ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.11, November 2009, pp.2977−2987
doi: 10.3724/SP.J.1001.2009.03422

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

XML 结构完整性约束下的路径表达式的最小化*

张剑妹[1,2], 陶世群[1+], 梁吉业[1]

[1](山西大学 计算机与信息技术学院,山西 太原 030006)

[2](长治学院,山西 长治 046011)

Minimization of Path Expression Under Structural Integrity Constraints for XML

ZHANG Jian-Mei[1,2], TAO Shi-Qun[1+], LIANG Ji-Ye[1]

[1](School of Computer and Information Technology, Shanxi University, Taiyuan 030006, China)

[2](Changzhi University, Changzhi 046011, China)

+ Corresponding author: E-mail: jmzhang@sxu.edu.cn, http://www.sxu.edu.cn

Zhang JM, Tao SQ, Liang JY. Minimization of path expression under structural integrity constraints for XML. *Journal of Software*, 2009,20(11):2977−2987. http://www.jos.org.cn/1000-9825/3422.htm

Abstract: A system of structural integrity constraints for XML (XSICs) is introduced, which specifies five structural relationships between different paths or nodes in XML documents, including path implication, path cooccurrence, path mutual-exclusion, obligatory inclusion and exclusive inclusion. This paper defines the syntax and semantics of these XSICs, and studies their core role in XML query optimization. Based on the concept of sub-path, this paper proposes an algorithm for minimizing path expression in the presence of XSICs. By using the path implication closure as a tool, the algorithm cannot only effectively eliminate redundant nodes or predicates, but also identify invalid path expressions. Experimental results show the effectiveness and efficiency of the proposed minimization algorithm.

Key words: path expression; structural integrity constraint for XML; path implication closure; sub-path; minimization

摘 要: 引入了一个 XML 结构完整性约束体系.这个体系描述了 XML 文档中节点或路径之间的 5 种结构关系,包括路径蕴涵、路径同现、路径互斥、必需性包含和排他性包含.给出了这些结构完整性约束的语法和语义定义,并研究了它们在 XML 查询优化中的作用.基于子路径的概念,提出了有结构完整性约束的路径表达式的最小化算法.该算法以路径蕴涵闭包为工具,不仅可以删除路径表达式的冗余,还可以识别无效路径表达式.实验结果表明了该算法的正确性和有效性.

关键词: 路径表达式;XML 结构完整性约束;路径蕴涵闭包;子路径;最小化

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.70471003 (国家自然科学基金); the Research Foundation for the Doctoral Program of the Ministry of Education of China under Grant No.20050108004 (国家教育部高等学校博士学科点专项科研基金)

Received 2007-11-06; Revised 2008-01-29, 2008-04-30; Accepted 2008-07-09

随着可扩展标记语言 XML 迅速成为 Web 上数据表示和数据交换的标准,XML 数据查询变得越来越重要.为此,学者们提出了多种 XML 查询语言,例如 Xpath,Xquery,XML-QL.这些语言的共同特点是利用路径表达式来导航 XML 文档的查询.XML 文档被看成节点标签的树,包含{/,//,[]}的路径表达式也可以表示成树形结构,这种树形结构称为树模式.路径表达式查询是寻找其树模式在 XML 文档树中的全部匹配,匹配操作的效率极大地依赖于树模式的大小[1],因此,路径表达式的最小化是一个极其重要的查询优化策略.

路径表达式最小化的实质是删除表达式中的冗余节点类型或谓词,通常有两种冗余:固有的冗余和依赖于文档结构的冗余.前者可以通过树模式分枝之间的包含测试来删除;要有效地删除后者,通常需要有一种表示文档结构的模式机制的支撑.目前,常用的 XML 文档模式是 DTD[2]和 XML Schema[3],它们规定了文档中允许出现的元素、属性及其嵌套关系,但不能处理路径之间复杂的结构关系,如在一个文档中从一个节点出发的路径要求(或排斥)另一个路径.文献[1,4]虽然提出了有结构约束的路径表达式的最小化算法,但其约束仅限于孩子约束、后代约束和子类型约束,没有考虑节点或路径之间复杂的结构关系,而这种结构关系为路径表达式的最小化提供了更多机会.本文考虑 XML 结构完整性约束下的路径表达式的最小化,主要工作如下:

(1) 引入了 XML 结构完整性约束的概念,并定义了排他性包含、必需性包含、路径蕴涵、路径互斥和路径同现的 5 种结构完整性约束的语法和语义;
(2) 提出了子路径的概念,并给出一个路径蕴涵闭包的计算算法;
(3) 基于子路径的概念,给出了有结构完整性约束情况下的路径表达式的最小化算法;
(4) 进行了一系列实验以验证最小化算法的正确性和有效性.

1 背景知识

1.1 XML文档

XML 文档通常被看成节点标签的树,假定存在 3 个互不相交的集合:E 是元素类型集合,A 是属性名集合,单元素集{S}表示文本,XML 文档树的形式化定义[5]如下:

定义 1. 一个 XML 文档树 T 被定义为一个六元组 $T=(V,lab,ele,att,val,root)$,其中:

① V 是节点的有限集.

② lab 是一个从 V 到 $E\cup A\cup\{S\}$的映射.每个 $v\in V$,若 $lab(v)\in E$,则 v 是一个元素;若 $lab(v)\in A$,则 v 是一个属性;若 $lab(v)=S$,则 v 是一个文本节点.

③ ele(或 att)是一个从 V 到 V^*的部分映射.每个 $v\in V$,若 $lab(v)\in E$,则 $ele(v)$(或 $att(v)$)是一个元素和文本节点序列(或属性集).每个 $v'\in ele(v)$(或 $v'\in att(v)$),v'称为 v 的子元素(或属性)且有一个从 v 到 v'的边.

④ val 是一个从 V 到字符串的部分映射.每个 $v\in V$,若 $lab(v)\in A$ 或 $lab(v)=S$,则 $val(v)$是一个字符串.

⑤ $root$ 是文档树中唯一的根节点.

图 1 是一个 XML 文档树.为了表示简单,图中忽略了文本值.本文所有的例子都建立在该文档上.我们对属性和元素采用相同的处理方法,因此不严格区分属性和元素,下文的节点类型泛指元素类型或属性名.

