

# 基于压缩结构树的 XML 数据频繁模式挖掘研究

曹洪其<sup>1</sup>, 牛天耘<sup>2</sup>, 孙志挥<sup>2</sup>

(1. 南通职业大学电子工程系, 南通 226007; 2. 东南大学计算机科学与工程系, 南京 210096)

**摘 要:** XML 文档频繁模式挖掘是 XML 相关研究工作中的重要内容。在现有的频繁树结构挖掘算法 WL 的基础上, 提出了一种高效的基于压缩结构树存储结构的 XML 数据频繁模式挖掘算法 AFPMX\_CST。该算法压缩了搜索空间, 减少了扫描次数, 相对于 WL 算法在时间效率和空间效率方面具有更加良好的性能。同时, 该文进一步研究了将挖掘结果转换为相应的 DTD 格式的方法及过程。实验结果表明 AFPMX\_CST 算法是可行和有效的。

**关键词:** XML; 数据挖掘; 频繁模式; 算法; DTD

## Research of Frequent Pattern Mining from XML Data Based on Compressed Structure Tree

CAO Hongqi<sup>1</sup>, NIU Tianyun<sup>2</sup>, SUN Zhihui<sup>2</sup>

(1. Department of Electronic Engineering, Nantong Vocational College, Nantong 226007;

2. Department of Computer Science and Engineering, Southeast University, Nanjing 210096)

**【Abstract】** Frequent pattern mining based on XML document is an important content in XML-related research. An efficient algorithm called AFPMX\_CST is presented to discover frequent pattern in XML data based on compressed structure tree of storing XML data, with the existing frequent tree structure mining algorithm WL. It compresses the searching space, reduces scanning times, so it is much better than WL in time efficiency and space efficiency. At the same time, the methods and process to change mining results into corresponding DTD patterns are researched. It is proved both in theory and practice that this algorithm is adoptable and effective.

**【Key words】** XML; Data mining; Frequent pattern; Algorithm; DTD

作为半结构化数据的典型代表, XML<sup>[1]</sup>已成为Web上数据表示、集成和交换的标准。在许多XML数据文档中存在着大量重复的子元素, 从而在XML文档结构树中有许多节点拥有完全相同的树型结构。为此, 本文结合XML文档的特点, 采用XOEM模型, 引入压缩结构树来表示这种完全相同的树型结构, 在基于扩展路径思想算法WL<sup>[2]</sup>的基础上, 提出了一种高效的基于压缩结构树存储结构的XML数据频繁模式挖掘算法AFPMX\_CST。同时进一步研究了将挖掘结果转换为相应的DTD格式的方法及过程。AFPMX\_CST算法压缩了搜索空间, 减小了扫描XML数据文档的次数, 从而大大提高了数据挖掘效率。

### 1 有关概念与定义

半结构化数据的内容与模式都包含在数据中, 被称为是自描述的。XOEM模型是建立在OEM<sup>[3]</sup>模型基础上, 用来表示XML数据的一种自描述的嵌套对象模型。

一个XOEM对象 $o$ 包括对象标识符( $\&o$ )、对象的类型(Type)、对象的值( $\text{val}(\&o)$ )。对象的类型包括元素类型(ELEMENT)、属性类型(ATTRIBUTE)、文本类型(TEXT)。对于元素类型, 值为其子对象的集合(子对象为无序的情况) $\{l_1:\&o_1, l_2:\&o_2, \dots, l_p:\&o_p\}$ , 或者值为子对象的列表(子对象为有序的情况) $\langle l_1:\&o_1, l_2:\&o_2, \dots, l_p:\&o_p \rangle$ 。其中 $\&o_i$ 是子对象 $o_i$ 的标识,  $l_i$ 是 $o_i$ 的标记。

根据 XOEM 模型, XML 数据能够用一个树型结构图(XML 文档数据图)来表示。在这种图中, 每个结点对应于一个

数据对象并赋予唯一的结点号(对象标识符); 节点的每一条引出边表示相应对象的一个子对象的标记或对象的一个属性标记, 并用子对象名或属性名标记; 对于叶子结点, 含有对象的字符串的值。

#### 1.1 结构表达式和压缩结构表达式

XML 文档数据图存储了 XML 数据的结构与内容。为了不失通用性, 引入结构表达式和结构树的概念, 只描述 XML 文档的结构信息而略去具体内容。同时, 引入压缩结构树来表示具有相同的树型结构, 并用压缩结构表达式来表示对应的压缩结构树。

