路径分区编码优化小枝查询

徐小双[1,2] 冯玉才[2,3] 王 锋[1] 周英飚[2,3] 张 俊[1]
(黄冈师范学院教育科学与技术学院 黄冈 438000)[1]
(华中科技大学计算机学院多媒体数据库研究所 武汉 430073)[2]
(达梦数据库有限公司预研部 武汉 430073)[3]

摘 要 有效地存储查询XML文档已经成为当今数据库领域的研究热点。从XML文档的路径统计出发,提出了路径分区存储编码方案,并依此消除了小枝查询的后裔边和通配符。针对这类不含"//"和"*"的小枝查询,利用路径分区编码的特性,给出了基于结构约束节点的Twig查询算法,极大地减少了结构连接次数。实验表明,该算法能有效滤除无关元素,提高小枝查询效率。

关键词 XML,小枝查询,结构连接,路径统计
中图法分类号 TP311.13 **文献标识码** A

Path-partitioned Encoding Optimizes Twig Queries

XU Xiao-shuang[1,2] FENG Yu-cai[2,3] WANG Feng[1] ZHOU Ying-biao[2,3] ZHANG Jun[1]
(School of Educational Sci. & Tech., Huanggang Normal College, Huanggang 438000, China)[1]
(Institute of Multimedia Database, School of Computer, Huazhong Univ. of Sci. & Tech., Wuhan 430073, China)[2]
(Dameng Database Corporation Limited, Wuhan 430073, China)[3]

Abstract Effectively storing and querying XML documents becomes a hot research topic on current database domain. In the light of path summary, path-partitioned encoding scheme was proposed to store an XML document, and useful for eliminating descendant axes and wildcards in twig queries. For twig queries without "//" or "*", a new query algorithm was developed based on structure-constrained nodes, so structural joins extremely decreases. The results of experiments indicate the algorithm can significantly filter useless elements and improve the performance for twig queries.

Keywords XML, Twig query, Structural join, Path summary

小枝(Twig)查询通常用带标签的树模式(tree pattern)建模[1]。小枝查询就是从XML文档树中找到与其树模式匹配的元素序列。给定一个XML文档,快速、有效地完成小枝查询是近几年来研究的热点。

许多XML文档不仅拥有庞大数量的元素,而且存在深度嵌套的路径。另外,其中许多元素拥有相同的路径,使得XML文档存在局部结构的相似性,文档中不同路径的数量远远小于元素的数量。对于任何XML文档,这些路径的数量都是有限的。人们已经通过各种路径索引来提高相同路径下的元素存取效率;若能把相同路径的元素有效编码而聚集存储,则会提高存取元素的速度。

小枝查询中的后裔边"//"和通配符"*"增强了路径描述的灵活性,也导致了路径描述的非确定性,这使得处理小枝查询复杂化。特别地,很多结构连接算法无法直接有效处理小枝查询中的通配符。因此,目前主要在查询前利用DTD改写小枝表达式[2]来减少路径中的不确定操作符,以提高查询执行效率。认识到DTD存在递归回路时,具有描述无限的路径的能力,因此利用DTD改写小枝表达式无法完全消除不确定操作符。正如前所述,XML文档存在局部结构的相似性和拥有有限的简单路径,若能利用它的路径统计改写小枝查询,完全消除不确定操作符,必将简化结构连接算法,优化小枝查询过程。

以国产达梦数据库为研究平台,实现了DM5.6系统[3]对XML的支持。本文的主要贡献可概括为以下4个方面:(1)提出的路径分区存储编码方案具有编码简洁和路径明晰的特点,能有序聚集存储相同路径的元素。(2)依据XML文档的路径统计,提出了将任一小枝查询改写为$P^{\{/,[]\}}$子类查询的方法,以完全消除后裔边和通配符。(3)针对$P^{\{/,[]\}}$子类的小枝查询,根据路径分区的编码特点,给出了非常简洁的小枝查询算法,最大限度地减少了结构连接的次数。(4)通过实验验证了DM XML数据存储的有效性和XML查询的高效性。

本文第1节给出一些后续章节需要引用的基本概念;第

到稿日期:2009-08-14 返修日期:2009-12-14 本文受国家信息产业部科技攻关课题(No. 2005BA112A02-DB-DM)和湖北省发改委基金(No. [2007]1334)资助。

徐小双(1970-),男,博士生,副教授,主要研究领域为半结构化数据、数据库系统实现技术,E-mail:xxsh99@hgnu.edu.cn;冯玉才(1944-),男,教授,博士生导师,主要研究领域为现代数据库技术和数据仓库;周英飚(1970-),男,博士,副教授,主要研究领域为半结构化数据、并行计算和数据库系统实现技术。

2 节描述路径分区的存储编码方案；第 3 节依据 XML 文档的路径统计，提出 Twig 查询模式的解析过程，包含验证和改写树模式，以消除后裔边和通配符，最终形成查询模式集；第 4 节展示了解析后的 Twig 查询过程；第 5 节从各个方面分析了实验数据，验证了原型系统的性能。最后总结全文并展望了未来的研究。

1 预备知识

小枝(Twig)查询通常用树模式(tree pattern)建模，也被称作小枝模式(twig pattern)。文献[1]将所有的树模式归结为 $P^{\{/,*,//,[]\}}$ 类。根据缺省后裔边(//)、通配符(*)和分支([])，可进一步分为 $P^{\{/,*,[]\}}$，$P^{\{/,//,[]\}}$，$P^{\{/,*,//\}}$ 等子类。$P^{\{/,*,//\}}$ 没有分支而被称为线性模式，$P\{/\}$ 是仅包含孩子边(/)的线性模式，通常称为简单路径(模式)。图 1 给出了 3 个线性模式 $p=A/B//D$，$q=A/B/D/D$ 和 $r=A/B//D/F$，其中，模式 q 为简单路径。文献[3]指出，两个线性模式存在包含的充要条件是它们之间存在 embedding 映射，发现 embedding 映射能在多项式时间内完成。在图 1(a)和图 1(b)中，存在 $e:p\rightarrow q$ 和 $f:p\rightarrow q$ 两个从 p 到 q 的 embedding 映射，则 q 包含于 p ($q\sqsubseteq p$)。