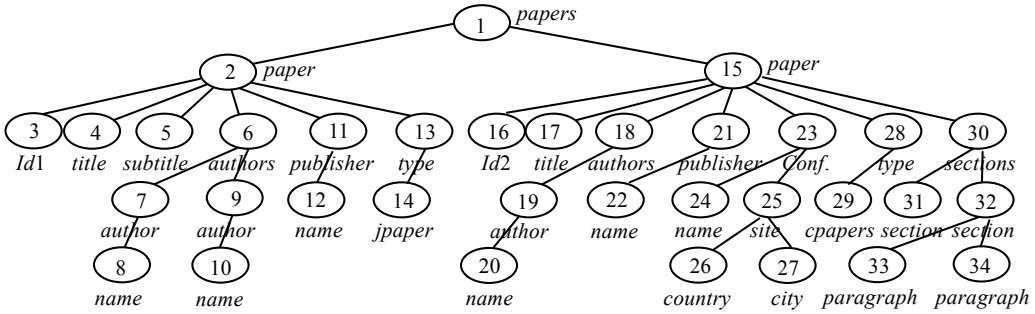

Fig.1 An example of XML document tree

图 1 XML 文档树示例

1.2 路径表达式及其相关概念

本文的路径表达式指 XPath 片断 $XP^{\{/,//,[]\}}$上的表达式,其语法定义如下:

$$P ::= \varepsilon \mid e \mid . \mid / P \mid // P \mid P / P \mid P // P \mid P[P],\ e \in E \cup A,$$

其中,“ε”表示空路径,“.”表示当前上下文节点,“/”和“//”分别表示直接包含(父-子)关系和间接包含(祖先-后代)关系,由“[”和“]”括起来的表达式是谓词.图 2 给出了一些路径表达式的例子.路径表达式中不可能出现冗余的内容谓词,因此,本文不考虑带内容谓词的路径表达式的最小化.不包含“[”和“]”的路径表达式称为线性路径表达式,下文用 $P \in LP$ 表示 P 是线性路径表达式.

定义 2. $\forall P,Q \in LP$,设 $P=c_1e_1c_2e_2\ldots c_ne_n$, $Q=c'_1e'_1c'_2e'_2\ldots c'_me'_m$,若 $n \geq m, c_i=c'_i$ 且 $e_i=e'_i$ (i=1,…,m),则 Q 是 P 的路径前缀,记为 $Q \prec P$;当 $n=m$ 时,称 P 等于 Q,记为 $P=Q$.

路径表达式可以表示成树模式,如图 3 所示.其中:单线边称为孩子边,表示直接包含关系;双线边称为后代边,表示间接包含关系;与路径表达式最后一个节点类型(谓词除外)相对应的节点称为选择节点,用星号标识选择节点;其他分枝与谓词相对应.树模式中,每个从根到叶的路径对应一个线性路径表达式,从根到选择节点对应的线性路径表达式称为选择路径,其他线性路径表达式称为条件路径[6].如 Q_2 的选择路径是“/*papers*/*paper*/*title*”,条件路径是“/*papers*/*paper*//*author*/*name*”.一个路径表达式只有 1 个选择路径,可有若干个条件路径.

Q_1=/*papers*/*paper*[*authors*/*author*/*name*]/*subtitle*
Q_2=/*papers*/*paper*[.//*author*/*name*]/*title*
Q_3=/*papers*/*paper*[*publisher*/*name*][.//*author*/*name*]//*section*
Q_4=/*papers*/*paper*[*type*/*conpaper*]//*author*
Q_5=/*papers*/*paper*[.//*section*][.//*paragraph*]/*title*
Q_6=/*papers*/*paper*[*conference*][*type*/*jourpaper*]/*title*

Fig.2 Examples of path expression
图 2 路径表达式举例

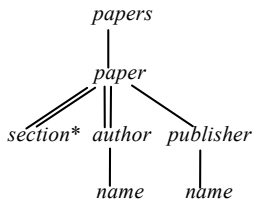

Fig.3 Tree pattern of Q_3
图 3 Q_3 的树模式

路径表达式的计算是寻找其树模式在文档树中的全部匹配.匹配的形式化定义如下:

定义 3. 给定树模式 Q 和文档树 T,N_q 和 N_t 分别表示 Q 和 T 的节点集,Q 在 T 上的匹配是一个映射 $f:N_q \rightarrow N_t$,该映射满足如下条件:

① 保持节点类型,即$\forall v \in N_q$,若 $lab(v)=e$,则 $lab(f(v))=e$.

② 保持边关系,即$\forall u,v \in N_q$,若(u,v)是孩子边,则 $f(v)$是 $f(u)$的孩子;若(u,v)是后代边,则 $f(v)$是 $f(u)$的后代.

文中用到的其他符号及其意义如下:$|P|$表示 P 中节点类型的个数;$nodes(P)$表示 P 的节点类型集;$last(P)$表示 P 的最后一个节点类型;设 u 是文档 T 中的一个节点,$u[P]$表示从 u 开始沿着 P 所指定的路径访问到的节点集.

1.3 子路径

定义 4. $\forall P,Q \in LP$,N_p 和 N_q 分别表示 P 和 Q 的树模式的节点集,Q 是 P 的子路径,记为 $Q \leq P$,当且仅当存在一个满足如下条件的映射 $f:N_q \cup \{root\} \rightarrow N_p \cup \{root\}$:

① 保持节点类型,即$\forall v \in N_q$,若 $lab(v)=e$,则 $lab(f(v))=e$.

② 保持边关系,即$\forall u,v \in N_q \cup \{root\}$,若$(u,v)$是孩子边,则在 P 中 $f(v)$是 $f(u)$的孩子节点;若(u,v)是后代边,则在 P 中 $f(v)$是 $f(u)$的后代节点.

例如,线性路径表达式“//*paper*//*section*”是“/*papers*/*paper*//*section*//*paragraph*”的一个子路径.显然,任何路径表达式 P 及其路径前缀都是其自身的子路径,ε是任何路径的路径前缀和子路径.

1.4 路径表达式的包含

用 $Q(T)$表示路径表达式 Q 在文档树 T 中的查询结果,路径表达式的包含与等价的定义如下:

定义 5. 如果对任何 XML 文档 T 都有 $Q_1(T) \subseteq Q_2(T)$成立,则 Q_2 包含 Q_1,记作 $Q_1 \subseteq Q_2$.