**定义 1** 结构表达式。假设 $se_i$ 是对象 $o_i$ 的结构表达式,  $i = 1, 2, \dots, p$ 。

(1) 空结构 是任何对象的结构表达式;

(2) 如果 $\text{val}(\&o) = \{l_1:\&o_1, l_2:\&o_2, \dots, l_p:\&o_p\}$ , 并且 $\{i_1, \dots, i_k\}$ 是 $\{1, \dots, p\}$ 的子集,  $k > 0$ , 则 $\{l_{i_1}:se_{i_1}, \dots, l_{i_k}:se_{i_k}\}$ 是对象 $o$ 的一个结构表达式。

定义中对象 $o$ 的一个结构表达式 $\{l_{i_1}:se_{i_1}, \dots, l_{i_k}:se_{i_k}\}$ 对应的就是一棵结构树, 标记为 $l_{ij}(1 \leq j \leq k)$ 的边连接着 $se_{ij}(1 \leq j \leq k)$ 对应的子树。如果 $se_{ij} = \text{空}$ , 标记为 $l_{ij}$ 的边连接一个叶子结点。

**定义 2** 压缩结构表达式。假设 $ce_i$ 是对象 $o_i$ 的结构表达式,

**基金项目:** 国家自然科学基金资助项目(70371015)

**作者简介:** 曹洪其(1957-), 男, 副教授, 主研方向: 数据库, 数据挖掘等; 牛天耘, 硕士生; 孙志挥, 教授、博导

**收稿日期:** 2006-03-30 **E-mail:** chq@mail.ntvc.edu.cn

1 i p。

(1)空结构 是任何对象的压缩结构表达式；

(2)如 $val(\&o)=\{l_1:\&o_1, l_2:\&o_2, \dots, l_p:\&o_p\}$ ，并且 $\{i_1, i_1+1, \dots, i_1+c_{i1}-1, \dots, i_k, i_k+1, \dots, i_k+c_{ik}-1\}$ 是 $\{1, \dots, p\}$ 的子集， $k>0$ ，若对任意 $j$ ，满足 $1 \leq j \leq k$ ， $l_{ij}=\dots=l_{ij+c_{ij}-1}$ ， $ce_{ij}=\dots=ce_{ij+c_{ij}-1}$ ，则 $\{l_{i1}[c_{i1}]:ce_{i1}, \dots, l_{ik}[c_{ik}]:ce_{ik}\}$ 是对象 $o$ 的一个压缩结构表达式。

由于压缩结构既可以减少占用的外存和内存空间，也可以缩短挖掘时对结构树的遍历时间，方便支持度的统计，因此，将用这种压缩的形式来描述 XML 文档结构，同时下文中所指的结构表达式即为压缩结构表达式。

## 1.2 结构包含关系和完全结构表达式

XML 数据的挖掘问题就是要找出所有的最大频繁结构表达式。为此，需引进结构包含、完全结构表达式等概念。

**定义 3** 结构包含：

(1)空结构 包含于任意结构表达式；

(2)设 $ce=\{l_1[c_1]:ce_1, \dots, l_p[c_p]:ce_p\}$ ， $ce'=\{l'_1[c'_1]:ce'_1, \dots, l'_q[c'_q]:ce'_q\}$ ，如果对于每个 $ce_i$ ，存在某个 $ce'_{j_i}$ ，使得 $ce_i$ 结构包含于 $ce'_{j_i}$ ，并且 $l_i=l'_{j_i}$ ， $c_i \leq c'_{j_i}$ ， $1 \leq i \leq p$ ， $\{j_1, \dots, j_p\}$ 是 $\{1, \dots, q\}$ 的子集，那么称结构表达式 $ce$ 包含于结构表达式 $ce'$ ，记为 $ce \subseteq ce'$ 。

**定义 4** 完全结构表达式。对于 XML 中的对象 $\&o$ ，存在唯一的一个结构表达式，该表达式不为 $o$ 的任何其他结构表达式所包含，则称其为对象 $\&o$ 的完全结构表达式。

## 1.3 频繁结构表达式

对于一个 XML 文档，如果其根元素的完全结构表达式(也称为该 XML 文档的完全结构表达式)包含结构表达式 $ce$ ，则称该 XML 文档支持 $ce$ 。XML 文档库中支持 $ce$ 的文档个数，称为结构表达式 $ce$ 的支持数。

