有效的非完全结构 XML 查询

李晓光 于 戈 龚 剑 王大玲 鲍玉斌
(东北大学信息科学与工程学院 沈阳 110004)

摘 要 讨论了有意义的非完全结构 XML 查询(NFS)结果的定义,提出了一种易于扩展的基于模式和实体的有意义判断模型——PE 模型;基于 PE 模型,设计了具体的等价模式和等价查询项判断方法,提出了 PE 索引和 I2P 倒排索引,设计了一种有效的 NFS 查询算法来处理有意义的判断以及路径查询和关键字查询.实验表明,文中方法的查询质量和效率要优于 XSEarch 系统和 Timber 系统.

关键词 非完全结构 XML 查询;有意义的查询结果;结构索引;倒排索引
中图法分类号 TP18

Towards Effective and Efficient NFS Querying on XML Document

LI Xiao-Guang YU Ge GONG Jian WANG Da-Ling BAO Yu-Bin
(*School of Information Science and Engineering, Northeastern University, Shenyang* 110004)

Abstract This paper discusses the issue of meaningful result determination for non-fully structured query (NFS) and proposes a scalable PE model to determine meaningful results based on the concept of pattern and entity. Within the framework of PE model, the paper proposes the method of identifying equivalent patterns and equivalent terms, puts forward the PE index and the improved inverted index I2P, and develops the effective and efficient NFS query algorithm for meaningful result evaluation, and the path query and keyword-based query. Experimental results indicate that the authors' approach outperforms XSEarch system and Timber system both on the querying quality and the efficiency of query processing significantly.

Keywords non-fully structured XML query; meaningful result; structural index; inverted index

1 引 言

XML 逐渐成为数据表示、存储和交换标准之一,如何高效查询 XML 文档引起了越来越多研究人员的关注.当用户不了解 XML 文档结构,或者文档缺少诸如 DTD 和 XSD 的结构说明时,用户无法写出准确描述查询要求的查询表达式.另外,对于大量的异构 XML 文档来说,即使存在文档结构说明,对同一个查询也要为每个异构文档编写不同的查询表达式.为此,研究人员提出了一种新的 XML 文档查询方式——非完全结构查询(Non-Fully Structured Query, NFS 查询).NFS 查询允许用户利用部分 XML 结构信息,甚至仅仅是关键字来描述查询要求[1-2]①.

为了便于讨论,本文 NFS 查询语言采用了一种类似于 Timber[2] 中扩展 XQuery 语言,如图 1 所示的 NFS 查询,但略有不同的是,本文用关键字 Rel

收稿日期:2005-07-14;修改稿收到日期:2006-05-16.本课题得到国家自然科学基金(60573090)资助.**李晓光**,男,1973 年生,博士,主要研究领域为数据库技术、Web 挖掘、信息检索.E-mail: xgli@lnu.edu.com.**于 戈**,男,1962 年生,博士,教授,博士生导师,主要研究领域为数据库理论与技术.E-mail: yuge@mail.neu.edu.cn.**龚 剑**,男,1981 年生,硕士研究生,主要研究方向为 XML 数据库.**王大玲**,女,1962 年生,博士,教授,主要研究领域为数据挖掘与 Web 挖掘.**鲍玉斌**,男,1968 年生,博士,副教授,主要研究方向为数据仓库、OLAP.

① Timber Database, the Timber Team, The University of Michigan, http://www.eecs.umich.edu/db/timber

代表一种有意义的判断标准，并代替 Timber 中的 mlcas 关键字. 由于 NFS 查询允许用户用不完整的结构信息来书写查询表达式，查询结果中通常包含大量没有意义的结果，所以 NFS 查询首先要解决的问题就是如何判定查询结果是否有意义，即有意义判断. 以图 2(a)所示的 XML 文档为例，如果不考虑 Rel 关键字，则符合查询 $Q1$ 的结果有(3,5)，(3,10)，(8,10)和(8,5)，但很明显真正有意义的结果只有(3,5)和(8,10). 针对此，XSEarch[1] 提出一种 interconnection relationship 判断标准，主要思想是“两个节点的最小公共子树中是否存在两个具有相同标签名的节点”. 对于图2(a)文档来说，由于节点 2 和 7 的标签名相同，那么(3,10)和(8,5)为无意义结果，然而对图 2(b)文档来说，XSEarch 则认为(3,10)和(8,5)是有意义的结果，这显然不合理. Timber 提出一种有意义的最小公共祖先(meaningful lowest common ancestor, mlca)的判断标准，主要思想为“两个节点的最小公共子树中是否存在一个节点，其标签名与两节点之一的标签名相同”，mlca 对图 2(a)和(b)文档来说，都可以返回有意义的结果，然而对图 2(c)中的文档却不再适用.

```
Q1: for $a in Rel doc//author,
    $t in Rel doc//title
    return ⟨r⟩{$a, $t}⟨/r⟩
```

图 1 一个 NFS 查询

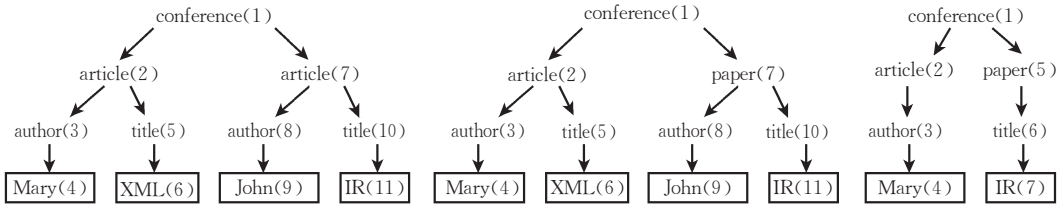


图 2 3 个 XML 数据示例(其中标记为方框的节点为值节点，括号中的数字是节点 ID)

除了有意义判断，NFS 查询同传统的 XML 查询类似，也涉及基本的路径查询和基于关键字的查询，但由于缺少完整的结构信息，NFS 查询的候选结果通常要大于传统查询，如果逐一判断候选结果是非常耗时的. 如 DBLP 文档中，“author”节点有 1256414 个，“title”节点有 552304 个，那么对于查询 $Q1$ 来说，候选结果数量为 1256414×552304. XSEarch 采取了离线计算任意两个节点 interconnection 关系，以二维数组或哈希表形式存储，其时间复杂度和空间复杂度均为 $O(|T|^2)$，T 为 XML 文档的节点个数，并采取逐一判断候选结果的方法. XSEarch 通常适用于小规模的数据集和基于关键字的查询，而对大规模数据集和路径查询而言，建立索引和有意义判断都非常耗时. Timber 采取了一种在线判断方法，不用创建任何用于有意义判断的索引，不需要对结果做笛卡儿积. 但 Timber 要求无论查询表达式如何，必须先将同一标签名的所有节点返回，判断须在其他谓词操作之前，不利于查询计划优化.