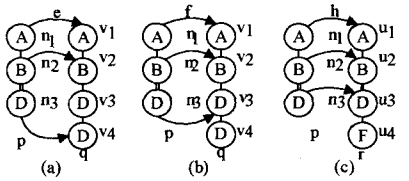

图 1 $q\sqsubseteq p$，$q\vdash p$ 和 $P=prefix(r,3)$

定义 1 设线性模式 $p,q\in P^{\{/,//,*\}}$。若存在 embedding 映射 $e:p\rightarrow q$，使得 p 的唯一根节点和叶节点分别对应 q 的根节点和叶节点，则 q 蕴含于 p，表示为 $q\vdash p$。

因为 e 是个特殊的 embedding 映射，若 $q\vdash p$，则必有 $q\sqsubseteq p$。同样，判定模式蕴含也能在多项式时间内完成，具体算法不再展开。用 $||p||$ 表示线性模式 p 的路径长度(节点数目)。在图 1(a)中，对于 $e:p\rightarrow q$，有 $e(n_1)=v_1$ 和 $e(n_3)=v_4$，故 $q\vdash p$。也可看出，$||p||=3$ 和 $||q||=4$。在图 1(c)中，线性模式 p 是以线性模式 r 的根节点开始的子树，我们说 r 是 p 的前缀，表示为 $p=Prefix(r,||p||)=Prefix(r,3)$。

2 路径分区编码

在实际中，XML 文档可能包含数量庞大的元素，具有复杂嵌套的结构，但文档中总是存在重复的简单路径。在文档中，简单路径的种类远远小于节点的数量，而且是有限的。例如，116M XMark 文档[3]包含多达 $1.78*10^6$ 个元素，但是其中的简单路径仅为 546 个。本节通过统计 XML 文档中出现的简单路径，建立了基于简单路径分区的 DM XML 存储编码方案。

2.1 路径统计

DM XML 将一份 XML 文档编码为 Doc(PT,ET,CP)三元组，PT 表统计出现在文档中的简单路径，ET 表存储文档中的元素，常量 CP 表示在任意简单路径下允许出现的元素个数。图 2 给出了一个 XML 文档树，图 3 中的 PT 表记录了文档中出现的简单路径。

PT 表可表示为 PT (PID, Path, PLabel)。其中，字段 Path 由路径标签和间隔符号“/”组成，表征路径信息。字段 PID 标示唯一的简单路径，为主键。设 $t\in$PT，t. PLabel 由简单路径及其所有前缀的 PID 和点号(.)连接而成：(1) 若 $||t.\text{Path}||=1$，$t.\text{PLabel}=t.\text{PID}$；(2)若 $||t.\text{Path}||>1$，必存在 $u\in$PT 满足 $u.\text{Path}=Prefix(t.\text{Path},||t.\text{Path}||-1)$，则 $t.\text{PLabel}=u.\text{PLabel}+.\ t.\text{PID}$。

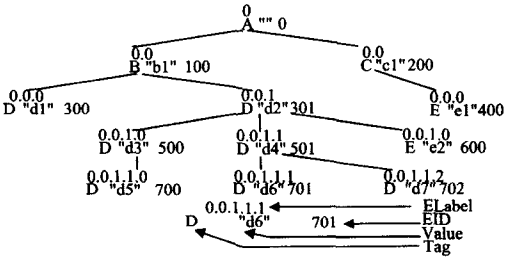

图 2 一棵 XML 文档树

2.2 元素编码

ET 表主要用来存储 XML 文档元素的信息，表示为：ET (EID, Value, ELabel)。其中，字段 Value 记录元素的值。

字段 EID 标示 XML 文档中的唯一元素，为主键。设 $u\in$ ET，必有 $t\in$PT 使得 t. Path 就是从根元素到 u 对应元素的路径，u. EID 方式编码为：(1) $u.\text{EID}\in[t.\text{PID}*\text{CP},(t.\text{PID}+1)*\text{CP}-1]$和$\lfloor u.\text{EID}/\text{CP}\rfloor=t.\text{PID}$。(2) 若 $u'\in$ET 且 u 和 u'的对应元素有相同路径，按深度遍历 u'是 u 后继，则 $u'.\text{EID}>u.\text{EID}$，$u'.\text{EID}\in[t.\text{PID}*\text{CP},(t.\text{PID}+1)*\text{CP}-1]$ 且 $\lfloor u'.\text{EID}/\text{CP}\rfloor=t.\text{PID}$。

字段 ELabel 推导元素的所在路径，其编码方式为：(1) 若 u 对应根元素，$u.\text{ELabel}=u.\text{EID} \bmod \text{CP}$；(2) 否则，设 $u'\in$ET 为其父亲元组，则 $u.\text{ELabel}=u'.\text{ELabel}+.\ u.\text{EID} \bmod \text{CP}$。

图 3 中的 ET 表展示了文档中元素的编码情况，其中 CP=100。路径 A/B/D/D/D 在 PT 表中标记为 7，在该路径下的 3 个元素对应元组 u_1,u_2,u_3 的 EID 字段编码在$[7*\text{CP},(7+1)*\text{CP}-1]=[700,799]$区间内。它们按深度遍历依次出现，故它们的 EID 分别为 700，701 和 702。从 ET 表可以看出，XML 文档的所有路径相同的元素按升序连续聚集在 ET 关系表中，因此把这种编码方案称为路径分区编码。ET 关系表能够很好地支持在字段 EID 上的索引。同时，从图 2，图 3 共同得到 u_1 的父亲元祖的 EID 为 500，ELabel 为 0.0.1.0，故 $u_1.\text{ELabel}=0.0.1.0+.\ u_1.\text{EID} \bmod \text{CP}=0.0.1.0.0$。

