.

表 4 长度为 1 的前缀的直接后缀

前缀	1	2	3	4	5	6
直接后缀	<(23)> <(23)> <(23)>	<(45)> <4> <4>	<6> <6> <6>	—	—	—

这里我们以前缀<1>为例来说明挖掘过程, 对其他前缀进行递归的方法和前缀<1>一样. 方法如下, 首先统计<1>的直接后缀的出现次数得到{2:3, 3:3}. 由于此时<2>, <3>均满足支持度阈值, 因此我们得到前缀为<1>的 2 阶频繁路径为<12>和<13>, 并记录下<12>的位置信息为 (1, 2, 1)(2, 2, 1)(3, 2, 1), <13>的位置信息为 (1, 2, 2)(2, 2, 2)(3, 2, 2). 接着我们分别递归<12>和<13>为前缀所对应的直接后缀. 首先看<12>前缀, 由<12>的位置信息找到其对应的直接后缀为<(45)>, <4>, <4>, 进行计数得到{4:3, 5:1}, 因此得到以<12>为前缀的 3 阶频繁路径为<124>, 并更新<124>的位置信息为 (1, 3, 1)(2, 3, 1)(3, 3, 1). 由<13>的位置信息找到其对应的直接后缀为<6>, <6>, <6>, 进行计数得到{6: 3}, 因此得到以<13>为前缀的 3 阶频繁路径为<136>, 并记录下<136>的位置信息为 (1, 3, 3)(2, 3, 2)(3, 3, 2). 继续递归以<124>和<136>为前缀的频繁路径. 由于前缀<124>和<136>对应的直接后缀为空, 因此不能产生 4 阶频繁路径. 至此以 1 为前缀的频繁路径挖掘结束, 产生的频繁路径为<1><12><13><124><136>. 同样的方法可以得到其他前缀对应的频繁路径. 频繁路径挖掘结果如表 5 所示.

表 5 频繁路径挖掘结果

1为前缀	2为前缀	3为前缀	4为前缀	5为前缀	6为前缀
<1> <12><13> <124><136>	<2> <24>	<3> <36>	<4>	<5>	<6>

(5) 最大频繁路径阶段

去除频繁路径中包含于其他频繁路径中的子路径, 得到最大频繁路径. 根据 Hash 表找到原始的最大频繁路径, 结果如表 6 所示.

表 6 最大频繁路径

最大频繁路径	原始最大频繁路径
<124><136>	<order, person, name><order, item, book>

3.2 算法描述

PXFP 算法是基于序列前缀的 XML 频繁路径挖掘算法, 其目标是挖掘 XML 序列数据库中满足支持度阀值的频繁路径. 下面我们对 PXFP 算法做一个归纳总结.

算法. PXEP 算法

输入: 序列数据库 S 和支持度阈值 αα

输出: 所有满足支持度要求的频繁路径集

1) 扫描序列数据库, 找到所有长度为 1 的序列模式 L_1, 作为初始的种子集;

2) 对于 L_1 中的每一个元素, 依次取出一个元素作为新候选模式的前缀, 以此来划分搜索空间;

3) 令前缀的长度 i=1;

4) 对于每个长度为 i 且满足支持度要求的前缀进行递归挖掘:

a) 找出前缀所对应的直接后缀, 并记录其位置信息. 如果直接后缀为空, 则递归返回.

b) 计算直接后缀中各项的支持度. 如果所有项的支持度小于阈值 αα, 则删除其位置信息并递归返回.
c) 将满足支持度阀值的各个单项和当前的前缀进行合并, 令 i=i+1, 得到若干新的前缀, 将其加入长度为 i 的序列模式 L_i 分别递归执行第 4) 步.
5) 重复 2)–4), 直到遍历完 L_1 中的所有序列.

值得注意的是, 与从单个 XML 文档中挖掘关键模式不同, 在多 XML 文档频繁路径的挖掘中, 如果某节点在前缀所对应的直接后缀中多次出现, 仍然只计数一次, 但位置信息都要记录下来.