本文着重研究在大规模 XML 文档下，如何保证查询质量的同时，高效地执行 NFS 查询. 首先，本文讨论了什么是有意义的 NFS 查询结果，并提出了一种易于扩展的基于模式和实体的判断模型(PE 模型). 基于 PE 模型，设计了一种等价模式和等价查询项的判断方法. 在模型实现中，设计了 PE 索引和增强倒排索引 I2P，提出一种高效执行 NFS 查询的方法. 本文方法有效地利用了 XML 数据库的索引资源，适用于大部分 XML 编码方案，适用于大规模 XML 文档，易于查询计划优化. 大量实验表明，本文方法可以有效地处理路径 NFS 查询和关键字 NFS 查询，查询质量和效率要远远高于 XSEarch 和 Timber.

本文第 2 节详细讨论什么是有意义的查询结果，并提出一种基于模式和实体的判断模型；第 3 节首先根据模型给出一个等价模式和等价查询项确定方法，然后提出 PE 索引和 I2P 索引设计，并提出有效 NFS 查询执行方法；第 4 节为本文方法同 Timber 在查询质量和效率上的实验比较；第 5 节为相关工作；最后，第 6 节为全文的结论.

2 预备知识

本文中 XML 文档采用树模型[1-2]，即 $T=(N, E, r)$. N 为节点集合，$N=NE \cup NV$，其中 NV 为叶子节点集合，又称为值节点集合，NE 为非叶子节点集合，又称内节点集合. E 为边集合且 $E \subseteq N \times N$，$r \in N$ 为根节点. E 满足 r 没有父节点；对 $\forall u \in N$ 且 $u \neq r$，u 有且仅有一个父节点.

定义函数 $label: NE \rightarrow String$，其中函数 $label$ 返回节点 $u \in NE$ 的标签名. 对节点 $u, v \in N$，如果 u

是 v 的父节点，则记做 $parent(u,v)=$true，如果 u 是 v 的祖先节点，则记做 $ancestor(u,v)=$true. 如果 $ancestor(u,v)=$true 或 $u=v$，则记做 $ancestor\text{-}or\text{-}self(u,v)=$true. T 的基于内节点 u 的子树 $T'_u=(N',E',u)$，其中 u 为子树 T'_u 的根节点，$N'=\{u\}\cup\{v\mid v\in N\wedge ancestor(u,v)=\text{true}\}$，$E'=E\cap\{N'\times N'\}$.

定义 1. 最小公共祖先(lowest common ancestor, lca). 节点 $u_1,u_2\in N$，如果节点 $u\in N$ 满足条件：

(1) $ancestor\text{-}or\text{-}self(u,u_1)=$true 且 $ancestor\text{-}or\text{-}self(u,u_2)=$true；

(2) $\forall u'\in N$，如果 $ancestor\text{-}or\text{-}self(u',u_1)=$true 且 $ancestor\text{-}or\text{-}self(u',u_2)=$true，则 $ancestor\text{-}or\text{-}self(u',u)=$true.

称 u 为 u_1 和 u_2 的最小公共祖先，记做 $u=lca(u_1,u_2)$.

定义 2. 节点路径 $np=u_1u_2\cdots u_m$ 是 T 中的节点序列，$u_i\in N$，$1\le i\le m$，并且 np 中任意一对节点 u_i 和 u_{i+1}，存在边 $(u_i,u_{i+1})\in E$. 标签路径 $lp=l_1l_2\cdots l_m$ 是 T 中内节点的标签序列，$\forall l_i$，$1\le i\le m$，$\exists u\in NE, label(u)=l_i$. 设标签路径 $lp=l_1l_2\cdots l_m$ 和节点路径 $np=u_1u_2\cdots u_m$，如果 $label(u_i)=l_i$，$0\le i\le m$，则节点路径 np 匹配标签路径 lp.

定义 3. 设节点 v，路径 np 为根节点到 v 的路径，即 $np=u_0u_1u_2\cdots u_mv$，$u_0=r$，称节点 u_i ($0\le i\le m$) 为 v 的 i-ancestor，记做 $ancestor_i(u_i,v)=$true.

另外，本文中路径的交和并操作是指将路径中节点序列作为集合后的操作.

定义 4. NFS 查询项. NFS 查询中标注为 Rel 的变量称为 NFS 查询项.

如图 3 所示的查询 $Q2$ 的 \$$a$，\$$t$ 和 \$$d$ 分别为 NFS 查询项. 一般来说，查询项分为：(1) 仅包含路径信息，如 $Q2$ 中的查询项 \$$a$；(2) 仅包含关键字，如 $Q2$ 中的查询项 \$$t$；(3) 同时包含路径和关键字，如 $Q2$ 中的查询项 \$$d$；

不失一般性，本文假定查询 Q 中只包含 NFS 查询项，记做 $Q=\{t_1,t_2,\cdots,t_n\}$，设 R_{t_i} 为满足查询项 t_i 的节点集合，称集合 $R=R_{t_1}\times R_{t_2}\times\cdots\times R_{t_n}$ 为 Q 的候选结果集合. 另外，在没有特别说明的情况下，本文所提到的查询均为 NFS 类型查询.

```
Q2: for $t in Rel doc//,
      $a in Rel doc//author,
      $d in Rel doc//description
    where contain($d, 'database') and
          contain($t, 'xml')
    return ⟨r⟩{$a, $t}⟨/r⟩
```

图 3　NFS 查询 $Q2$

3　有意义的查询结果

给定查询 Q 和查询候选结果 R，显然最有意义的查询结果是能够准确反应用户查询意图的结果. 但是用户的查询意图是千变万化的，而所提供的查询表达式却可能相同，在这种情况下，准确反应所有用户的查询意图几乎是不可能的. 以图 4 中的 XML 文档为例，假设有两个查询意图：

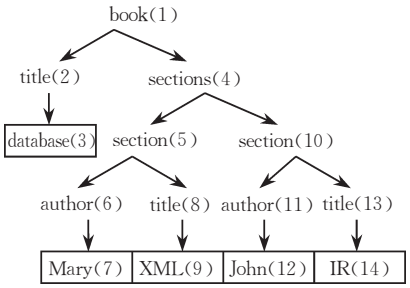


图 4　XML 文档示例

意图 1：查找书中一个部分(section)的作者为 Mary 的书标题.