PID	Path	PLabel
0	A	0
1	A/B	0.1
2	A/C	0.2
3	A/B/D	0.1.3
4	A/C/E	0.2.4
5	A/B/D/D	0.1.3.5
6	A/B/D/E	0.1.3.6
7	A/B/D/D/D	0.1.3.5.7

PT 表

EID	Value	ELabel
0	null	0
100	b1	0.0
200	c1	0.0
300	d1	0.0.0
301	d2	0.0.1
400	e1	0.0.0
500	d3	0.0.1.0
501	d4	0.0.1.1
600	e2	0.0.1.0
700	d5	0.0.1.0.0
701	d6	0.0.1.1.1
702	d7	0.0.1.1.2

ET 表

图 3 PT 表和 ET 表(CP=100)

2.3 路径分区编码的属性

定理 1 给定 XML 文档的编码三元组 Doc(PT,ET,CP),对于任意 $u\in ET$,u. ELabel 唯一确定 u 所有的祖先元组。

例证定理 1。例如,标记为 702 的元组 u_3 的 ELabel 表示为 $\rho=(0,0,1,1,2)$,u_3. EID mod CP=7 唯一确定它的简单路径 A/B/D/D/D,该路径的 PLabel 表示为 $\lambda=(0,1,3,5,7)$。则 $\lambda*CP+\rho=(0*100+0,1*100+0,3*100+1,5*100+1,7*100+2)=(0,100,301,501,702)$顺序标识了 u_3 及其所有祖先元组。

推论 1 给定 XML 文档的编码三元组 Doc(PT,ET,CP),存在 $u,u'\in ET$,其路径分别为 $p,p'\in P^{\{/\}}$。若 $p'=Prefix(p,||p'||)$且 u'. ELabel 是 u. ELabel 前缀字符串,则 u'是 u 的祖先元组。

推论 1 能通过定理 1 得到。通过推论 1,容易判定来自 ET 表的两个元组是否存在 AD(祖先/子孙)关系。若推论 1 还满足 $||p'||=||p||-1$,可得知两元素存在 PC(父亲/孩子)关系。因此,路径分区编码方案若能明确表明两元组对应元素的 PC 关系和 AD 关系,必将加快 XML 数据查询。

3 Twig 模式解析

将 XML 文档映射存储在关系数据库后,在处理 Twig 查询过程中,需要首先验证和改写树模式,形成查询模式集,以消除后裔边(//),甚至通配符(*)。若查询模式集不为空,则要求集合中的任一查询模式属于子类 $P^{\{/,[]\}}$,即仅含孩子边(/)和分支([]),以方便形成简洁高效的连接算法。

3.1 线性 Twig 模式解析

定义 2 给定 XML 文档的编码三元组 Doc(PT,ET,CP),设线性 Twig 模式 $p\in P^{\{/,//,*\}}$。p 的解析模式集定义为 $\mathbf{A}(p)=\{p_i \mid p_i\vdash p, p_i\in \pi_{Path}(PT)\}$,其中,$\pi$ 为关系投影运算。

例如,设线性模式 A//D,A//D//D 和 A//D//E。联系图 3 的 PT 表,有 $\mathbf{A}$(A//D)={A/B/D,A/B/D/D,A/B/D/D/D},$\mathbf{A}$(A//D//D)={A/B/D/D,A/B/D/D/D}和 $\mathbf{A}$(A//D//E)={A/B/D/E}。根据定义 1 中线性模式蕴含的定义,在 PT 表下求解解析路径集能在多项式时间内完成。通过解析线性 Twig 模式,得到了与之匹配的简单路径,在结构连接前仅仅取出这些路径下的元组,过滤了大部分无关元组,减小数据处理规模。

3.2 Twig 模式的路径连接

定义 3 在 Twig 模式 $p\in P^{\{/,//,[],*\}}$ 中,对任一节点 n,如果属于输出节点、分支节点或叶节点三者之一,则称之为结构约束节点(SCN),从根到该节点的线性模式称为约束路径,表示为 n. Path。输出节点及其祖先约束节点统称为主结构约束节点,该集合表示为 $M(p)$。

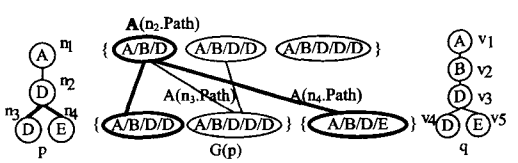

图 4 Twig 模式解析

设有 Twig 模式 p=A//D[//D][//E]如图 4 所示,n_2 既是输出节点,又是分支节点;n_3 和 n_4 是叶节点,故结构约束节点有 n_2,n_3 和 n_4,并且有 n_2. Path=A//D,n_3. Path=A//D//D 和 n_4. Path=A//D//E。正如上节所说,我们可以求出 n_2 的解析路径集 $\mathbf{A}(n_2.Path)=\mathbf{A}$(A//D)={A/B/D,A/B/D/D,A/B/D/D/D}。剩余两个节点的解析路径集也已经在上节中求出,并展示在 G(p)中。根节点 n_1 因不满足定义而不是结构约束节点。

定义 4 给定 XML 文档的编码三元组 Doc(PT,ET,CP),存在 n'和 n 是相间的两个结构约束节点,n'是 n 的祖先。如果 $p\in\mathbf{A}(n'.Path)\subset P^{\{/\}}$,$\exists q\in\mathbf{A}(n.Path)\subset P^{\{/\}}$,有 embedding 映射 f:n'. Path→p 能测试出 $p\vdash n'$. Path,embedding 映射 h:n. Path→q 能测试出 $q\vdash n$. Path,且满足 $p=prefix(q,||p||)$和 $f(n'.Path)\subset h(n.Path)$,我们说简单路径 p 能连接简单路径 q。n'与 n 的路径连接定义为:n'. Path $\bowtie$ n. Path={$(p,q)\mid p\in\mathbf{A}(n'.Path), q\in\mathbf{A}(n.Path)$, p 能连接 q }。