定义 6. 如果 $Q_1 \subseteq Q_2$ 且 $Q_2 \subseteq Q_1$,则 Q_1 和 Q_2 等价,记作 $Q_1 \equiv Q_2$.

子路径和包含既有区别又有联系.例如"//*paper*//*section*"是"/*papers*/*paper*//*section*/*paragraph*"的子路径,但其查询结果不相交.任意两个线性路径表达式 P 和 Q,若 $Q\leq P$ 且 $last(P)=last(Q)$,则 $P\subseteq Q$;反之亦然.

2 XML 文档的结构完整性约束

描述元素或路径之间结构关系的完整性约束称为 XML 结构完整性约束(简称为 XSICs).XSICs 分为两类:基于路径的 XSICs 和基于元素的 XSICs.

2.1 基于路径的XSICs

基于路径的 XSICs 描述了 XML 文档树中路径之间的蕴涵、互斥和同现关系(简称路径约束),定义如下:

定义 7. 给定文档树 T,$\forall C,P,Q\in LP,u,v,v'\in V$,则,

① 若$\forall u\forall v(u\in root[C]\wedge v\in u[P]\rightarrow\exists v'(v'\in u[Q]))$,则称 P 蕴涵 Q,记作 $C(P\rightarrow Q)$,T 满足 $C(P\rightarrow Q)$,记作

$$T\models C(P\rightarrow Q).$$

② 若$\forall u\forall v(u\in root[C]\wedge v\in u[P]\rightarrow\exists v'(v'\in u[Q]))$且$\forall u\forall v(u\in root[C]\wedge v\in u[Q]\rightarrow\exists v'(v'\in u[P]))$,则称 P 与 Q 互斥,记作 $C(P\nrightarrow Q)$,T 满足 $C(P\nrightarrow Q)$,记作 $T\models C(P\nrightarrow Q)$.

③ 若 $C(P\rightarrow Q)\wedge C(Q\rightarrow P)$,则称 P 与 Q 同现,记作 $C(P\leftrightarrow Q)$,T 满足 $C(P\leftrightarrow Q)$,记作 $T\models C(P\leftrightarrow Q)$.其中:

- 路径 C 为上下文路径,它指定路径约束关系在哪个文档子树成立.规定当 $C=\varepsilon$ 时,特定的路径约束关系在整个文档树上成立,且规定上下文路径 ε 可以被缺省;
- P 和 Q 分别称为约束的左部路径(left hand path,简称 LHP)和右部路径(right hand path,简称 RHP);C 是 P 和 Q 的公共前缀;当 $|P|=|Q|=|C|+1$ 时,路径蕴涵、互斥和同现表示兄弟节点之间的蕴涵、互斥和同现.

除非特别声明,在路径约束 $C(P\circ Q)(\circ\in\{\rightarrow,\leftrightarrow,\nrightarrow\})$中,$P$ 和 Q 都是以 C 为路径前缀的表达式.为书写简单,在约束实例中通常将 P 和 Q 简写为相对于 C 的相对路径.例如,表 1 中的 $\varphi_1-\varphi_7$ 是图 1 所示的 XML 文档满足的路径约束,φ_1 是/*papers*/*paper*(/*papers*/*paper*/*title*→/*papers*/*paper*//*author*/*name*)的缩写形式,其他约束与此类似.

Table 1 XSICs for the papers document

表 1 Papers 文档的 XSICs

Constraint type	Constraints
Path constraints	φ_1: /*papers*/*paper*(*title*→.//*author*/*name*) φ_2: / *papers* / *paper*(*type* / *jourpaper* ↛ *conference*) φ_3: // *paper*/*conference*(*name*→*site*/*city*) φ_4: /*papers*/*paper*(*type*/*conpaper*→ *conference*) φ_5: /*papers*/*paper*/*authors*(*email*→*author*/*name*) φ_6: /*papers*/*paper*/*authors*(*phone*→*author*/*name*) φ_7: /*papers*/*paper*(.//*author*↔*publisher*)
Element inclusion constraints	φ_8: *publisher*⇒*name* φ_9: *section*⇒*paragraph* φ_{10}: *paper*⇒*author* φ_{11}: *author*⇒*name* φ_{12}: *paper* ↦ *author* φ_{13}: *conference* ↦ *city*

通俗地讲,路径蕴涵 $C(P\rightarrow Q)$表示在 C 所确定的文档子树上,如果 P 出现,则 Q 一定出现;路径同现是双向的路径蕴涵;路径互斥 $C(P\nrightarrow Q)$ 表示在 C 所确定的文档子树上,P 与 Q 不能同时出现.根据其语义,路径互斥 $C(P\nrightarrow Q)$ 可表示为 $C((P\rightarrow\neg Q)\wedge(Q\rightarrow\neg P))$,其中,$\neg P$ 表示 P 不存在.

给定一个线性路径表达式集 B 和一个线性路径表达式 P,$P\propto B$ 当且仅当存在一个线性路径表达式 Q 使得 $Q\in B$ 且 $P\leq Q$;$\neg P\propto B$ 当且仅当存在一个线性路径表达式 Q 使得 $\neg Q\in B$ 且 $Q\leq P$.

2.2 基于元素的XSICs

2.2.1 排他性包含和必需性包含的语法与语义

基于元素的 XSICs 描述节点之间的蕴涵、互斥、同现和结构包含关系.如前所述,节点之间的蕴涵、互斥和同现是路径约束的特例,因此,这里只定义节点之间的结构包含关系.根据其在查询优化中的不同作用,将节点之间的结构包含分为排他性包含和必需性包含(简称为元素包含约束),定义如下:

定义 8. 给定文档树 T,$\forall P,Q\in LP,e_i\in E,e_j\in E\cup A,u,v,v_1,v_2\in V$,则,

① 若$\forall u(lab(u)=e_i\rightarrow\exists v(lab(v)=e_j\wedge v\in u[P]))$,则称 e_i 必需包含 e_j,记作 $e_i\Rightarrow e_j$,T 满足 $e_i\Rightarrow e_j$,记作 $T\models e_i\Rightarrow e_j$.

② 若$\forall u(lab(u)=e_i\wedge\exists v_1\exists v_2(v_1\in u[P]\wedge v_2\in u[Q]\wedge lab(v_1)=lab(v_2)=e_j)\rightarrow P=Q)$,则称 e_i 排他包含 e_j,记作 $e_i\mapsto e_j$,T 满足 $e_i\mapsto e_j$,记作 $T\models e_i\mapsto e_j$.