意图 2：查找书中作者是 Mary 的部分的标题.

如果用户完全了解 XML 文档的结构信息，那么根据这两个查询意图，可以提交诸如图 5(a)和图 5(b)的传统 XQuery 查询，显然意图 1 的查询结果应为节点 2，意图 2 的查询结果应为节点 8. 然而当用户只了解部分结构，假设只了解“author”和“title”标签，则对于不同的查询意图可能提交完全相同的 NFS 查询，如图 5(c)所示. 该 NFS 查询候选结果有 2 和 8，显然很难区分哪些是符合意图 1，哪些是符合意图 2.

类似的问题在信息检索领域中已作了充分的讨论，问题的根本在于“是否存在一种可实现的方法能

```
for $z in doc/book,
    $a in $z /sections/section/author,
    $b in $z /title
where $a/text()="Mary"
return ⟨result⟩{$b}⟨/result⟩
```

(a) 意图 1 的 XQuery 查询

```
for $z in doc/book/sections/section,
    $a in $z/author,
    $b in $z/title
where $a/text()="Mary"
return ⟨result⟩{$b}⟨/result⟩
```

(b) 意图 2 的 XQuery 查询

```
for $a in Rel doc//author,
    $b in Rel doc//title
where $a/text()="Mary"
return ⟨result⟩{$b}⟨/result⟩
```

(c) 意图 1 和 2 的扩展 XQuery 查询

图 5　3 个 XQuery 查询

够从系统的角度来准确猜测用户的意图”,答案是否定的.对于这个问题,通常的办法是从系统角度出发,而不是从用户角度出发,定义一种用户普遍接受的判断方法,如信息检索中广为采用的关键字匹配方法.本文在判断有意义的查询结果时也遵循了这种思想,提出一种基于模式和实体的判断模型.

3.1 模式与实体

XML 文档中的每一个内节点 u 以及其子树 T'_u 可视为一个逻辑块,XML 文档由一系列的逻辑块以一定的组织形式构成.每个逻辑块可作为一个语义单位,其代表了现实中某个具体的对象,逻辑块的结构元信息反映了该对象的属性和特征,相同类型的对象应具有类似的结构元信息.本文称逻辑块为实体,称逻辑块的结构元信息为模式.另外,实体对应的对象类型称为实体类型.

定义 5. 设 XML 文档 T,实体 n_u 为二元组 (u, T'_u),其中 $u\in NE$,T'_u 为 u 的子树.设实体 n_u 和 n_v,如果 v 为 n_u 的 T'_u 的内节点,则称实体 n_v 为实体 n_u 的子实体.如果 n_v 为 n_u 的子实体且 $parent(u,v)=$true,则称 n_v 为 n_u 的直接子实体.在不引起歧义的情况下,本文用节点 ID 来代表节点所对应的实体.

设节点 v 和实体 n_u,N' 为 T'_u 的节点集合,如果 $v\in N'$,则称实体 n_u 包含节点 v.设节点集合 $V=\{v_1,v_2,\cdots,v_n\}$,$n\geq 1$,N_i 为包含节点 v_i 的实体集合,$1\leq i\leq n$,则包含 V 的最小实体 n_u 满足条件:

(1) $u\in\bigcap_{i=1}^{n}N_i$;

(2) $\forall u'\in\bigcap_{i=1}^{n}N_i, ancestor\text{-}or\text{-}self f(u',u)=$true.

定义 6. 实体 $n_u=(u,T'_u)$ 的模式 p_u 是 n_u 的结构元信息,其定义为树 $p_u=(N_u,E_u,r_u)$.$r_u=label(u)$.设 NE' 为 T'_u 的内节点集合,E' 为 T'_u 的边集合,则 $N_u=\{l\mid l=label(u'),u'\in NE'\}$,$E_u\subseteq N_u\times N_u$;$\forall e'=(u_1,u_2),e'\in E'$ 且 $u_1,u_2\in NE'$,$\exists e=(l_1,l_2)$,$e\in E_u$ 且 $l_1=label(u_1)$,$l_2=label(u_2)$.模式 p_u 中的边 $e=(l_1,l_2)$ 表示实体 n_u 中标签为 l_1 的节点可以有零个,一个或者多个标签为 l_2 的节点.在不引起歧义的情况下,p_u 以 r_u 来表示.

设模式 p_u 和 p_v,如果 $ancestor(r_u,r_v)=$true,则称 p_v 为 p_u 的子模式;如果 $parent(r_u,r_v)=$true,则称 p_v 为 p_u 的直接子模式.对模式 p_u 和节点 u',如果 u' 是模式 p_u 的叶子节点,且满足 $parent(r_u,u')=$true,则称 u' 为模式 p_u 的属性.

在实际中,实体类型相同,但模式表现形式不同的情况是普遍存在的.这主要是因为:

(1) XML 文档往往是不完整的,即允许缺失值和重复值,如图 2(c)所示的 XML 文档中的节点缺失.

(2) XML 文档表达方式的不同,如图 2(b)的 XML 文档所示,尽管实体 2 和实体 7 均是类型为文章的实体,但其节点 2 和 7 的标签却不同.

(3) XML 文档组织方式的不同,如图 4 的 XML 文档,在其他文档中可能忽略“sections”节点.

定义 7. 等价模式.设模式 p_u 和 p_v,如果 p_u 和 p_v 分别为同种类型实体的模式,则称 p_u 和 p_v 等价,并记做 $equvilent(p_u,p_v)=$true.

定义 8. 等价查询项.设查询项 t_1 和 t_2,如果用户关于 t_1 和 t_2 的查询意图为具有等价模式的不同实体时,称 t_1 和 t_2 为等价查询项.

3.2 判别原则

本文在有意义判断时,遵循下面这个假设:

(1) 如果两个节点对应的查询项非等价,且在包含节点的实体中,不存在等价模式的实体,则二者有意义.

(2) 如果两个节点所对应的查询项等价,则二者有意义.

(3) 如果节点集合是有意义的,集合中任意两个节点都有意义.

以图 2(b)为例,如果给定节点 3 和节点 10,则分别包含这两个节点的实体有实体 2 和 7,但实体 2 和 7 的模式都是“文章”模式,是等价的,那么在实体 2 和 7 中,分别会包含较节点 10 和 3 更有意义的节点,节点 3 和 10 是不相关的.

定义 9. 设节点 v_1 和 v_2,N_1 和 N_2 为包含节点 v_1 和 v_2 的实体集合,n_u 为包含集合 $\{v_1,v_2\}$ 的最小实体,则集合 $(N_1\cup N_2)-(N_1\cap N_2)\cup\{n_u\}$ 为节点 v_1 和 v_2 的上下文实体集合.