在定义 4 中,作为简单路径 $q\in\mathbf{A}(n.Path)$的前缀,简单路径 $p\in\mathbf{A}(n'.Path)$能连接 q 的关键点在于:(1)映射 h:n. Path→q 能测试出 $q\vdash n$. Path;(2)作为映射 h 的部分,映射 f:n'. Path→p 能测试出 $p\vdash n'$. Path。即两个测试线性模式蕴含的映射能无冲突建立起节点的约束路径到相应节点的解析路径的 embedding 映射,而这两个解析路径存在前缀关系,如图 5 所示。

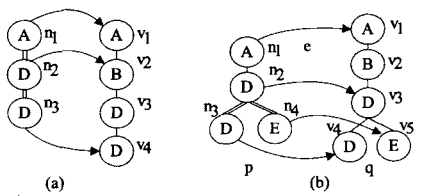

图 5 无冲突的 embedding 映射

例如,在图 4 的模式 p 中,n_2. Path=A//D,n_3. Path=A//D//D 和 A/B/D∈$\mathbf{A}(n_2.Path)$,A/B/D/D∈$\mathbf{A}(n_3.Path)$,我们建立从 n_3. Path 到 A/B/D/D 的 embedding 映射 h={(n_1,v_1),(n_2,v_3),(n_3,v_4)},能测试出 A/B/D/D⊢n_3. Path;同样建立从 n_2. Path 到 A/B/D 的 embedding 映射 f={(n_1,v_1),(n_2,v_3)},能测试出 A/B/D⊢n_2. Path,这里 A/B/D=$Prefix$(A/B/D/D,3)且 $f(n_2.Path)\subset h(n_3.Path)$,如图 5(a)所示。因此,(A/B/D,A/B/D/D)∈n_2. Path $\bowtie$ n_3. Path;同理,(A/B/D,A/B/D/D/D)∈n_2. Path $\bowtie$ n_3. Path。虽然 A/B/D/D 同时属于 $\mathbf{A}(n_2.Path)$和 $\mathbf{A}(n_3.Path)$的元素,根据定义 4,(A/B/D/D,A/B/D/D)∉n_2. Path $\bowtie$ n_3. Path;同时,(A/B/D/D/D,A/B/D/D)∉n_2. Path $\bowtie$ n_3. Path。最后,有 n_2. Path $\bowtie$ n_3. Path={(A/B/D,A/B/D/D),(A/B/D,A/B/D/D/D),(A/B/D/D,A/B/D/D/D)}。

给定 XML 文档的编码三元组 Doc(PT,ET,CP)和 Twig 模式 p,求出所有结构约束节点的路径连接,把解析路径看成一个带标签的节点,并在能连接的解析(简单)路径上加边,形成一个路径连接图 G(p),如图 4 所示。G(p)反映了路径连接的总体特征。

3.3 复杂模式解析

给定 XML 文档的编码三元组 Doc(PT,ET,CP),作为定义 2 的延伸,模式 $p\in P^{\{/,//,[],*\}}$ 的解析模式集也表示为 $\mathbf{A}$(p)。

在它的路径连接图 $G(p)$ 中抽取每个约束节点的一条解析路径，形成来自于 $G(p)$ 的一个子树 T，如图 4 的 $G(p)$ 加黑部分。按如下步骤形成模式 p 的一个解析模式 $q\in A(p)$，它属于 $P^{\{/,[]\}}$ 子类，不含任何后裔边(//)、通配符(*)：

步骤 1　根据 T 的根的标签形成一个简单树模式。

步骤 2　作为 T 的非根节点，它的标签减去其父亲的标签生成的路径片断形成一个树模式片断，作为 q 的一部分，并与其父亲的标签形成的树模式的最后节点相联接。

步骤 3　重复步骤 2，直到 T 的所有节点被遍历。

步骤 4　从 p 的输出节点中的那条解析路径形成对应的树模式(片断)的最后节点为 q 的输出节点。

图 5(b)显示出了 p=A//D[//D][//E]的一个解析模式 q=A/B/D[/D][/E]。

定理 2　给定 Doc(PT,ET,CP)，如果模式 $q\in P^{\{/,[]\}}$ 是模式 $p\in P^{\{/,//,[],*\}}$ 的一个解析模式，则 $q\sqsubseteq p$。

证明：通过上述解析模式形成过程，得知 q 生成于 $G(p)$ 的一个子树 T。根据子树 T 的节点性质，对于任一 p 上的约束节点 n_i，存在一个路径 $p_i\in \mathbf{A}(n_i.\mathrm{Path})$ 是 T 上一个节点的标签，并能建立测试出蕴含的 embedding 映射 $f_i:n_i.\mathrm{Path}\rightarrow p_i$。再根据子树 T 的边性质，可知 $f_i:n_i.\mathrm{Path}\rightarrow p_i$ 与 p 上其它的测试蕴含的 embedding 映射无冲突。

由于 q 生成于子树 T，令 $e=\bigcup f_i:n_i.\mathrm{Path}\rightarrow p_i$，则 e 就是一个从 p 到 q 的一个 embedding 映射。根据文献[3]得 $q\sqsubseteq p$。由包含的性质可知，根据 q 查询的结果一定包含在根据 p 查询的结果内。

图 5(b)显示了从 p=A//D[//D][//E]到 q=A/B/D[/D][/E]的一个 embedding 映射 $e=\{(n_1,v_1),(n_2,v_3),(n_3,v_4),(n_4,v_5)\}$。

3.4　解析模式算法

给定 Doc(PT,ET,CP)，模式 $p\in P^{\{/,//,[],*\}}$ 的解析模式不是唯一的，通常 $\mathbf{A}(p)\geqslant 0$。算法 1 描述了根据 $G(p)$ 求出解析模式集的过程。