换言之,$e_i\Rightarrow e_j$ 是指任何一个类型为 e_i 的节点(e_i 节点)必须有 e_j 后代;$e_i\mapsto e_j$ 表示若一个 e_i 节点有一个 e_j 后代,则从 e_i 节点到 e_j 节点的访问路径是唯一的.如表 1 中的φ_8~φ_{13} 是图 1 的 XML 文档满足的元素包含约束.

定理 1. 设 E 是元素类型集,A 是属性名集,$\forall e_i,e_j\in E,\forall e_k\in E\cup A$,若 $e_i\Rightarrow e_j$ 且 $e_j\Rightarrow e_k$,则 $e_i\Rightarrow e_k$.

这个定理叙述了必需性包含的传递性,其正确性是显然的.

2.2.2 约束 chase 技术

在最小化过程中,用必需性包含删除表达式中的冗余叶节点,用排他性包含缩短线性路径表达式.此外,元素包含约束还可以转换成等价的路径蕴涵约束,根据其语义,$e_i\Rightarrow e_j$ 等价于$//e_i\rightarrow e_i//e_j$,$e_i\mapsto e_j$ 等价于$//e_i//e_j\rightarrow //e_ic_{i+1}e_{i+1}\dots c_je_j$.其中,$//e_ic_{i+1}e_{i+1}\dots c_je_j$ 是以“//”开头的中间不含“//”的线性路径表达式.在关系数据库中,通常采用 chase 技术改写查询,使其包含完整性约束的作用[7].为了相同的目的,改进的 chase 技术被应用到 XML 路径表达式最小化中[1,4,8].本文采用 chase 技术改写路径约束集,使其集成元素包含约束的作用.由于应用环境不同,不能照搬原有的 chase 技术.令 $P=c_1e_1c_2e_2\dots c_ke_k$,$\Omega$为元素包含约束集,$\Sigma$为路径约束集,约束 chase 技术如下:

(1) 利用定理 1 计算必需性包含闭包 $cls(\Omega)$.

(2) 检查Σ中的每个路径表达式 P(约束的 LHP 和 RHP),若 $e_i\mapsto e_j\in cls(\Omega)$ 且 $e_i,e_j\in nodes(P)(i<j)$,则在Σ中增加约束 $c_1e_1\dots c_ie_i//e_j\dots c_ke_k\rightarrow P$;若 $e_i\Rightarrow e_x\in cls(\Omega)$且 $e_i\in nodes(P)\wedge e_x\notin nodes(P)(i=1,2\dots k)$,则在$\Sigma$中增加约束 $P\rightarrow c_1e_1c_2e_2\dots c_ie_i//e_x$;若 $e_{k-1}\Rightarrow e_k\in cls(\Omega)$且 $e_{k-1},e_k\in nodes(P)$,则在Σ中增加约束 $c_1e_1c_2e_2\dots c_{k-1}e_{k-1}\rightarrow P$.

(3) 重复步骤(2),直到Σ不发生变化为止.

2.3 路径蕴涵闭包

路径蕴涵和路径同现用来识别路径表达式中的冗余谓词,如在没有完整性约束时,Q_2 为最小查询,若已知 /*papers*/*paper*(*title*→.//*author*/*name*),则条件路径“/*papers*/*paper*//*author*/*name*”是冗余的,该查询被最小化为“/*papers*/*paper*/*title*”.查询结果为空的路径表达式为无效路径表达式,路径互斥用来识别无效路径表达式.本文以路径蕴涵闭包作为查询最小化和无效路径表达式识别的主要工具.下面给出路径蕴涵的推理规则:

规则 1. $C(P\rightarrow Q),C'\prec C\Rightarrow C'(P\rightarrow Q)$.

规则 2. $C(P\rightarrow Q),C'\subseteq C\Rightarrow C'(P\rightarrow Q)$.

规则 3. $\forall P,Q\in LP,Q\leq P\Rightarrow P\rightarrow Q$.

规则 4. $C(P\rightarrow Q),C(Q\rightarrow S)\Rightarrow C(P\rightarrow S)$.

规则 5. $C(P\rightarrow Q),C(Q\rightarrow\neg S)\Rightarrow C(P\rightarrow\neg S)$.

规则 6. $C(P\rightarrow\neg Q),C(S\rightarrow Q)\Rightarrow C(P\rightarrow\neg S)$.

规则 7. $C(P\rightarrow\neg Q),C\leq C'\Rightarrow C'(P\rightarrow\neg Q)$.

规则 1~规则 7 是正确的和完备的.规则的正确性可以根据路径约束的语义得到证明.规则的完备性可以通过如下步骤得到证明:给定约束φ:$C(P\rightarrow Q)$(或 $C(P\rightarrow\neg Q)$),首先计算 P 的路径蕴涵闭包 B,然后构造一个文档树 T,使 B 中除 Q(或$\neg Q$)以外的路径都出现在 T 中.限于篇幅,本文不给出详细证明过程.

定义 9. 给定约束Σ和 $C,P\in LP$,P 关于Σ和 C 的路径蕴涵闭包是在Σ和 C 下 P 所蕴涵的全部路径表达式的最小集合,记为 $P^+_{(\Sigma,C)}$,即 $P^+_{(\Sigma,C)}=\{X|C(P\rightarrow X)$能够使用规则 1~规则 7 从$\Sigma$推出,其中,$X\in\{Q,\neg Q\}$.

设Ω是元素包含约束集,Σ是改写并 chase 后的路径约束集,计算路径蕴涵闭包的算法如下:

算法 1. *ClosureComp*(Σ,C,P).

输入:非空约束集Σ,$C,P\in LP$.

输出: $P^+_{(\Sigma,C)}$.