定义 10. PE 规则(Rule Based on Pattern and Entity).给定查询 $Q=\{t_1,t_2,\cdots,t_n\}$,$n\geq 2$,设 $r_Q=\{v_1,v_2,\cdots,v_n\}$ 为 Q 的一个查询结果,如果 r_Q 是有意义的,当且仅当对任意节点 v_i 和 v_j,$1\leq i,j\leq n$,$i\neq j$.

(1) 设 N' 为 v_i 和 v_j 的上下文实体集合,不存在实体 $u,v\in N'$,满足 $equvilent(p_u,p_v)=$true.

(2) 如果存在实体 $u,v\in N'$,满足 $equvilent(p_u,p_v)=$true,则 v_i 和 v_j 所对应的查询项 t_i 和 t_j 为等价查询项.

4 有效的 NFS 查询

本节基于第 2 节的判断模型,着重解决如何有

效地执行 NFS 查询. 首先讨论了适用于本节方法的 XML 编码，提出一种具体的等价模式和等价查询项的判断方法. 然后提出了一种 PE 索引和增强的倒排索引(Inverted Index with Path，I2P)，并给出一种具体的判断方法. 最后，利用本节提出的索引结构和判断方法，给出一种有效的查询算法.

4.1　XML 编码

本文假设 XML 文档编码符合：

(1) 任意两个节点的编码符合先序关系“<”，即编码是可以通过先序遍历 XML 文档得到的；

(2) 根据节点编码，可以直接判断节点间的祖孙关系，并且给定一个节点 v，可以获得 i-ancestor 的编码.

目前主要的编码方式均符合上述条件. 如前缀编码[6,8]和区域编码[3-5]等，节点编码均为先序遍历获得，并可由此获得节点的 i-ancestor 编码，如前缀编码中的 Dewey 编码，节点的 i-ancestor 编码为该编码长度为 i 的前缀，区域编码中 i-ancestor 的编码应包含该节点编码且层数为 i.

性质 1.　设 XML 文档编码符合上述要求，则对节点集合 $N=\{u_1,u_2,\cdots,u_i,\cdots,u_m\}$，设 $u_1<u_2<\cdots<u_{i-1}<u_i<u_{i+1}<\cdots<u_m$，假设存在 d，$\forall u'_i$，$ancestor_d(u'_i,u_i)=\text{true}$，$1\leq i\leq m$，那么节点集合 $N'=\{u'_1,u'_2,\cdots,u'_i,\cdots,u'_m\}$ 满足 $u'_1\leq u'_2\leq\cdots u'_{i-1}\leq u'_i\leq u'_{i+1}\cdots\leq u'_m$，其中“$\leq$”表示符合序“<”或者相等(证明：略).

4.2　等价模式和等价查询项

对第 2 节中的有意义判断模型而言，关键问题是如何定义具体的等价模式和等价查询项判断方法. 前面提到，由于 XML 文档的不完整性，表达方式和组织方式的差异，往往导致同样类型的实体而模式不同. 实际上，在同一个 XML 文档中，组织方式的差异一般不存在，本文主要考虑了前两个因素，结合标签内容和模式结构相似性确定等价模式.

设模式 p_u 和 p_v，A_u 和 A_v 分别为二者的属性集合，SUB_u 和 SUB_v 分别为二者的直接子模式集合，则如果模式 p_u 和 p_v 满足条件：

(1) $r_u=r_v$ 或者

(2) $sim(p_u,p_v)\geq\theta$，

$$sim(p_u,p_v)=\frac{|A_u\cap A_v|+|SUB_u\cap SUB_v|}{|A_u\cup A_v|+|SUB_u\cup SUB_v|},$$

则 $equvilent(p_u,p_v)=\text{true}$，其中 $|A_u\cap A_v|$ 为 A_u 和 A_v 中相同属性的个数，$|A_u\cup A_v|$ 为 A_u 和 A_v 中不同属性的个数，$|SUB_u\cap SUB_v|$ 为 SUB_u 和 SUB_v 中等价模式的个数，$|SUB_u\cup SUB_v|$ 为 SUB_u 和 SUB_v 中不同模式的个数. θ 为给定的阈值，$0\leq\theta\leq1$. 显然 $0\leq sim(p_u,p_v)\leq1$.

条件 1 考虑了标签名中包含的语义. 这里假设标签名不存在多义性(在单个 XML 文档内几乎都满足)，则当两个模式的根节点标签名相同，它们描述的实体类型必然相同. 条件 2 考虑了对于同一类实体的表达方式不同所带来的差异，其出发点是如果两个模式代表同种类型的实体，那么它们的结构应该是相似的.

另外，由于 XML 文档不完整性的特点，如果模式 p_u 和 p_v 满足条件 $r_u=r_v$，那么即使 p_u 和 p_v 在表现形式上有所不同，通常也属于同一个模式，尤其当 u 和 v 的父节点相同的时候，有如下假设成立.

假设 1.　设模式 p_u 和 p_v，如果满足条件 $r_u=r_v$，且 $parent(u)=parent(v)$，则模式 p_u 和 p_v 为同一个模式.

由于等价查询项往往涉及到用户的查询意图，很难判断哪些查询项为等价的，一般需要由用户直接指定. 本文采用了另一种方法，即利用节点的模式等价性来区分是否为查询项等价. 设节点 v_1 和 v_2，如果 $equvilent(p_{v_1},p_{v_2})=\text{true}$，则节点 v_1 和 v_2 为查询项等价.

4.3　PE 索引

PE 索引包括两个部分：基于标签内容的等价模式索引和基于结构相似性的等价模式索引. 基于标签内容的等价模式索引的基本思想是对于同一父模式下的子模式，如果符合第 4.2 节中等价模式判断条件 1，则根据假设 1，将它们合并为同一模式. 该索引中每个内节点及其子树代表了一个模式，并且包含了 XML 文档中所有实体的模式. 如图 6 的 XML

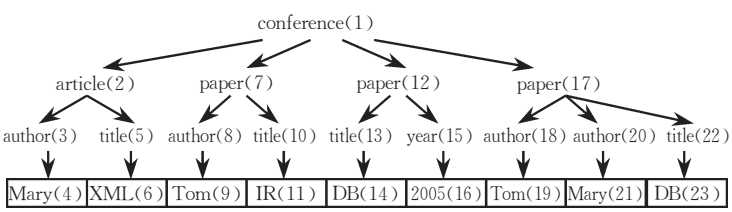


图 6　XML 文档

文档,基于标签内容的等价模式索引如图 7(a)所示,其中方框内为符合该模式的实体 ID. 基于结构相似性的等价模式索引的基本思想是在基于标签内容的等价模式索引的基础上,根据第 4.2 节中等价模式的判断条件 2,查找所有符合条件的等价模式,如图 7(b)所示.