算法 1　求解模式的解析模式集

```
Globe Variable;
Node N, Node R
Boolean FindTree(Node n', int i )
1:  p←A(n'. Path)[i]; Expr[n'] ←p;
2:  bMatch←true;
3:  for k=1 to ChildNum(n') do
4:      n←GetSCN(n',k);
5:        bfound←false;
6:      for j=1 to | A(n. Path)|
7:        q←A(n . Path)[j];r←false;
8:      if (p,q)∈n'. Path ⋈ n. Path
          then r←FindTree (n,j);
          else continue;
9:      if r then Expr [n] ←q;
10:      if (n=N ) ∧ r then output Expr ;
11:    bfound ←bfound ∨ r;
12:  end for
13:  bMatch←bMatch ∧ bfound;
14:  if ! bMatch return false;
15:  return true;
Void AnalyzePattern(Pattern P)
16: N←the last constrained structural node of p;
17: R←the first constrained structural node of p;
18: for i=1to |A(R. Path)|
19: FindTree(R, i );
20: End for
```

在算法 1 中，自解释函数 ChildNum(n') 测试结构约束节点 n' 的分支个数；GetCSN(n',k)是获得节点 n' 的第 k 个分支上的最近结构约束节点；数组 Expr 记录满足路径连接且来自结构约束节点路径的解析路径，根据约束节点间暗示的 $G(p)$ 边关系，数组 Expr 按 3.3 节构造出一个解析模式。由行 10 可得，每次找到所有结构约束节点选择的简单路径，都会及时输出数组 Expr，因此，能在给定的 Doc(PT,ET,CP)下构造出所有的解析模式，最终形成 $\mathbf{A}(p)$。

根据上述算法，对于图 4 中的模式 p=A//D[//D][//E]，求得 $\mathbf{A}(p)$={ A/B/D[/D][/E],A/B/D[/D/D][/E] }。

根据算法 1，还能够有效地消除后裔边(//)和通配符(*)。例如，在图 3 所示的三元组 Doc(PT,ET,CP)中，设有 x=A//*//*//D，y=A//*//D[//D][/E]和 z=A//*[//D/D][//E]，则有 A(x)={A/B/D/D, A/B/D/D/D}，A(y)={A/B/D[/D][/E],A/B/D[/D/D][/E]}和 A(z)={A/B/D[/D/D][/E], A/B[/D/D/D][/D/E], A/B[/D/D][/D/E]}。

定理 3　给定 XML 文档的编码三元组 Doc(PT,ET,CP)，求解 $p\in P^{\{/,//,[],*\}}$ 的所有解析模式的时间复杂度是 $O(\prod|\mathbf{A}(n_i.\mathrm{Path})|)$，其中 n_i 为模式 p 的结构约束节点，$|\mathbf{A}(n_i.\mathrm{Path})|$ 为 n_i.Path 的解析路径个数。

由于通常模式 p 的结构约束节点少，产生的结构约束节点的解析路径个数不多，因此，解析过程的时间代价较小。对于相间的结构约束节点，它们的解析路径还要满足路径连接，因此也导致了最终生成的解析模式个数不是很多，即 $|\mathbf{A}(p)|$ 较小，这为后面的快速完成 Twig 查询打下了基础。

定理 4　给定 XML 文档的编码三元组 Doc(PT,ET,CP)，模式 $p\in P^{\{/,//,[],*\}}$ 的查询与 $\mathbf{A}(p)$ 的查询等价。

证明：对于任意 $q\in\mathbf{A}(p)$，根据定理 2，有 $q\sqsubseteq p$。因此，根据 A(p) 的所有模式查询给定的 XML 文档所得的结果一定被包含于根据 p 查询的结果内。设 $\mathbf{A}(p)\cup\mathbf{B}(p)$ 的查询与 p 的查询等价，且 $\mathbf{A}(p)\cap\mathbf{B}(p)=\emptyset$，其中，对于任意 $q'\in\mathbf{B}(p)$，有 $q'\sqsubseteq p$，$\mathbf{B}(p)$ 可能为无限集合。根据算法 1，至少存在一条从根到某节点的简单路径不在该 XML 文档出现，否则 $q'\in\mathbf{A}(p)$。而根据 q' 查询该 XML 文档必然返回空集。此时，根据 $\mathbf{A}(p)$ 查询的结果一定等于根据 p 查询的结果。命题得证。

定理 4 的意义在于把属于 $P^{\{/,//,[],*\}}$ 类的 Twig 查询等效转化为零个、一个或多个 $P^{\{/,[]\}}$ 子类的 Twig 查询。

4　Twig 查询算法

4.1　线性 Twig 查询

定理 5　给定 Doc(PT,ET,CP)，存在 $t\in$PT 使得简单路径 t.PLabel 对应于模式 $q\in P^{\{/\}}$ 的表达式，Twig 模式 q 的查询元素集为 $R(q)=\sigma_F\{u\mid u\in\mathrm{ET}\}$，其中条件表达式 $F=(t.\mathrm{PID}*\mathrm{CP}\leqslant u.\mathrm{EID}\leqslant(t.\mathrm{PID}+1)*\mathrm{CP}-1)$，$\sigma_F$ 为选择关系运算。

证明：根据 ET 表的编码规则，属于路径 t.PLabel 的所有元素按升序聚集存放在 ET 表中，这些元素的编号在[t.PID*

CP,(t.PID+1)*CP-1]区间内。

通过定理5,即使模式 $q\in P^{/}$ 拥有足够长的线性路径,它的Twig查询也不需要作任何连接操作。一种简单的方法是执行SQL语句查询按路径聚集在一起的元组,ET表在EID字段上的聚簇索引将加快这一查询。

4.2 复杂Twig模式查询

根据上节的结论,属于 $P^{/,//,[],*}$ 类的Twig查询能够等效转化为零个、一个或多个 $P^{/,[]}$ 子类的Twig查询。我们重点讨论 $P^{/,[]}$ 子类的Twig查询实现。

从定义3可知,在Twig模式 $q\in P^{/,[]}$ 中,仍然存在输出节点、分支节点、叶节点。更特别的是,这些结构约束节点从根到该节点的约束路径属于 $P^{/}$ 子类,祖先节点的路径是子孙节点的路径的前缀。图5(b)中,模式 q 的 v_3. Path=A/B/D, v_4. Path=A/B/D/D, v_5. Path=A/B/D/E。