**定义 5** 频繁结构表达式。给定 XML 文档库(用 $T$ 表示)，如结构表达式 $ce$ 的支持数不小于用户给定的最小支持数 $min\_sup$ ，则称 $ce$ 为频繁结构表达式；如 $ce$ 是频繁的且不包含于其他的频繁结构表达式，则称 $ce$ 是最大频繁结构表达式。

## 1.4 k-结构表达式和路径表达式

**定义 6** k-结构表达式。包含 $k$ 个叶子结点的结构树就是 $k$ -结构树，对应的结构表达式即为 $k$ -结构表达式，把频繁 $k$ -结构表达式的集合记为 $F_k$ 。

结构树中的每个叶子结点可以用由根结点出发到达该结点的唯一路径来表示。路径可用一个形如 $[l_1[c_1], \dots, l_n[c_n]]$ 的路径表达式来表示，其中 $l_i$ 表示树中根结点， $l_i[c_i]$ 是路径中边的标记。一个 $k$ -结构表达式可由 $k$ 个互不为前缀的路径表达式 $p_1, p_2, \dots, p_k$ 顺序粘合而成，所有路径的 $l_i$ 粘合成根结点。在以后的讨论中，将用路径表达式序列 $p_1 p_2 \dots p_k$ 来表示 $k$ -结构表达式。

## 2 XML 频繁模式发现算法 AFPMX\_CST

### 2.1 算法思想

AFPMX\_CST算法是基于类Apriori<sup>[4]</sup>算法的一种改进算法，使用一种逐层搜索的迭代方法，找出所有的最大频繁结构表达式。

算法的主要步骤为：

(1)通过扫描XML文档库，找出频繁 1-结构表达式集合 $F_1$ 。

(2)根据Apriori性质，通过迭代循环，不断地由 $F_{k-1}$ 产生候选 $k$ -结构表达式集合 $C_k$ ，并由 $C_k$ 产生 $F_k$ ，直到不能产生新的频繁结构表达式集合(非空集合)。

(3)在所有的频繁结构表达式中找出最大的部分。

由于一个频繁 $(k+1)$ -结构表达式必定有 $k+1$ 个频繁子 $k$ -结构表达式，因此如果当一个XML文档包含 $k$ -结构表达式的数量少于 $k+1$ 时，则该文档必然不包含频繁 $(k+1)$ -结构表达式。此时，在形成 $F_k$ 后，就可以将该文档消除，以便在形成 $F_{k+1}$ 时减少扫描XML文档的个数，从而提高算法执行速度。为此，在AFPMX\_CST算法中，对每个文档设置了文档删除标志 $delete$ ，其作用就是用于实现以上功能。

### 2.2 AFPMX\_CST 算法

AFPMX\_CST 算法如下：

输入 XML 文档库  $T$ ， $min\_sup$ 。

输出 所有最大频繁结构表达式。

方法：

(1) $F_1=find\_F1(T, min\_sup)$ ;

//生成频繁 1-结构表达式集合

(2)for ( $k=2$ ;  $F_{k-1} \neq \emptyset$ ;  $k++$ )

{ $C_k=Candidates\_gen(F_{k-1}, min\_sup)$ ;

//根据 $F_{k-1}$ 生成候选集 $C_k$

for each  $t \in T$  do{

// $t$ 为相应 XML 文档的完全结构表达式

if  $t.delete=0$  then{

$t.count=0$ ;

//  $t.count$  为  $t$  包含的候选项数

for  $C_k$ 中的每个候选 $c$  do

if  $contain\_structure(t, c)$  then{

$c.count++$ ;

//如  $c$  被  $t$  包含，则支持度计数

$t.count++$ ; }

}

if  $t.count \geq min\_sup$  then

$t.delete=1$ ; //打上文档删除标志

}

$F_k=\{c \in C_k | c.count \geq min\_sup\}$ ;

}

(3) $F=find\_maximal(\bigcup_k F_k)$ ;