3.3 算法伪代码

基于如上分析, PXFP 算法的伪代码描述如下.

```
算法. PXEP算法
输入: 序列数据库S; 最小支持度min-sup
输出: S的频繁路径KS
1. scan S to get L1; //扫描数据库找到所有的频繁1-序列
2. FOR m=0 TO L1.size()//依次遍历L1中的每个序列
3.    FOR father_node=L1.get(m) TO longest subsequence
   //由1阶前缀向高阶扩展
4.       FOR n=1 TO S.size();
      //在各个XML文档集标签序列中
5.       child_node[]=father.getchild();
      //(根据位置信息)找到子节点即直接后缀
6.       save or update Position of subsequence;
         //记录或更新位置信息
7.    Calculate support of every child_node occur in S;
   //计算子序列支持度
8.    IF(support >=min_sup)
   //大于等于最小支持度为频繁模式
9.    subsequence=father_node+child_node;
10.   add subsequence to KS;
11. ELSE
12.   delete Position;//小于最小支持度删除位置信息
13. Retrun KS //输出所有频繁路径
```

4 实验与分析

本部分实验 1 的数据来源于图 3, 实验 2 和实验 3 的数据来源于 INEX XML 挖掘竞赛 (XML Document Mining Challenge) 中结构挖掘部分的电影数据集[16], 这是从 11 个网站得到的 XML 文件.

实验的计算机采用的配置为: 4 GB 内存、英特尔酷睿 i3, 1.40 GHz 处理器、Windows 7 操作系统、Myeclipse 运行环境. 采用的对比算法是同样基于前缀的 PrefixSpan 算法. 实验的目标主要是检测 PXFP 算法的空间效率和时间效率.

实验 1. 比较内存空间使用情况

由于 PXFP 算法不需要产生投影数据库, 而只记录直接后缀的位置信息, 因此占用更少的内存. 实验 1 利用 PrefixSpan 算法和 PXFP 算法对图 3 中的 3 个 XML 文档序列化后得到的序列数据进行分析, 每一步的内存占用如表 7 所示. 由于两个算法都是基于前缀的分治思想, 因此表 7 中先讨论以 1 阶频繁序列为前缀的情况, 最后再从总体上进行讨论.

表 7 PrefixSpan 算法和 PXFP 算法的内存使用情况

前缀	PrefixSpan算法		PXFP算法	
	投影数据库个数	投影数据库内存	位置信息个数	位置信息内存
1-	8	51	12	36
2-	4	26	3	9
3-	3	9	3	9
4-	3	18	0	0
5-	2	5	0	0
6-	3	3	0	0
最大	8	51	12	36
合计	23	112	18	54

从表 7 的实验结果可以看出, 无论利用分治的思想在每一个小范围内讨论, 还是从总体上考虑最大内存使用和合计内存使用情况, PXFP 算法的空间效率都要好于 PrefixSpan 算法. 虽然 PXFP 算法需要记录较多的位置信息, 但由于每个位置信息仅占用 3 个字符的内存, 相比于 PrefixSpan 算法投影数据库中的子序列相比, 仍然有优势.

图 3 中 XML 文档的数量少, 并且每个 XML 文档的节点数少, 当应用于实际中的 XML 文档集时, PrefixSpan 算法将产生更多的投影数据库, 因而 PXFP 算法的空间优势将会更明显.

实验 2. 比较不同支持度阀值下的执行时间

利用实验数据集, 在两个支持度区间进行实验, 分别是: ①小支持度区间 (5%–30%); ②大支持度区间 (10%-90%), 对比本文提出的 PXFP 算法和 PrefixSpan 算法的运行时间. 表 8 给出的是实验数据在小支持度区间范围内挖掘出的频繁路径的个数, 图 4 对应给出在该区间内 PrefixSpan 算法和 PXFP 算法运行时间的对比; 表 9 给出的是实验数据在大支持度区间范围内挖掘出的频繁路径的个数, 图 5 对应给出在该区间内 PrefixSpan 算法和 PXFP 算法运行时间的对比.

由表 8 和图 4 可以看出, 在小支持度区间, 实验数据集产生较多的频繁路径, 随着支持度阀值增大, PXFP 算法和 PrefixSpan 算法的执行时间都在缩短, 原

因在于随着支持度阀值的增加, 生成的各阶频繁路径在减少, 并且在该数据集中, 频繁路径减少的速度很快. 同时也可以看出, PXFP 算法处理 XML 文档标签序列的时间效率明显高于 PrefixSpan 算法.