根据定理5,先按这些节点的约束路径得到Twig查询的元组升序列。由于祖先/子孙节点间的路径存在前缀关系,根据推论1,能够判断相应序列中的元组的祖先/子孙关系。若从各个序列中找出一组元组满足模式 $q\in P^{/,[]}$ 的约束节点的结构关系,则输出节点序列上的那个元组即为所求之一,如图6所示。

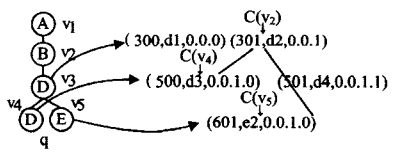

图6 $P^{/,[]}$ 子类的结构连接

算法2展示了整个结构连接的过程。主函数StructuralJoin(p)求出所有满足 $p\in P^{/,//,[],*}$ 的元组。对于 $q\in$ **A**(p),预先调整非根主结构约束节点,使之总成为其父亲的最后孩子。对于模式 q 的任意约束节点 n,C(n)是指向该元组升序列的光标;Advance(C(n))把光标向前推进一格,或因序列结束而返回空;Descendant (u',n) 根据推论1在该序列中依次寻找元组 u' 的所有子孙元组;MatchPattern(u,n)检测来自于该元组升序列的元组 u 是否有子孙元组在完成如图6所示的结构连接后,匹配以 n 为根节点的来自于模式 q 的子树模式。

在行21,如果 $\acute{n}$ 不属于主结构约束节点,只需要发现元组 $\acute{u}$ 与其子孙元组的一个结构连接,避免不必要的结构连接。如果 $\acute{n}$ 属于主结构约束节点,则需要发现 $\acute{u}$ 与其子孙元组的所有的结构连接,迫使程序找到所有满足模式 q 的输出元组。在行24,在非根主结构约束节点总是其父亲的最后孩子的保证下,一旦测试出 n 为输出节点且对应序列的元组 u 匹配模式 q,元组 u 一定是所求之一。

算法2 $P^{/,//,*,[]}$ 类的结构连接

```
tructuralJoin(p)
1: OutSet←Φ;
2: for each q∈A(p) do
3:   m←M(q);
4:   RNode←Output node of q;
5:   n←the first SCN of q;
6:   Initializing all cursors of q
7:     While (C(n)) do
8:     u←C(n);
9:     MatchPattern (u, n);
10:      Advance(C(n));
11:    end While;
12: end for;
13: Output OutSet
MatchPattern (u, n)
14: res←true;
15: for k←1 to ChildNum(n) do
16:   ń←GetSCN(n, k);
17:   D←Descendant (u, ń);
18:   if D =Φ then return false;
19:   for each ú∈D do
20:     r←MatchPattern(ú, ń);
21:     if r ∧ń∉m then break ;
22:   end for
23:   res←res∧r;
24: end for
25: if n=RNode∧res then
26: OutSet←OutSet∪{u};
27: end if
28: return res;
Descendant (u, ń)
29: D←Φ;
30: While (C(ń)) do
31:   ú←C(ń);
32:   L←Prefix(ú.ELabel,||u.ELabel||);
33:   if L<u.ELabel then Advance(C(ń));
34:   if L=u.ELabel then D←D∪{ú};
35:   if L>u.ELabel then break;
36: end While
37: return D ;
```

定理6 给定Doc(PT,ET,CP)和模式 $p\in P^{/,//,[],*}$,算法2的结构连接的时间复杂度为 $O(\sum_{q\in A(P)}\sum_{n\in SCN(q)}|L(n)|)$,其中SCN($q$)为 $q\in P^{/,[]}$ 的结构约束节点集,$|L(n)|$ 为约束节点 n 的元组序列长度。

定理6证明从略。算法2与 p 的分解模式和分解模式上的约束节点有关,而与 p 的最大路径长度无关。

5 实验与分析

以达梦关系数据库为平台,实现了DM5.6系统对XML的支持。本文的实验在一台基本配置为P4 2.8GHz,1G RAM,160G硬盘的PC上运行,底层操作系统是Windows XP。本实验所用数据集为XMark[5],分别选择1M,11M,22M,52M和110M的XML文档来生成不同大小的数据库。实验中,CP=2^{16},PT表统计出现在XML文档中的不同路径,从表1可以看出,文档元素个数远远大于PT的元组个数,XML文档规模的成倍增长并没有引起PT元组个数的显著增加,这也表明XML文档中存在大量重复的路径,同质XML文档在结构上具有相似性。ET表的字段ELabel记录元素路径标签,以十进制加点的字符串存贮,从表2可知,元素路径标签的存贮空间代价总体上还是可以接受的。

表1 XMARK文档存储属性

Data size(MB)	1	11	22	52	116
Elements($*10^6$)	0.02	0.17	0.33	0.76	1.78
Max/Ave depth	12	12	12	12	12
Path number	339	391	412	546	546
|ELabel| (MB)	0.17	1.19	2.31	5.32	12.4

实验中,测试对象是XMark 110M文档生成的数据库。

本文给出了具有各自特点的 $P1-P8$ 共 8 个 Twig 查询，如表 2 所列。$P1$，$P3$ 和 $P5$ 是线性查询，$P1$ 具有长路径，$P5$ 具有短路径，而 $P3$ 包含子孙边，其解析模式中存在递归路径。剩下的 5 个查询都带有分支结构。由于 XML 文档含有递归路径，$P6$，$P7$ 和 $P8$ 会解析为多个 $P^{\{/,//,[]\}}$ 子类的树模式。$P2$ 本身属于 $P^{\{/,[]\}}$ 子类，$P4$ 虽然包含子孙边，但解析后只生成一个 $P^{\{/,[]\}}$ 子类的查询。各个解析模式集的个数如表 2 所列。同时从中可以看出，解析过程所花费的时间很少。