//在所有的频繁结构表达式中找出最大的部分

算法中在由 $F_{k-1}$ 产生候选 $k$ -结构表达式集合 $C_k$ 过程中，可以通过 $F_{k-1}$ 中两个前 $(k-2)$ 项相同的频繁 $(k-1)$ -结构表达式： $p_1 \dots p_{k-2} p_{k-1}$ 和 $p_1 \dots p_{k-2} p_k$ 合并产生频繁候选 $k$ -结构表达式 $p_1 \dots p_{k-2} p_{k-1} p_k$ ，我们称这样的两个频繁 $(k-1)$ -结构表达式构成一个匹配对，相应的合并操作称为连接操作。只要将 $F_{k-1}$ 中所有匹配对完成连接操作，便得到了候选集合 $C_k$ 。

### 2.3 算法分析和进一步优化

AFPMX\_CST算法是基于类Apriori算法的一种改进型算法，算法中 XML 数据文档采用压缩结构树的存储结构，与采用未经压缩的结构树的 WL 算法相比，具有良好的空间效率和时间效率。

(1)大大减少了树中的结点数，从而可以有效地节省内、外存的存储空间，同时挖掘时将压缩树读入内存所需的时间也相应地减少。

(2)压缩了搜索空间，减少了扫描次数。在AFPMX\_CST算法中，路径表达式也是一种压缩形式的表示，对应于WL中的多个路径表达式。因此，对于AFPMX\_CST算法，由 $F_{k-1}$ 产生 $F_k$ 时，频繁结构的扩展显然要快于WL算法。从而在对相

同的数据进行挖掘时, AFPMX\_CST算法所需的扫描次数更少, 整个挖掘过程将更快结束。

结合 XML 中的 DTD 的特点, 可以进一步改进 AFPMX\_CST 算法。由 DTD 定义可知, 当一个 XML 元素包含重复的子元素时, 则在元素类型声明中使用字符\*或者+来表示子元素任意的重复次数。实际中, 在分析 AFPMX\_CST 算法的挖掘结果时, 如果某一元素的出现次数大于阈值, 就可以把它当作重复子元素来看待, 而不必考虑其出现的实际次数。显然, 越大, 表示的结构越严格, 更趋于发现 XML 数据中准确的模式信息。但是如果过大, 过分拘泥于元素精确的次数, 又会失去模式定义中重复元素存在的意义。

与在分析挖掘结果阶段引入阈值相比较而言, 在 AFPMX\_CST 算法挖掘进行阶段就引入, 不仅最终可以得到相同的模式, 而且更可以起到提高挖掘速度改善算法性能的作用。因此, 对于 XML 文档压缩结构树中每一条边所对应的一个计数值  $c$  (初始结构树中具有相同子结构的边数) 作这样的处理: 当  $c$  时, 将  $c$  值统一计为+。所有大于等于的  $c$  值具有了相同的值+, 可以合并的边数量增加, 压缩结构树的压缩比例得到进一步的提高, 改善了算法的性能。同时由于的引入, 使得在挖掘重复元素时, 频繁模式不会无限扩张, 而是父子节点的比例达到 1: 时停止, 避免了大量冗余模式的生成。

### 3 挖掘结果的 DTD 表示

AFPMX\_CST算法的输出结果是一系列最大频繁结构表达式。为了能够存储和有效地利用这些信息, 需要将这些结构表达式转换成DTD形式, 产生的DTD对于XML数据的转换、存储<sup>[5]</sup>有着非常重要的作用。

DTD(Document Type Definition)是用来描述 XML 数据模式的模式定义语言, DTD 文档主要包括元素类型声明和属性类型声明。

元素类型声明的基本语法结构为

<!ELEMENT 元素名 元素定义>

属性类型声明的基本语法结构为

<!ATTLIST 元素名 属性名 属性类型 属性默认值类型>

每个结构表达式对应于一棵树, 可以通过广度优先遍历频繁结构树将挖掘结果转换成DTD形式。设树的根结点为 root, 记为root[1], 树中结点i的输入边标记为 $l_i[c_i]$ , 输出边标记序列为  $\{l_{i1}[c_{i1}], l_{i2}[c_{i2}], \dots, l_{im}[c_{im}]\}$  转换过程如下:

(1)广度优先遍历每一棵频繁结构树, 生成树中各元素的类型声明和属性声明。

1)依次对一棵树中每个结点i进行分析与操作, 形成相应元素 $l_i$ 的元素定义表达式 $ele\_exp_i$ 和属性类型表达式 $att\_exp_i$ 。其分析与操作过程如下:

If 结点 i 不为叶结点 then

{ $ele\_exp_i = "$ ”;

$att\_exp_i = "$ ”;

for ( $j=1$ ;

$j \leq m; j=j+1$ ) do

if  $l_{ij}[c_{ij}]$ 连接的不是叶子结 then

{ if  $c_{ij}=1$  then

{元素 $l_i$ 生成子元素 $l_{ij}$ ;

else  $exp_i = ele\_exp_i + ", " + l_{ij}$

}

else if  $c_{ij} < +$  then

{元素 $l_i$ 生成 $c_{ij}$ 个子元素 $l_{ij-1}, \dots, l_{ij-c_{ij}}$ ;

$ele\_exp_i = ele\_exp_i + ", " + l_{ij-1} \dots + ", " + l_{ij-c_{ij}}$ ;

}

else

.{元素 $l_i$ 生成重复子元素 $l_{ij}$ ;

$ele\_exp_i = ele\_exp_i + ", " + l_{ij} + "+ "$ }

.}

else

{if  $l_{ij} = "TEXT"$  then

$ele\_exp_i = ele\_exp_i + ", " + "#PCDATA"$ ;

else

$att\_exp_i = att\_exp_i + "CDATA" +$

$"#REQUIRED"$ ;

}

if  $ele\_exp_i = ""$  then

生成元素类型声明<!ELEMENT  $l_i$  ( $ele\_exp_i$ )>;

if  $att\_exp_i = ""$  then

生成属性声明<!ATTLIST  $l_i$   $att\_exp_i$ >;

if  $ele\_exp_i = ""$  and  $att\_exp_i = ""$  then

生成元素类型声明<!ELEMENT  $l_i$  EMPTY>.

}

2)重复 1)将所有的挖掘结果频繁结构树转换成 DTD 形式。

(2)挖掘结果常常包含多个频繁结构表达式, 可能会挖掘出某个元素的不同结构来, 从而在第(1)步生成所有频繁结构树的元素类型声明和属性声明之后, 随之会出现元素重复定义的问题。而 DTD 中元素声明具有唯一性, 因此, 需要合并这些不同的结构。

1)元素定义表达式的合并

若对应于某个元素 $l_i$ , 存在一组互不相同的元素定义表达式 $ele\_exp_{ij}, 1 \leq j \leq n$ , 则将这些表达式进行合并, 合并后的表达式表示为 $ele\_exp_i = ele\_exp_{i1} | \dots | ele\_exp_{in}$ 。设规则中的 $e, e_1, e_2$ 是 $ele\_exp_{ij}(1 \leq j \leq n)$ 所包含的内容, 采用如下变换规则来简化合并后的表达式 $ele\_exp_i$ :

$(e | e?) \quad e?$

$(e | e*) \quad e*$

$(e | e+) \quad e+$

$(e | e?) \quad e*$

$(e_1, e_2) | (e_1, e_3) \quad (e_1, (e_2 | e_3))$

$(e_1, e_2) | (e_1) \quad (e_1, e_2?)$

2)属性类型表达式合并

若对应于某个元素 $l_i$ , 存在一组互不相同的属性声明表达式 $att\_exp_{ij}, 1 \leq j \leq n$ , 则将这些表达式合并时, 对于 $att\_exp_{ij}$ 中所包含的所有属性 $att_k$ 的处理规则如下:

如果 $att_k$ 在所有的属性表达式 $att\_exp_{ij}(1 \leq j \leq n)$ 中出现, 则 $att_k$ 属性的缺省值类型为#REQUIRED, 否则 $att_k$ 属性的缺省值类型为#IMPLIED。合并后删除 $att\_exp_{ij}(1 \leq j \leq n)$ 中重复出现的关于 $att_k$ 的声明。

### 4 实验及分析

硬件运行环境: CPU 为 Pentium , 内存为 256MB, 硬盘为 40GB 的计算机; 软件运行环境: Windows 2000 操作系统, SQL Server 2000 关系数据库管理系统, Delphi6.0, 并指定 MS-XML 解析器作为分析 XML 文档、提供 DOM 编程接口的工具。实验数据来自于某药品行业销售数据库, 实验结果如图 1 所示。该图给出了支持度变化时 WL 算法与 AFPMX\_CST 算法执行时间的比较情况, 图中数据文档个数

(下转第 116 页)