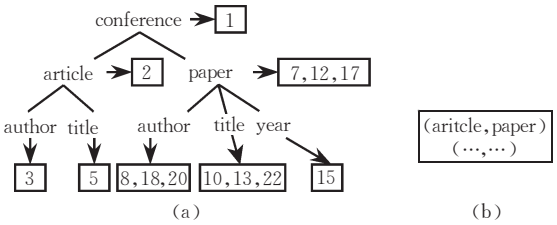


图 7 PE 索引

为了便于区分,在没有特别说明的情况下,下面部分将 PE 索引中的节点称为标签节点,XML 文档中的节点称为节点,并且如果节点 ID 在标签节点所对应的 ID 集合中,则称标签节点包含该节点.

性质 2. 基于标签内容的等价模式索引等价于基于树模型的 DataGuide 索引[9]和 1-index[10](证明:略).

性质 3. 给定节点路径,基于标签内容的等价模式索引中必然有且仅有一条与节点路径匹配的标签路径. 这里,匹配含有除了定义 1 中的含义,还要求节点路径中的每个节点包含在所对应标签节点中(证明:略).

4.4 判别方法

给定第 4.2 节中的等价模式和等价查询项判定方法和 PE 索引,对查询 $Q=\{t_1,t_2,\cdots,t_n\}$,$n\geq 2$,设 $r_Q=\{v_1,v_2,\cdots,v_n\}$ 为 Q 的一个候选结果,如果 r_Q 是有意义的,当且仅当任意节点 v_i 和 v_j 满足以下下条件($1\leq i,j\leq n$, $i\neq j$):

(1) 设 l_n^i 和 l_m^j 分别为 PE 索引中包含节点 v_i 和 v_j 的标签节点,$l_k=lca(l_n^i,l_m^j)$,d 为 PE 索引中根节点到 l_k 的深度,以及路径 $lp^i=l_kl_{k+1}^i\cdots l_n^i$ 和 $lp^j=l_kl_{k+1}^j\cdots l_m^j$,那么在模式集合 $lp^i\cup lp^j$ 中,不存在两个等价模式,且

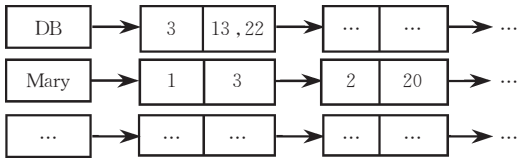
(2) 设节点 v_i' 和 v_j', $ancestor_d(v_i',v_i)=\text{true}$, $ancestor_d(v_j',v_j)=\text{true}$,那么 $v_i'=v_j'$;

(3) 否则,$label(v_i)=label(v_j)$.

定理 1. 对第 4.2 节等价模式和等价查询项而言,本节中判别方法与 PE 规则一致(证明略).

4.5 增强倒排索引

由于判断方法需要为每个节点在 PE 树中查找所对应的标签路径,如果查询项中仅仅包含关键字信息,那么对每个节点必须遍历整个 PE 索引并判断节点是否包含在该索引节点中才能确定标签路径,显然是非常耗时的. 本节在传统的倒排索引基础上,提出一种带有路径信息的倒排索引(Inverted Index with Path, I2P). 同传统的倒排索引不同,对每个关键字的索引项,除了包含关键字的节点 ID,I2P 还保存了从该节点到根节点的标签路径信息. I2P 索引由两部分组成:增强的倒排索引和标签路径索引. 增强倒排索引中具有相同路径的节点被组织在一起并有序. 如图 8 为图 6 中文档的 I2P 索引.



(a) 增强的倒排索引

1:conference/article/author
2:conference/paper/author
3:conference/paper/title
…:…

(b) 标签路径索引

图 8 I2P 索引

当然,如果 I2P 中只包含部分路径信息,会降低标签索引的存储空间,但查询时,则必须在 PE 索引中查找完整的标签路径. 如果保存从根节点开始的路径信息,则可以省略查找,直接根据该路径信息判断是否有意义,而且一般情况下 PE 索引的节点数量很小,标签路径索引所占的空间很小,所以本文选取后一种方法.

4.6 算 法

可以看出,如果 XML 提供了 DTD 或者 XSD,则可以直接利用并创建索引. 本节给出在缺少 DTD 或者 XSD 下创建 PE 索引的算法以及 NFS 查询算法. 下面为相关结构定义:

Structure StackNode {bool *b* Pattern; void * *p*;} Structure Node {char * label; ID *id*[];};

Structure Pattern {char * label; ID *rid_list*[]; Node *attribute_list*[]; Pattern *subpattern_list*[];};

Structure QueryItem{char * labelpath; ID *id*[];}.

4.6.1 创建索引

算法 1 为索引创建算法,它是一种基于堆栈的算法. 特别注意,由于函数 Traverse 访问节点的顺序同 XML 文档编码方式一致,则 PE 索引中节点的 ID 集合也有序.

算法 1. 创建 PE 索引和 I2P 索引.

输入: An XML document T

输出: PE 索引和 I2P 索引

1. $STACK$←NULL;EP←NULL;$curNode$←NULL;$I2P$←NULL;
2. while((ret←$Traverse$($curNode$))≠$FINISHED$) {
3. if(ret=$BEGIN\_ELEMENT$) {
4. Node u; $u.label$←the tag of $curNode$; add the ID of $curNode$ into $u.id$[];
5. StackNode s; $s.bPattern$←true; $s.p$←&u; push s into $STACK$;}
6. else if (ret=$END\_ELEMENT$){
7. if($STACK$ 顶部元素标签与 $curNode$ 标签相同){
8. pop $STACK$ into s;
9. 自栈顶查找栈节点 s',其满足 $s'.p$→$label$=$s.p$→$label$,直到 $bPattern$=true. 如果存在,则 append $s.p$→id[] into $s'.p$→id[]; continue;}
10. Pattern newPattern; newPattern.$label$←tag of $curNode$;
11. while (t $STACK$ 顶部元素标签与 $curNode$ 标签不同) {
12. pop $STACK$ into s;
13. if ($s.bPattern$=false) { add $s.p$ into $newPattern.attribute\_list$;}
14. else{add $s.p$ into $newPattern.subpattern\_list$;}}
15. for each stack node s of $STACK$ and $s.b$ Pattern isn't true) {
16. if ($s.b$ Pattern=true && $s.p$→$label$=$newPattern.label$) {
17. 将模式 $newPattern$ 与 $s.p$ 合并; delete s;}}
18. initialize a new stack node s, $s.bPattern$←true, $s.p$←&$newPattern$;
19. push s into STACK;}
20. else if (ret=TEXT){for each keyword, insert $curNode$'s ID and $labelpath$ into I2P, 其中 $labelpath$ 为栈自底到顶节点标签序列}}
21. pop $STACK$ into s;
22. 根据式(1),标记所有 $s.p$ 上不同层次上的等价子模式并将其添加到 EP;
23. return $s.p$ and EP;