表 2　Twig 查询及其模式解析

No	Twig 查询	|A(p)|	解析时间
P1	site/regions/africa/item/description/parlist/listitem/text/keyword	1	16
P2	site/closed_auctions/closed_auction[annotation/description/parlist/listitem/text/keyword/bold]/price	1	16
P3	site/closed_auctions//emph	9	113
P4	site/people/person[//age]//education	1	16
P5	site/people/person/name	1	16
P6	site//text[/bold]/emph/keyword	30	132
P7	site//listitem[//bold]/text[//emph]/keyword	243	415
P8	site//listitem[//bold]/text//emph	243	409

5.1　Twig 查询过程

为了便于比较，我们的结构连接命名为 DM XML，并与建立在区间编码上的 SegSJ[4]，iTwigJoin+TL[5] 算法进行了对比，参考数据来自于文献[6]。*i*TwigJoin+TL 采用 Tag+Level 索引过滤元素后完成结构连接运算，它是目前已知基于区间编码上的最快结构连接。SegSJ 把查询作为一个整体来匹配 Twig 查询，采用 F&B 结构索引可过滤掉所有无关的元素，但需要扫描的结点数量远多于 Tag+Level 索引的结点数量。DM XML 路径查询只需要读出结构约束节点解析路径对应的元素，大大降低了读入的元素数目，例如，对于 $P1$，SegSJ 扫描大于 32 万的元素，iTwigJoin+TL 扫描 2632 个元素；由于 $P1$ 只有唯一的输出节点为结构约束节点，且解析路径只有 1 条，因此仅需要扫描 245 个元素，涉及的元素种类和数目大为减少，如图 4(a)所示。在进行结构连接阶段，针对每个属于 $P^{\{/,[]\}}$ 子类的解析模式，结构连接过程与模式的长度无关，只与结构约束节点的个数有关，因此能大量节省连接时间。例如，$P1$ 的线性 Twig 查询对应着一条简单的 SQL 语句，由于元素按路径聚集存放，能方便读入所需元素，同时根本不需任何结构连接，整个查询仅耗时 22ms；$P3$ 本身就是自己的解析模式，含有 3 个结构约束节点，完成结构连接的时间为 370ms，比其他两种算法的时间花费少得多。

由于 SegSJ 和 iTwigJoin+TL 为了匹配通配符(*)必须遍历 XML 文档，导致扫描和查询大量无关的元素而使查询效率迅速降低，查询时间不可接受[7]。因此它们不适应含通配符的小枝查询。另一方面，在查询前生成解析模式集，该集合内所有小枝查询都属于 $P^{\{/,[]\}}$ 子类，消除了通配符和子孙边带来的不确定性。例如 site/*/person/name 花费 15ms 解析得到 $P5\in P^{\{/,[]\}}$，它根据算法 2 完成结构连接并返回查询结果；对于 site//*//closed_auction[annotation/description/parlist/ listitem/text/keyword/bold]/price 的表达式，只花费 16ms 解析得到 $P2$ 后实施查询。对于 site//*//*//listitem[//bold]/text//emph，花费 369ms 解析模式，生成与 $P8$ 相同的解析模式集。所以 DM XML 先根据 PT 表解析模式，提供了一种适应“*”的小枝查询方法。扫描的元素数量和查询运行时间如图 7 和图 8 所示。

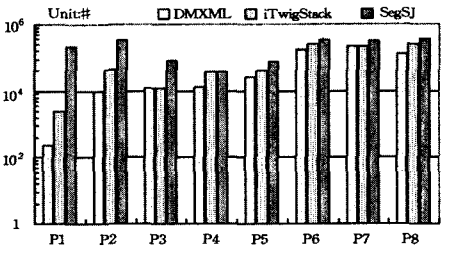

图 7　扫描的元素数量

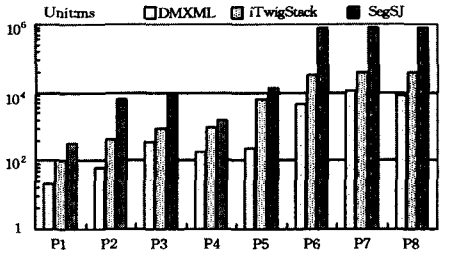

图 8　查询运行时间

5.2　扩展性分析

实验中，DM XML 系统读入 XML 文档后建立相应的数据库。图 9 给出了在 110k，1M，10M，50M 和 110M 的 XMark 数据库下，P1－P8 的查询扩展性能。从图 9 可以看出，随着 XML 文档规模的扩大，DM XML 执行小枝查询的时间较为缓和地增加。

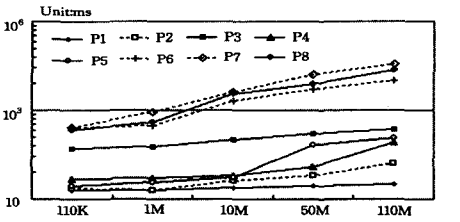

图 9　查询的扩展性能

在执行小枝查询前，文献[2,9]利用不包含环路的 DTD 去除路径表达式中的部分“*”和“//”，但当 DTD 存在环路时，无法改写树模式以除去全部的“*”和“//”；文献[10]单纯依据小枝查询表达式的语义用层约束条件近似消除通配符。应用路径统计在查询前改写小枝表达式的研究工作尚未见到，目前路径统计通常仅为 XML 文档建立路径索引。如 1-index[11] 路径索引能够很好地满足简单路径查询的要求，但在实现性能上稍显不足。在结构连接的过程中，文献[5]利用标签和路径长度来过滤结构连接中的无效元素，完成了基于区间编码的最有效的结构连接算法，但仍然存在无效元素参加结构连接。路径统计比标签和路径长度含有更多的路径信息，文献[12]根据路径统计分区存储区间编码的元素，并讨论了优化树模式和求解有效路径，但整个查询过程完全建立在关系代数基础上。