表 8 小支持度区间内挖掘出频繁路径个数

支持度阀值(%)	频繁路径个数
5	453
10	346
15	274
20	157
25	89
30	54

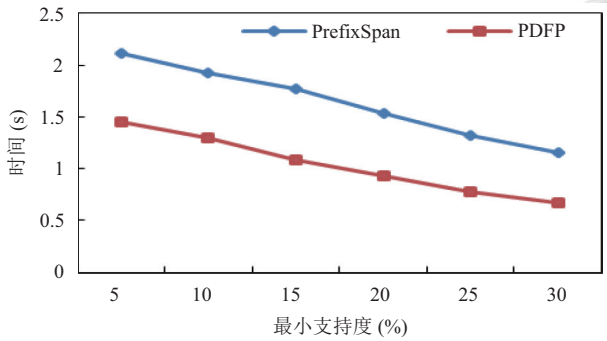

图 4 小支持度区间内算法时间效率对比

表 9 大支持度区间内挖掘出频繁序列个数

最小支持度(%)	频繁序列个数
10	346
20	157
30	54
40	28
50	17
60	8
70	3
80	0
90	0

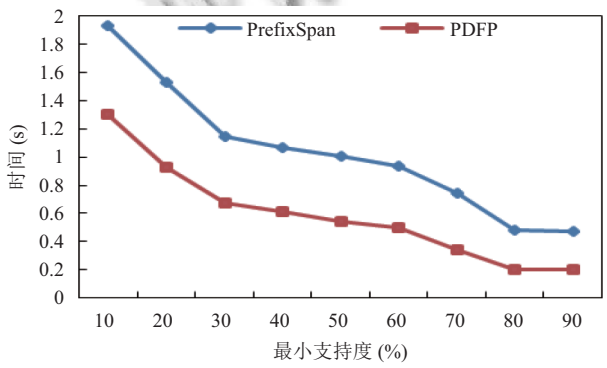

图 5 大支持度区间内算法时间效率对比

表 9 和图 5 从更宏观、更全面的角度进行实验, 展现了最小支持度在更大范围内变化时两算法的执行效率对比. 实验结果表明: 随着支持度阀值增大, PXFP 算法和 PrefixSpan 算法的执行时间都在下降, 同时, PXFP 算法在各支持度下的时间效率都要明显高于 PrefixSpan 算法. 其中, 最小支持度在 10%–30% 的范围内变化时, 由于挖掘出的频繁子路径数量减少速度很快, 因此算法执行时间下降速度很快; 最小支持度在 40%–70% 的范围内, 频繁子序列数量减少速度相对较慢, 因此挖掘算法的执行时间降速放缓; 然而, 在 70% 及以上的支持度下, 由于此时已经不再有频繁模式被挖掘出来, 算法执行时间再一次骤减, 并且由于两算法在扫描一次数据库后基本不用再进行后面的循环, 因此两算法的运行时间都趋于 0, PXFP 算法的优势也不如产生大量频繁子路径时明显.

实验 3. 比较不同 XML 文档数下的执行时间

创建 XML 数据集, 其中存放的 XML 文档数量由 1000, 2000, 3000, 逐渐递增至 10000. 控制支持度阀值为 20%, 对该数据集进行频繁路径挖掘, 考察随着 XML 文档数量, 即标签序列数量攀升的情况下, 两算法的执行效率. 实验结果如图 6.

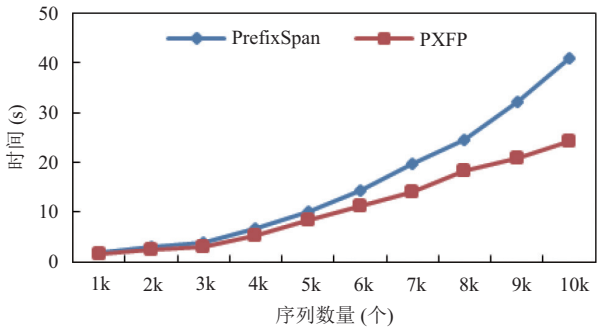

图 6 标签序列数量增加时执行时间的比较

图 6 表明, 在固定的支持度阀值下, 随着 XML 文档数据集规模的增大, 即标签序列数量的增加, PXFP 算法和 PrefixSpan 算法的执行时间在攀升. 在数据量较小的情况下, 两算法的运行时间效率相差不多, 随着数据量不断增加, PXFP 算法执行时间上的优势在增大, 特别地, 当序列数量为 10 K 时, PXFP 算法所需要的时间几乎是 PrefixSpan 算法运行时间的 60%, 并且可以预见的是, 当数据量更大时, PXFP 算法的优势将更加明显. 因此可以得出结论, 在任何数据量下,

PXFP 算法有高于 PrefixSpan 算法的执行效率, 并且数据量越大, PXFP 算法的优势越明显.