定理 2. 设 N 为模式的直接子模式最大个数,d 为 PE 索引中树的最大深度,N 为文档节点个数,m_k 为关键字个数,c_k 为每个关键字加入到倒排索引中所需最大时间,算法 1 的时间复杂度为

$$O\left(N\left(m_a+m_p+m_a^2\frac{m_p^d-1}{m_p-1}+m_p^2\frac{m_p^{d-1}-1}{m_p-1}\right)+m_kc_k\right).$$

证明略.

从定理 2 中,我们可以看出算法 1 的时间复杂度与文档节点数和关键字个数成线性.

4.6.2　NFS 查询

算法 2 为 NFS 查询算法,步 15～22 为有意义判断阶段,步 21 为根据第 4.2 节的条件 2 判断,注意由于 $qi_j.id$[], $qi_k.id$[]为有序,则在给定 d,根据第 4.1 节中性质 1,它们的 d-ancestor 有序,那么步 22 的判断类似于"结构连接",只需判断 $|qi_j.id[]|+|qi_k.id[]|$ 次.

算法 2.　NFS 查询算法.

输入:$Q=\{t_1,t_2,\cdots,t_n\}$,PE 索引和 I2P

输出:查询结果 R

1. QueryItem QI_1[],QI_2[],…, QI_i[],…,QI_n[];
2. for each t_i,$1\leq i\leq n$, do{
3. if t_i 只包含路径信息 then {
4. 根据 PE 索引,查找所有满足 t_i 的标签路径,对每一个标签路径 p, do {
5. QueryItem qi; $qi.labelpath$←p;
6. append the id set of node at the end of p into $qi.id$[]; append qi into QI_i[];}}
7. if t_i 包含路径和关键字信息 then {
8. 根据 I2P 索引,查找所有满足关键字的节点集合 N;
9. 根据 PE 索引,查找所有满足 t_i 的标签路径,对每一个标签路径 p, do {
10. QueryItem qi;$qi.labelpath$←p; 根据 N 过滤 p 的节点集合并将其添加到 $qi.id$[]; append qi into QI_i[];}}
11. if ti 只包含关键字 then {
12. 根据 I2P 索引,获得相应的倒排项集合,对每一个倒排项 it do {
13. QueryItem qi; $qi.labelpath$←$it.labelpath$; append $it.id$[]into $qi.id$[];append qi into QI_i[];}}}
14. create Cartesian $QS=QI_1\times QI_2\times\cdots\times QI_n$;
15. for each $qs\in QS$, suppose that $qs=\{qi_1,qi_2,\cdots,qi_n\}$ do {
16. 如果存在 qi_i and qi_j 不满足条件 1 和 3,则继续;
17. QueryItem tmp[];append qi_1 into tmp[];
18. for each $qi_j\in qs$, $2\leq j\leq n$ do {
19. for each $qi_k\in tmp$, do {
20. 计算 $qi_j.labelpath$ 和 $qi_k.labelpath$ 的 lca 和深度 d;
21. 根据条件 2,顺序确定 $qi_j.id$[]和 $qi_k.id$[]中相关的节点对;}}
22. append tmp[] into R;}
23. return R;

以图 6 的 XML 文档和查询 $Q1$ 为例,预处理阶段后结果见图 9(a),做笛卡儿积后,获得 4 个中间结果 qs_1～qs_4. 对 qs_2 和 qs_3 来说,由于模式"paper"和"article"等价,即不符合条件 1,所以 qs_2 和 qs_3 中的节点相互不相关(步 16). 而对 qs_1 和 qs_4 来说,显然符合条件 1. qs_4 中,步 21 的过程如图 9(c)所示:d=2,括号中为节点的 2-ancestor,该过程类似与结构连接,连接条件为节点的 2-ancestor 是否相同,由于空间所限,这里不赘述了.

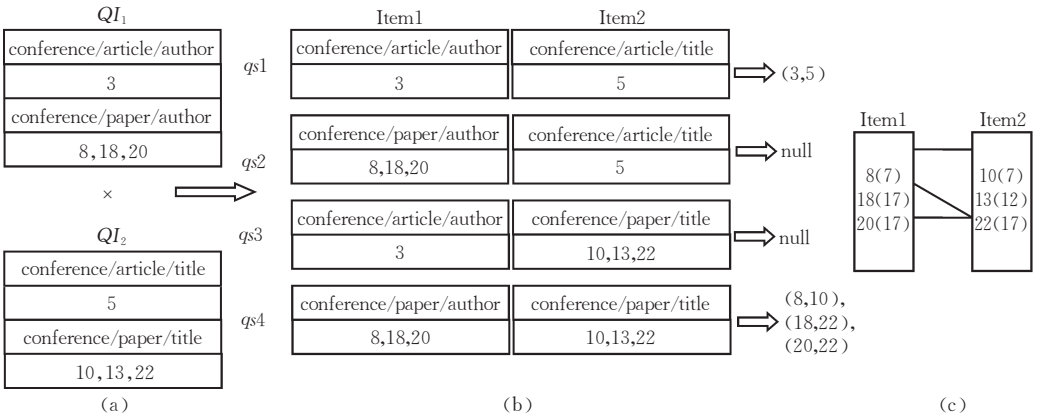


图 9 算法 2 的一个实例

定理 3. 设 d 为 PE 索引中树的最大深度，$m^i_{l_p}$ 为查询项 t_i 在 PE 索引中不同标签路径的个数，m 为每个标签路径下符合查询项的节点最大个数，n 为查询项个数，m' 为中间结果集合中元素最大个数，则算法 2 中有意义判断的时间复杂度为

$$O\left(\left(\prod_{i=1}^{n} m^i_{l_p}\right)(n^2(m+m')+d^2)\right).$$

证明略.