B={P}; /*以下代码对 B 进行初始化*/
if (e_i∈*nodes*(P) && e_i⇒e_x∈Σ)
  if e_i=*last*(P) **then** B={P//e_x}; **else** B=B∪{$c_1e_1c_2e_2\ldots c_ie_i$//$e_x$};
repeat checking each X∈B **until** all path expressions in B are checked /*以下代码计算 B*/
  for each φ:C'(P'→Q')∈Σ
    if (X==S) then /*X==S 表示 X 是形如 S 的线性路径表达式*/
    if ($C \prec C'$) **then**
      if (P'≤S && $Q' \not\leq S$) **then** B=B∪{Q'}; /*规则 1,规则 3,规则 4*/
    else if (C'≤S && *last*(C')∉*nodes*(C)–*last*(C) && P'≤S) **then**
      replace the prefix C' of Q' with $C_S^{lC'}$; /* $C_S^{lC'}$ 是 S 上从第 1 个节点类型到 *last*(C')的子路径*/
      if ($Q' \not\leq S$) **then** B=B∪{Q'}; /*规则 2,规则 3,规则 4*/
    if (X==¬S) **then** /*X==¬S 表示 X 是形如¬S 的线性路径表达式*/
      if ($C \prec C'$ && S≤Q') **then** B=B∪{¬P'}; /*规则 1,规则 3,规则 6*/
      else if (C'≤S && *last*(C')∉*nodes*(C)–*last*(C)) **then**
        replace the prefix C' of P' and Q' with $C_S^{lC'}$;
        if (S≤Q') **then** B=B∪{¬P'} /*规则 2,规则 3,规则 6*/
  for each φ:C'(P'→¬Q')∈Σ
    if (X==S) **then** /*规则 3,规则 5,规则 7*/
      if (C'≤C && P'≤S) **then** B=B∪{¬Q'};
      else if (C'≤S && P'≤S) **then** B=B∪{¬Q'}; tag ¬Q' with $C_S^{lC'}$;
  for each P∈B /*以下代码删除 B 中的冗余路径表达式*/
    if (Q∈B && Q≤P) **then** B=B–{Q}
  for each ¬P∈B
    if (¬Q∈B && P≤Q) **then** B=B–{¬Q}
return B

该算法通过判断两个线性路径表达式的子路径关系来检查 B 中的路径是否蕴涵Σ中路径蕴涵的 RHP,最多执行 $O(|B|\times|\Sigma|)$次检查.在不出现递归的情况下,子路径判断的时间复杂度为两条线性路径长度之和(记为 l).此外,最坏情况下,$|B|$等于$|\Sigma|$.因此,该算法的复杂度为 $O(l\times|\Sigma|^2)$.

3 路径表达式的最小化

如前所述,路径表达式中有两类冗余.本文使用子路径识别和删除路径表达式中固有的冗余,使用路径蕴涵闭包识别和删除依赖于文档结构的冗余.

3.1 利用子路径识别冗余条件

定理 2. 给定路径表达式查询 Q,令 P_i 是 Q 的任意条件路径,P_j 是 Q 的选择路径或另一条件路径,Q–P_i 表示删除条件路径 P_i 后得到的路径表达式查询,若 P_i≤P_j,则 Q–P_i≡Q.

证明:若 P_i≤P_j,根据子路径和匹配的定义可知,P_j 在文档 T 中的每个匹配都包含一个 P_i 在 T 中的匹配.因此,条件路径 P_i 是冗余的,故 Q–P_i≡Q. □

该定理说明了如何利用子路径识别冗余条件路径.此外,必需性包含使一些隐含的子路径关系显现出来.例如,无约束时,查询 Q_5 是最小查询;注意到"/*papers*/*paper*//*paragraph*"是"/*papers*/*paper*//*section*/*paragraph*"的子路径,若已知 *section*⇒*paragraph*,则 Q_5 的条件路径"/*papers*/*paper*//*paragraph*"是冗余的.为了识别这类冗余,首先判断一个条件路径 P_1 删除最后一个节点类型后是否是另一条件路径 P_2 的子路径;若结果为真,则再判断 P_2 中的节点类型是否必需包含这个被删除节点类型.算法如下:

算法 2. *IsSubPathOb*(P_1,P_2).

输入:两个不同的线性路径表达式 P_1,P_2.

输出:false 或 true.

m=|P_1|; n=|P_2|;

if (m>n+1) **then** return false;

else if ($P_1^{m-1} \le P_2$) **then** /* P_1^{m-1} 为 P_1 截取最后一个节点类型后形成的路径*/

if (($e_{1(m-1)}$,e_{1m})∈*Cedge*(P_1)) **then**

if ($e_{2m'}$∈*nodes*(P_2) && $e_{2m'}$⇒e_{1m}) **then** return true; /*$e_{2m'}$是 P_2 中与 P_1 中的 $e_{1(m-1)}$相匹配的节点类型*/

else for each e_{2i}∈*nodes*(P_2)(i=m' to n)

if (e_{2i}⇒e_{1m}) **then** return true;

return false

3.2 利用路径约束识别冗余条件和无效路径表达式

定理 3. 给定一个路径约束集Σ,令 P_i 是路径表达式查询 Q 的任意条件路径,P_j 是 Q 的选择路径或另一条件路径,C 为 P_i 和 P_j 的最长公共路径前缀,若 $P_i \propto P^+_{j(\Sigma,C)}$,则 Q–P_i≡Q.

证明:由 $P_i \propto P^+_{j(\Sigma,C)}$ 得 $P_i \in P^+_{j(\Sigma,C)}$ 或者 $P_i' \in P^+_{j(\Sigma,C)}$ 且 $P_i \le P_i'$;根据定义 9 和规则 3、规则 4 可知 $P_j \to P_i$ 成立,即在 C 确定的文档子树上,P_j 出现时 P_i 必出现,因此 P_i 是冗余的,故 Q–P_i≡Q 成立. □

定理 3 说明如何利用路径蕴涵闭包识别冗余条件路径.此外,路径蕴涵闭包还可用来识别无效路径表达式.

定理 4. 给定一个路径约束集Σ,令 P_i 和 P_j 是路径表达式查询 Q 的任意条件路径或选择路径,C 为 P_i 和 P_j 的最长公共路径前缀,若¬$P_i \propto P^+_{j(\Sigma,C)}$ (或¬$P_j \propto P^+_{i(\Sigma,C)}$),则 Q 是无效路径表达式.