结束语　本文提出了 XML 文档的路径分区编码存储方案，它能有效表示元素间的 AD 或 PC 关系，且存储代价不是太高。在借助路径统计完成模式解析后，查询模式集中的每个模式都属于 $P^{\{/,[]\}}$ 子类。该类树模式能利用路径分区编码

（下转第 204 页）

限于词表面的联系不同，该计算过程充分考虑了词语间的语义关系。因此，文本聚类过程中使用本体来表示和处理文本，降低了聚类对象的维度，减小了聚类复杂度，提高了聚类效率和准确度。此外，该方法还具有较好的灵活性和适用性，可以实现对任意语言和任意领域文本的聚类。

参 考 文 献

[1] Sanguthevar R. Efficient parallel hierar chical-clustering algo - rithms [J]. IEEE Transactions on Parallel and Distributied Systems, 2005, 16(6): 497-502

[2] Li X G, Yu G, Wang D L. MMPClust: A skew prevention algorithm for model-based document clustering[C]// Proceedings of the 10th International Conference on Database Systems for Advanced Applications(DASFAA'05). 2005: 536-547

[3] Miller G. WordNet: A lexical database for english[J]. Communications of the ACM (CACM'95), 1995, 38(11): 39-41

[4] Agirre E, Rigau G. Word sense disambiguation using conceptual density[C]// Proceedings of the 16th Conference on Computational Linguistics (COLING'96). 1996: 16-22

[5] Dario B, Fulvio C, Federico P. Automatic learning of text-to-concept mappings exploiting WordNet-like lexical networks [J]. ACM, 2005: 1639-1644

[6] Kolte S G, Bhirud S G. Word Sense Disambiguation Using WordNet Domains[J]. ACM, 2008: 1187-1191

[7] Sedding J, kazakov D. Wordnet-based text document clustering [C]// Proceedings of the Third Workshop on Robust Methods in Analysis of Natural Language Data (ROMAND'04). 2004: 104-113

[8] Hotho A, Staab S, Stumme G. Ontologies improve text document clustering[C]// Proceedings of the Third IEEE International Conference on Data Mining(ICDM'03). 2003: 541-544

[9] Hotho A, Maedche A, Staab S. Ontology-based Text Document Clustering[J]. Kunstliche Intelligenz, 2002, 16(4): 48-54

[10] Xie H W, Yan X L, Yu X L. Research on Web Page Clustering Based on Ontology [J]. Computer Science, 2008, 35(9): 153-155

[11] Li P, Tao L, Wang B Z. Measuring semantic similarity in ontology and its application in information retrieval [J]. Computer Engineering and Design, 2007, 28(1): 227-229

[12] Salton G, Buckley C. Term-weighting Approaches in Automatic Text Retrieval [J]. Information Processing and Management, 1988, 24(5): 513-523

[13] David D L, Yang Y, Fan L. RCV1: A New Benchmark Collection for Text Categorization Research[J]. Journal of Machine Learning Research, 2004, 5: 361-397

[14] Krishnamurthy B. On stationarity in Internet measurements th - rough an information-theoretic lens [C] // Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE'05). 2005: 1185-1185

[15] Strehl A, Ghosh J. Cluster ensembles-a knowledge reuse framework for combining partitions[J]. Journal of Machine Learning Research, 2002, 3: 583-617

(上接第 187 页)

的特点实施 DM XML 的结构连接，迅速完成 Twig 查询。

在模式解析中，可能得到许多无关的简单路径，如图 4 中的 A/B/D/D/D∈ A(n_2. Path)，该路径下的所有元素绝不会作为输出元素。另外，在模式解析完成后，多个 $P^{//,[]}$ 子类树模式间可能存在多个相同路径而造成部分子模式结构相同。因此，如何消除结构约束节点的无效路径来减少访问数据库的次数，如何充分利用多个解析模式之间的相同结构来减少结构连接的次数，是将来的研究方向。

参 考 文 献

[1] Miklau G, Suciu D. Containment and equivalence for a fragment of XPath [J]. Journal of the ACM, 2004, 51(1): 2-45

[2] 高军，杨冬青，唐世渭，等. 基于树自动机的 XPath 在 XML 数据流上的高效执行[J]. 软件学报，2005，16(2)：223-232

[3] Busse R, Carey M, Florescu D, et al. Xmark: An XML benchmark project. 2003[EB/OL]. http://monetdb. cwi. nl/xml/index. html

[4] Wang W, Wang H Z, Lu H J, et al. Efficient processing of XML path queries using the disk-based F&B index[C]// Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB). 2005: 145-156

[5] Chen Ting, Lu Jiaheng, Tok W L. On boosting holism in XML twig pattern matching using structural indexing techniques[C]// Proc. of ACM SIGMOD Int'l Conf. on Management of data. 2005: 45-46

[6] 周军锋，孟小峰，蒋瑜，等. F-Index：一种加速 Twig 查询处理的扁平结构索引[J]. 软件学报，2007，18(6)：1429-1442

[7] Lu J H, Tok W L, Chart C Y, et al. From region encoding to extended dewey: On efficient processing of xml twig pattern matching[C]// Proc. of VLDB05. 2005: 193-204

[8] Shanmugasundaram J, et al. Relational databases for querying XML documents: limitations and opportunities [C] // Proc. of VLDB. 1999: 302-314

[9] Peter T W. Minimizing simple XPath expressions[C]// Proc. of the 4th Int'l Workshop on Web and Databases (WebDB 2001). ACM Press, 2001: 13-18

[10] 李国良，冯建华，塔娜，等. TwigStar——快速处理 XML Twig 查询中含通配符 * 的算法[J]. 计算机研究与发展，2006，43(增刊)：430-437

[11] Milo T, Suciu D. Index structures for path expressions[C]// Proc. of the Int'l Conf. on Database Theory (ICDT). Springer-Verlag, 1999: 277-295

[12] Arion A, Bonifati A, Manolescu I, et al. Path Summaries and Path Partitioning in Modern XML Databases[J]. World Wide Web, 2008, 11(1): 117-151