从定理 3 可以看出，算法 2 的复杂度主要受中间结果 m' 影响，实际上，大部分无意义的候选结果在步 17 就可以被过滤，而实际中 d 值很小，这使得算法 2 的运行时间较定理 3 的时间要少的多. 较小的 m' 也使得算法 2 较 mlca 的效率更高，特别是带有关键字的查询.

5 实验分析

5.1 实验设计

实验选取了实际 XML 文档 SIGMODRECORD①、DBLP② 和自动生成的 XML 文档 XMARK③. 由于 Timber 在 DBLP 数据集上的运行时间远远超过 10h，所以本文从 DBLP 中抽取部分作为测试数据集，称 DBLP1. 实验根据不同的生成系数(分别为 0.05,0.1,0.2,0.4,0.8 和 1.6)，自动生成了 6 个 XMARK XML 文档.

由于 XSEarch 在计算 interconnection 关系时非常耗时，在上述 XML 文档规模下，XSEarch 建索引时间通常在 10h 以上，而且查询质量低于 Timber 中的 mlca 方法. 因此，本文选取 Timber 中的 mlca 方法作为比较方法. 在 Timber 系统中，为每个 XML 文档创建了标签名索引和属性名索引，并采用 timbersoap+GUI 的方式运行. 这里将本文提出的方法称为 PE 方法. PE 方法采取了 Dewey 编码方式，θ 值为 0.5. 另外，PE 方法只将 PE 索引中的树结构和 I2P 的所有索引项读入缓存，而对标签节点对应的 ID 集合和索引项对应的倒排表采取了动态交换的方法，缓存大小为 50MB. 本文的实验环境为：CPU：P4，2.4GHz，内存 512MB，硬盘 80GB，操作系统 WINDOWS2000.

实验评价分两个部分：查询质量评价和查询效率评价. 查询质量评价采用了查准率和查全率指标. 设 R_Q 为查询结果，R_C 为标准查询结果，则查准率 $=|R_Q \cap R_C|/|R_Q|$，查全率 $=|R_Q \cap R_C|/|R_C|$. 查询测试用例分为三种：路径查询($Q1 \sim Q12$)、关键字查询($Q17 \sim Q20$)和混合查询($Q13 \sim Q16$). 关键字均为随机选取，查询项个数为 2～4 个. 测试用例的详细情况见表 1. 另外，对每个测试用例，实验采用在完全了解文档结构下 XQuery 查询结果作为标准结果.

表 1 测试用例

SIGMOD RECORD	Q1:{//author: //title:}
	Q2:{//author: //volume:}
	Q3:{//volume: //number:}
	Q4:{//volume: //number: //title:}

DBLP1	Q5:{//author: //title:}
	Q6:{//author: //year:}
	Q7:{//title: //year: //url:}
	Q8:{//author: //title: //year: //url:}

XMARK	Q9:{//location: //quantity:}
	Q10:{//location: //quantity: //name:}
	Q11:{//homepage: //emailaddress: //phone:}
	Q12:{//seller: //buyer: //price: //date:}
	Q13:{//location:United //quantity:1}
	Q14:{//location:United //quantity:1 //name:nine}
	Q15:{//homepage:Emery //emailaddress:Emery //phone:37214856}
	Q16:{//seller:person520 //buyer:person728 //price:78.33 //date:1999}
	Q17:{:United :1}
	Q18:{:United :1 :nine}
	Q19:{:Emery :Emery :37214856}
	Q20:{:person520 :person728 :78.33 :1999}

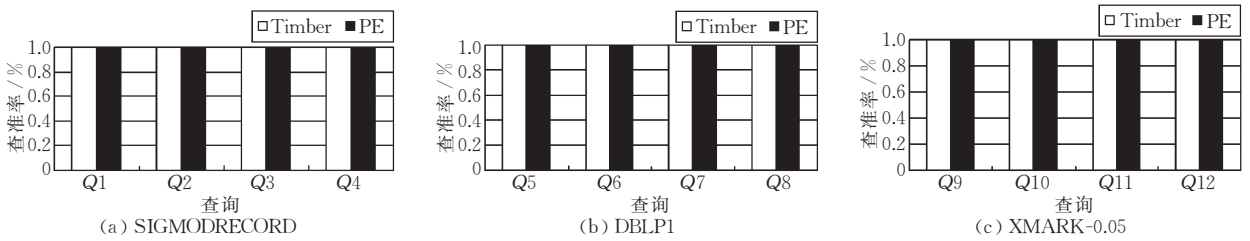
① Sigmod Record. http://www.dia.uniroma3.it/araneus/sigmod/record/sigmodrecord/sigmoderecord.xml

② Ley M. DBLP bibliography. http://dblp.uni-trier.de/xml/, 2003

③ Xmark. http://monetdb.cwi.nl/xml/index.html

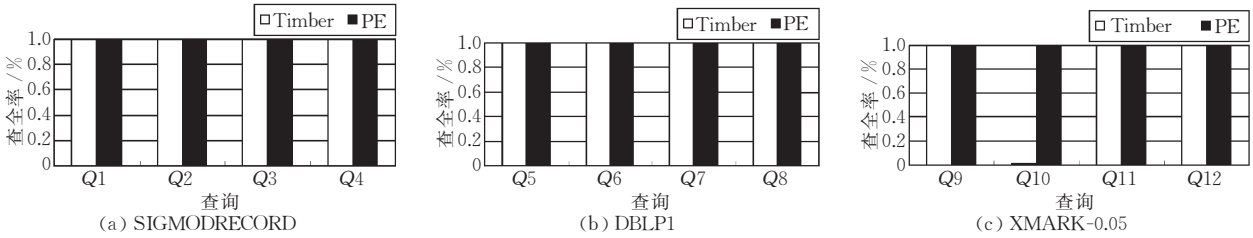
5.2 性能比较和分析

由于 Timber 中扩展 XQuery 不支持关键字查询，查询质量评价实验中选取测试用例 $Q1\sim Q12$ 和 SIGMODERECORD，DBLP1 和 XMARK-0.05. 图 10 和图 11 为 Timber 和 PE 方法的查准率和查全率比较. 在所有的测试用例和文档上，PE 的查准率和查全率均为 100%，Timber 的查准率均为 100%，而查全率基本为 100%. 从整体上看，Timber 查询质量同 PE 方法基本相同，这是因为实验中的文档基本上不存在缺失值，而且等价模式间的属性绝大部分重叠，这使得 Timber 中的判断方法同本文方法同样有效，但在文档 XMARK-0.05＋用例 Q10 中，其查全率仅为 0.008，主要因为 Timber 判断方法的缺陷过滤了“查找所有人可能所在的地区位置”的查询结果.