证明:由¬$P_i \propto P^+_{j(\Sigma,C)}$ 得¬$P_i \in P^+_{j(\Sigma,C)}$ 或者 ¬$P_i' \in P^+_{j(\Sigma,C)}$ 且 $P_i' \le P_i$;根据定义 9 和规则 3、规则 6 可得 $P_j \to \neg P_i$,即在 C 确定的文档子树上,P_j 出现时 P_i 必不出现,也即这两个条件路径是不相容的,故 Q 是无效路径表达式. □

3.3 路径表达式最小化算法

算法由 3 部分组成:分解、最小化和重构.分解算法将查询分解成选择路径和条件路径,最小化算法删除查询中的冗余,重构算法将非冗余条件路径按原来结构关系连接到选择路径上.下文着重描述最小化算法.算法首先利用子路径删除冗余条件路径,然后使用必需性包含递归地删除非冗余条件路径的冗余叶节点,最后利用路径蕴涵闭包删除冗余条件路径.给定查询 Q,令 P_s 为选择路径,P[i](i=1,…,m)为条件路径,m 为 Q 中的条件路径数,C 为 Q 中最高分支点的路径,*PCnt* 为未删除的条件路径数,算法如下:

算法 3. Min*XPath*(P_s,P[]).

输入:选择路径 P_s 和条件路径 P[i],非空约束集Σ.

输出:选择路径 P_s 和非冗余的条件路径 P[j].

/*第 1 阶段:使用子路径删除冗余条件*/

PCnt=m;

for each condition path P[i] /*函数 *IsSubPath*(P_i,P_j)判断 P_i 是否 P_j 的子路径,算法略*/

if (*IsSubPath*(P[i],P_s)||*IsSubPathOb*(P[i],P_s)) **then** P[i]=ε; *PCnt*–1;

if (*PCnt*>1) **then**

for each non-empty condition path P[i]

for each non-empty condition path P[j]

if (P[i]≠P[j] && (*IsSubPath*(P[i],P[j])||*IsSubPathOb*(P[i],P[j]))) **then** P[i]=ε; *PCnt*–1;

/*第 2 阶段:使用必需性包含删除非冗余条件路径的冗余叶节点*/

if (*PCnt*>=1) **then**

for each non-empty condition path $P[i]$
  DelRedNode($P[i]$); /**DelRedNode*($P[i]$)删除 $P[i]$中的冗余叶节点,算法略*/
  if ($|P[i]|==|C|$) **then** $P[i]=\varepsilon$; $PCnt$−1;
/*第 3 阶段:使用路径蕴涵闭包删除冗余条件*/
if ($PCnt$>=1) **then**
  B=*ClosureComp*(Σ,C,P_s);
  for each non-empty condition path $P[i]$
    if ($\neg P[i]\propto B$) **then** return "The query result is empty";
    if ($P[i]\propto B$) **then** $P[i]=\varepsilon$; $PCnt$−1;
if ($PCnt$>1) **then**
  for each non-empty condition path $P[i]$
    B=*ClosureComp*(Σ,C,$P[i]$);
    for each non-empty condition path $P[j]$
      if ($P[i]\neq P[j]$ && $\neg P[j]\propto B$) **then** return "The query result is empty";
      if ($P[i]\neq P[j]$ && $P[j]\propto B$) **then** $P[j]=\varepsilon$; $PCnt$−1

该算法的复杂度由第 3 阶段的双重循环决定.算法首先计算一个非冗余条件路径的路径蕴涵闭包,需花费$O(l\times|\Sigma|^2)$时间;然后,算法检查其他非冗余条件路径是否属于该路径蕴涵闭包,最多执行 $m\times|\Sigma|$次检查,每次检查需 $O(l)$时间,共花费 $O(m\times l\times|\Sigma|)$时间;通常$|\Sigma|\geq m$,故每次循环花费 $O(l\times|\Sigma|^2)$时间;最坏情况下,非冗余条件路径数为 m,即最多执行 m 次循环,令 n 为查询的大小,则 l 的最大平均长度为 $2n/m$.因此,算法的复杂度为 $O(n|\Sigma|^2)$.

4 实验设计和性能分析

实验的硬件环境为奔腾 IV 2.4GHz,256M RAM,40G 硬盘,单 CPU 的 PC 机,操作系统为 Windows XP,所有代码用 VC6.0 编写,用微软的 MSXML4.0 DOM 解析器解析并装入文档,用 DOM API 进行数据查询.

实验数据集有:人工数据集 Papers(见图 1)和经典测试数据集 XMark.Papers 文档由我们设计的文档生成器根据其 DTD 自动生成,大小为 65.2MB,共 2 000 035 个节点,其中 1 245 905 个元素节点,65 639 个属性节点,688 491 个文本节点;XMark 文档由 xmlgen[9]选择 1.0 的缩放因子生成,大小为 113MB.根据文档的结构特征和应用需求,分别为这两个文档设计了 XSICs(见表 1 和表 2).约束被存储在随机文件中,最小化时,被预先读入内存.图 2 和图 4 分别给出了 Papers 文档和 XMark 文档的查询,需要说明,尽管有些查询在实际应用中出现的几率较小,但由于其代表了不同的冗余类型,最小化算法必须考虑到这些情况的出现.

Q_7=//*open_auctions*/*open_auction*[*bidder*/*date*][*quantity*]/*seller*
Q_8=/*site*//*closed_auction*[*buyer*]/*seller*
Q_9=//*people*/*person*[*phone*][*homepage*]/*name*
Q_{10}=/*site*/*people*/*person*[*address*][*address*/*province*]/*name*
Q_{11}=//*open_auction*[*initial*][*reserve*]/*type*
Q_{12}=/*site*//*closed_auction*[*buyer*][*seller*]

Fig.4 Queries for the XMark document
图 4 XMark 文档的查询

Table 2 XSICs for the XMark document
表 2 XMark 文档的 XSICs

Constraint type	Constraints
Path constraints	δ_1: /*site*/*closed_auctions*/*closed_auction*(*seller*→*price*) δ_2: /*site*/*closed_auctions*/*closed_auction*(*seller*→*buyer*) δ_3: /*site*/*open_auctions*/*open_auction*(*bidder*→*seller*) δ_4: /*site*/*open_auctions*/*open_auction*(*bidder*→*quantity*) δ_5: /*site*/*open_auctions*/*open_auction*(*reserve*→*initial*) δ_6: /*site*/*people*/*person*(*address*→*emailaddress*) δ_7: /*site*/*people*/*person*(*address*→*name*)
Element inclusion constraints	δ_8: *open_auction*⇒*seller* δ_9: *closed_auction*⇒*seller* δ_{10}: *closed_auction*⇒*buyer* δ_{11}: *bidder*⇒*date* δ_{12}: *address*⇒*city* δ_{13}: *address*⇒*street*