5 总结

本文中提出了基于序列前缀技术的 XML 频繁路径挖掘算法—PXFP 算法. PXFP 算法结合了序列前缀、位置信息等思想及 XML 树形结构特征, 用于从 XML 文档集中挖掘出频繁模式. PXFP 算法有如下的优点:

1) 广度优先遍历 XML 文档树并以“节点: 父节点”形式表示每个节点, 这种序列化的方式在不遗漏 XML 文档树的结构信息的同时降低了序列中的节点冗余, 减小了待挖掘的序列长度;

2) 较之 PrefixSpan 算法, 不需要产生投影数据库, 大大节约了存储空间;

3) 由位置信息直接定位到序列数据库中, 不需要多次扫描数据库及投影数据库, 减少时间开销.

在实验中, PXFP 算法用于挖掘 XML 频繁路径在时间效率和空间效率上均优于经典的 PrefixSpan 算法, 证明了算法的有效性. 但是, 本文中提出的 XML 频繁路径挖掘算法在应用于大型 XML 文档或 XML 数据流还有改进和提升的空间, 这是未来进一步研究的方向.

参考文献

1 Agrawal R, Srikant R. Mining sequential patterns. Proceedings of the 11th International Conference on Data Engineering. Washington, DC, USA. 1995. 3–14.

2 Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology. London, UK. 1996. 3–17.

3 Han JW, Pei J, Yin YW. Mining frequent patterns without candidate generation. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. New York, NY, USA. 2000. 1–12.

4 Han JW, Pei J, Mortazavi-Asl B, *et al*. FreeSpan: Frequent pattern-projected sequential pattern mining. Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, NY, USA. 2000. 355–359.

5 Pei J, Han J, Mortazavi-Asl B, *et al*. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. Proceedings of the 17th International Conference on Data Engineering. Washington, DC, USA. 2001. 215.

6 张坤, 朱扬勇. 无重复投影数据库扫描的序列模式挖掘算法. 计算机研究与发展, 2007, 44(1): 126–132.

7 张利军, 李战怀, 王淼. 基于位置信息的序列模式挖掘算法. 计算机应用研究, 2009, 26(2): 529–531.

8 刘栋, 尉永清, 薛文娟. 基于 Map Reduce 的序列模式挖掘算法. 计算机工程, 2012, 38(15): 43–45. [doi: 10.3778/j.issn.1002-8331.2012.15.010]

9 吴信东, 谢飞, 黄咏明, 等. 带通配符和 One-Off 条件的序列模式挖掘. 软件学报, 2013, 24(8): 1804–1815.

10 刘端阳, 冯建, 李晓粉. 一种基于逻辑的频繁序列模式挖掘算法. 计算机科学, 2015, 42(5): 260–264. [doi: 10.11896/j.issn.1002-137X.2015.05.052]

11 Beedkar K, Gemulla R. LASH: Large-scale sequence mining with hierarchies. Proceedings of the 2015 ACM SIGMOD Inter-national Conference on Management of Data. New York, NY, USA. 2015. 491–503.

12 Leung HP, Chung FL, Chan SCF, *et al*. XML document clustering using common XPath. Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration. Washington, DC, USA. 2005. 91–96.

13 贝毅君. XML 数据频繁模式挖掘技术研究[博士学位论文]. 杭州: 浙江大学, 2008.

14 雷向欣, 杨智应, 黄少寅, 等. XML 数据流分页频繁子树挖掘研究. 计算机研究与发展, 2012, 49(9): 1926–1936.

15 李巍, 李雄飞, 郭建芳. XML 空间频繁变化结构挖掘方法. 计算机学报, 2013, 36(2): 317–326.

16 INEX. Initiative for the evaluation of XML retrieval. http://inex.mmci.uni-saarland.de/data/documentcollection.html, 2014.