用本文的最小化算法优化查询,得到最小化算法的执行时间(MinTime)和最小化前后的查询响应时间(OrgQry time/MinQry time),统计最小化后的时间收益.对每个查询反复执行 6 次,取其平均时间得到表 3 和表 4 的实验结果.其中,时间缩短量(time red)=OrgQry time−(MinQry time+MinTime).从实验结果可以看出:对于没有冗余的查询,最小化算法会带来一些附加时间,但附加时间远远小于其查询时间,如查询 Q_4 和 Q_9;对于无效路径表达式,最小化算法在很短的时间内给出无查询结果的判断,从而大幅度地提高了查询效率,如查询 Q_6,最小化算法用约 1.5ms 的时间给出无查询结果的判断,若直接查询文档,则需 32ms 左右的时间得到 0 个查询结果;对有冗余的查询,最小化算法均不同程度地缩短了其查询响应时间,且缩短的时间远远大于最小化时间.图 5 和图 6 直观地显示了最小化后的时间收益.从图中可以看出,最小化后查询响应时间比原查询时间缩短了 10%~20%,缩短的查询响应时间是最小化时间的 10 倍~15 倍,可见,本文的最小化算法是有效的.

由表 3 和表 4 可以看出,不同查询的最小化时间相差较大,下面就查询 Q_1~Q_6 来分析引起这种差异的主要原因.最小化时间最短的是查询 Q_2,最小化时间在 1ms 以上的是查询 Q_5 和 Q_6.原因是 Q_2 只有 1 个较短条件路径,该条件路径在最小化的第 1 阶段即被删除,没有进行路径蕴涵闭包的计算;而 Q_5 和 Q_6 直到最小化的最后条件路径也没有被全部删除,在整个过程多次进行路径蕴涵闭包的计算;Q_1,Q_3 和 Q_4 仅计算选择路径的路径蕴涵闭包.由此可见,引起最小化时间变化的主要原因是路径蕴涵闭包的计算.

Table 3 Experimental results on the papers document

表 3 Papers 文档的实验结果

Original queries	Minimal queries	OrgQry time (ms)	MinTime (ms)	MinQry time (ms)	Time red (ms)
Q_1	/*papers*/*paper*[*authors*]/*subtitle*	0.522	34.506	28.863	5.121
Q_2	/*papers*/*paper*/*title*	0.185	34.216	27.474	6.557
Q_3	/*papers*/*paper*[*publisher*]//*section*	0.625	35.276	27.330	7.321
Q_4	Q_4	0.508	33.201	–	–
Q_5	/*papers*/*paper*[.//*section*]/*title*	1.062	34.216	26.882	6.272
Q_6	The query result is empty	1.446	31.969	–	–

Table 4 Experimental results on the XMark document

表 4 XMark 文档的实验结果

Original queries	Minimal queries	OrgQry time (ms)	MinTime (ms)	MinQry time (ms)	Time red (ms)
Q_7	//*open_auctions*/*open_auction*[*bidder*]/*seller*	0.808	85.377	74.764	9.805
Q_8	/*site*//*closed_auction*/*seller*	0.421	70.870	61.385	9.064
Q_9	Q_9	0.893	86.643	–	–
Q_{10}	/*site*/*people*/*person*[*address*/*province*]/*name*	0.632	88.815	76.863	11.329
Q_{11}	//*open_auction*[*reserve*]/*type*	0.851	78.087	70.332	6.904
Q_{12}	/*site*//*closed_auction*	0.506	65.010	51.929	12.575

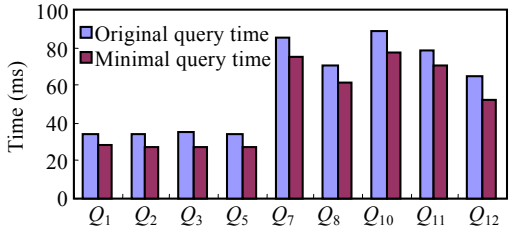

Fig.5 Response time of original queries and minimal queries

图 5 原查询与最小查询的响应时间

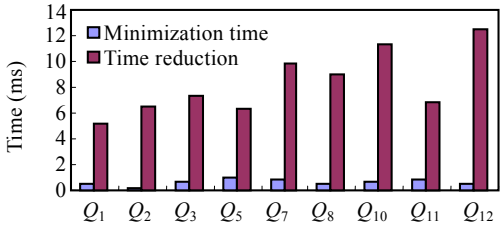

Fig.6 Comparison of minimization time and time reduction

图 6 最小化时间与节省时间的对比

5 相关研究与分析比较

路径表达式最小化的本质是查询包含,查询包含的主要技术是同态技术[10],但同态是查询包含的充分但非必要条件.为此,文献[11]提出了条件同态和隐含条件同态(hidden conditioned homomorphism)技术,基于这种技

术提出了更完备的包含测试算法.最先专门研究路径表达式最小化问题的是文献[1,12].基于同态的概念,文献[12]研究了包含{/,*,[]}的路径表达式的最小化,并得出复杂度为多项式的结论;文献[1]研究了不含通配符的树模式的最小化,并提出了多项式的算法.文献[4]把模拟的概念引入到最小化研究中,在与文献[1]相同的表达式上提出了更优的最小化算法.文献[8]对文献[1,4]的路径表达式进行了扩展,使其包含 descendant-or-self 轴,并针对父亲、孩子、后代和子类型约束提出了相应的最小化算法.文献[13]研究了包含通配符的树模式的最小化问题,并声明表达式的最小树模式可以从其子模式中找到,证明了这是一个 NP 完全问题.同时,在对表达式进行一定限制后提出了多项式的最小化算法.与前面的研究不同,文献[14]引入一个 layer axis 消除路径表达式中非冗余的通配符步,从而缩短了有通配符的表达式.文献[15]提出了两条路径缩短策略:缩短路径法和补路径法,其基本思想是用与查询等价的相对路径或互补路径取代原查询.文献[16]研究了路径蕴涵、互斥和同现 3 种结构约束及其逻辑蕴涵和可满足性,其约束是用简单路径定义的.文献[6]提出了使用 DTD 中隐含的结构约束优化 XPath 查询的方法,这种方法将不确定的路径步转换为确定路径步,增大了查询规模.文献[17]也研究了如何利用模式中隐含的约束最小化路径表达式,但其算法并不增加查询规模,为有模式的路径表达式优化提供了有效的解决方法.

本文考虑与文献[1,4]相同的表达式的最小化.然而,文献[1,4]的约束仅限于孩子、后代和子类型约束;XSICs是这些约束的扩展:必需性包含包括了孩子和后代约束,子类型约束是路径蕴涵的特殊情况.当没有完整性约束时,本文的算法通过测试线性路径表达式之间的子路径来删除表达式中的固有冗余(见算法 3 的第 1 阶段删除 If 条件中的 *IsSubPathOb*()函数),其复杂度为 $O(n^2)$;而文献[1,4]的复杂度分别为 $O(n^4)$和 $O(n^2)$.在有孩子和后代约束的情况下,只需执行算法 3 的前两个阶段,复杂度为 $O(n^2|\Sigma|)$,文献[1,4]的算法复杂度分别为 $O(n^6)$和 $O(n^2)$.本文的算法复杂度是在条件路径数(谓词数)m 等于查询大小 n 的假设下得到的,事实上,m 远远小于 n.因此,在第 1 种情况下,本文的算法优于文献[4]的算法;在第 2 种情况下,本文的算法也并不比文献[4]的算法差.另外,本文的算法将查询分解成线性路径表达式,通过测试线性路径表达式之间的包含关系来删除冗余条件路径,从而降低了包含测试的难度,使算法易于扩展到带有通配符的路径表达式.可见,本文的最小化算法是可行的.

6 结 论

路径表达式的最小化是XML查询优化的一个关键问题.基于子路径的概念,本文提出了有XSICs的路径表达式的最小化算法.与其他最小化算法不同,本文的算法以路径蕴涵闭包为主要工具,不仅能够有效地删除路径表达式中的冗余谓词和节点类型,而且可以识别无效路径表达式.算法的基本思想是,将路径表达式分解成若干线性路径表达式,通过测试线性路径表达式之间的包含关系删除冗余条件路径.根据文献[18]的研究成果,有通配符的线性路径表达式的包含是 PTIME 问题.如何扩展该算法,使其能够处理包含通配符的路径表达式,是我们将要解决的一个问题.XSICs 可被看成是对 DTD 的扩展,如何从 DTD 中获得其隐含的全部结构完整性约束,并用 XSICs 扩展所获得的约束以得到 DTD 中所有路径表达式的路径蕴涵闭包,将是今后要研究的一个重要课题.

References:

[1] Amer-Yahia S, Cho S, Lakshmanan LVS, Srivastava D. Minimization of tree pattern queries. In: Aref WG, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. Santa Barbara: ACM Press, 2001. 497−508.

[2] W3C. Extensible markup language (XML) 1.0. 3rd ed., W3C Recommendation, 2004. http://www.w3.org/TR/2004/REC-xml-20040204

[3] W3C. XML schema part 1: Structures. 2nd ed., W3C Recommendation, 2004. http:// www.w3.org/TR/xmlschema-1

[4] Ramanan P. Efficient algorithms for minimizing tree pattern queries. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 299−309.

[5] Buneman P, Davidson S, Fan WF, Hara C, Tan WC. Reasoning about keys for XML. Information Systems, 2003,28(8):1037−1063.

[6] Kwong A, Gertz M. Schema-Based optimization of XPath expressions. Technical Report, University of California at Davis, 2001. http://sirius.cs.ucdavis.edu/publications/KG02a.pdf

[7] Popa L, Deutsch A, Sahuguet A, Tannen V. A chase too far? In: Chen WD, Naughton JF, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. Dallas: ACM Press, 2000. 273−284.

[8] Liu XP, Wan CX. Minimization of XPath queries with constraints. Journal of Computer Research and Development, 2004, 41(Suppl.):342−348 (in Chinese with English abstract).

[9] Busse R, Carey M, Florescu D, Kersten M, Manolescu I, Schmidt A, Waas F. Xmark—An XML benchmark project. http://monetdb.cwi.nl/xml/index.html

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

[11] Feng JH, Liao YG, Zhang Y. HCH for checking containment of XPath fragment. Journal of Computer Science and Technology, 2007,22(5):736−748.

[12] Wood PT. Minimizing simple XPath expressions. In: Mecca G, Siméon J, eds. Proc. of the 4th Int'l Workshop on the Web and Database (WebDB 2001). Santa Barbara: ACM Press, 2001. 13−18.

[13] Flesca S, Furfaro F, Masciari E. On the minimization of XPath queries. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Base. Berlin: Morgan Kaufmann Publishers, 2003. 153−164.

[14] Chan CY, Fan WF, Zeng YM. Taming XPath queries by minimizing wildcard steps. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer B, eds. Proc. of the 30th Int'l Conf. on Very Large Data Base. Toronto: Morgan Kaufmann Publishers, 2004. 156−167.

[15] Wang GR, Sun B, Lv JH, Yu G. RPE query processing and optimization techniques for XML database. Journal of Computer Science and Technology, 2004,19(2):224−238.

[16] Kwong A, Gertz M. Structural constraints for XML. Technical Report, University of California at Davis, 2002. http://sirius.cs.ucdavis.edu/publications/KG02b.pdf

[17] Wang Y, Meng XF, Wang S. Schema based pattern tree semantic optimization. Journal of Computer Research and Development, 2004,41(Suppl.):355−362 (in Chinese with English abstract).

[18] Milo T, Suciu D. Index structures for path expressions. In: Beeri C, Buneman P, eds. Proc. of the 7th Int'l Conf. on Database Theory. Jerusalem: Springer-Verlag, 1999. 277−295.

附中文参考文献:

[8] 刘喜平,万常选.带约束 XPath 查询的最小化.计算机研究与发展,2004,41(增刊):342−348.

[17] 王宇,孟小峰,王珊.基于模式的 Pattern Tree 语义优化.计算机研究与发展,2004,41(增刊):355−362.

张剑妹(1970－),女,山西屯留人,博士,副教授,CCF 会员,主要研究领域为数据库技术,XML 数据管理.

梁吉业(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为粗糙集理论,数据挖掘,人工智能.

陶世群(1946－),男,教授,博士生导师,主要研究领域为数据库理论与技术,XML 数据管理.