(a) SIGMODRECORD　(b) DBLP1　(c) XMARK-0.05

图 10　查准率



(a) SIGMODRECORD　(b) DBLP1　(c) XMARK-0.05

图 11　查全率

表 2 为 PE 和 Timber 在 XMARK 文档上的运行时间比较，其中“NA”代表超过 10h，“0”代表小于 1ms. 每个查询用例运行时间为三次运行的平均值. 总的来说，PE 运行时间远小于 Timber，而在用例 Q10 上，PE 运行时间平均大于 Timber，原因在于 Timber 的查全率平均为 0.008，而 PE 查全率均为 1，也就是说，PE 判断的候选结果要远大于 Timber，从而导致运行时间大于 Timber. 而在 DBLP 文档（未经筛选）以及用例 $Q5\sim Q8$ 上，尽管 PE 和 Timber 的查全率相近，但 PE 的运行时间分别为 340.125s，227.89s，420.468s 和 1284.531s，而 Timber 运行时间则远远高于 10h. 表 3 为 PE 在文档 XMARK-0.05～XMARK-1.6$^+$ 用例 $Q13\sim Q16$ 上的实验结果，由于测试用例 $Q13\sim Q16$ 带有关键字信息，候选结果小于 $Q9\sim Q12$，PE 的运行时间大幅降低，并且运行时间基本同文档大小成线性. 而前面提到，即使 Timber 支持关键字查询，测试用例 $Q13\sim Q16$ 必须在完成路径查询后才能根据关键字信息过滤结果，那么查询时间较仅路径查询要长. 图 12 比较了在传统倒排索引下和 I2P 倒排索引下 PE 方法关键字查询的运行时间，可以看出 I2P 索引下的关键字查询效率有大幅度提高.

表 2　PE 和 Timber 在不同大小文档上的运行时间比较　（单位：s）

查询	XMARK-0.05		XMARK-0.1		XMARK-0.2		XMARK-0.4		XMARK-0.8		XMARK-1.6	
	Timber	PE	Timber	PE	Timber	PE	Timber	PE	Timber	PE	Timber	PE
Q9	14.469	0.047	25.188	0.093	76.015	0.187	226.89	0.375	523.609	0.766	3028.812	1.735
Q10	20.937	323.641	41.328	5150.861	106.343	NA	261.657	NA	1311.532	NA	7877.266	NA
Q11	14	0.047	27.344	0.11	31.062	0.234	166.359	0.422	528.688	1.063	1864.359	2.156
Q12	24.157	0.062	59.812	0.124	75.75	0.235	188.656	0.468	1678.25	1	NA	2.203

表 3　XMARK-0.05～XMARK-1.6$^+$ $Q13\sim Q16$ 上 PE 运行时间比较　（单位：s）

用例	XMARK-0.05	XMARK-0.1	XMARK-0.2	XMARK-0.4	XMARK-0.8	XMARK-1.6
Q13	0.360	1.266	2.953	7.251	21.327	68.063
Q14	0.359	0.828	1.922	4.453	10.265	23.672
Q15	0	0	0.015	0.047	0.172	0.860
Q16	0	0	0	0	0	0

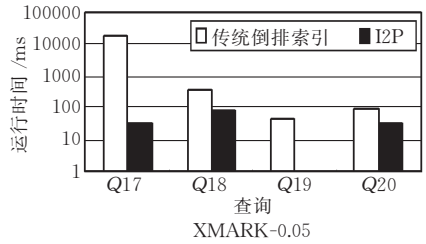


图 12 传统倒排索引和 I2P 索引下 PE 运行时间比较

最后，在 XMARK 文档上，PE 索引平均节点数为 653，最大深度为 10，平均叶子节点数为 434. 在 SIGMODRECORD 上，节点数为 11，最大深度为 7，叶子节点为 5；DBLP1 上节点数为 104，最大深度为 3，叶子节点为 92. 图 13 为 PE 索引大小同 XML 文档大小和 XML 节点数的比值，图 14 为在不同文档下，PE 索引创建时间，其中 X 和 Y 轴刻度值呈倍数增长. 可以看出，PE 索引大小和创建时间基本同 XML 文档节点数和大小成线性. 图 15 为 I2P 索引同传统倒排索引的大小比较，I2P 索引较传统倒排索引平均只增加了 6.3%，而关键字查询效率却平均提高了 99.23%.

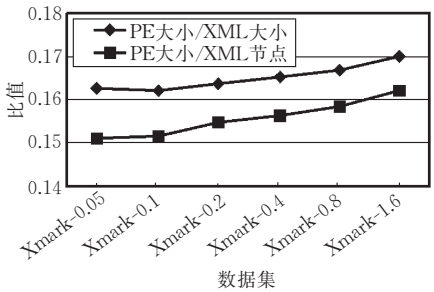


图 13 PE 索引大小变化

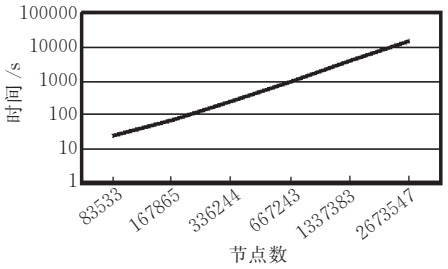


图 14 PE 索引创建时间

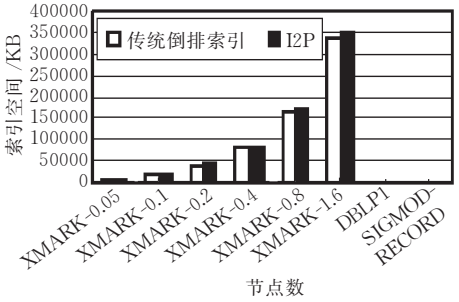


图 15 I2P 和传统倒排索引大小比较

6 相关工作

MEET[7] 提出将节点集合的最小公共祖先作为返回结果. XRANK[8] 方法均要求返回最特殊的结果，并提出一种类似 PAGERANK 的排序方法. 这两种方法都不涉及有意义判断，自然返回大量无意义的结果. 文献[11]中提出一种有用户事先对 XML 文档分片指定，落在一个分片内的节点被认为是有意义的，但显然在大规模 XML 文档情况下，这种方法很不适用.

目前，判断有意义的查询结果的方法主要有两种：XSEarch 中的 interconnection 方法和 Timber 中的 mlca 方法. 实际上，这两种方法的基本思想也是基于本文提出的思想，可以作为本文提出的模型的一个特例. interconnection 方法中的等价模式定义实际上是与本文中的条件 1 是一致的，也就是说本文的方法包含了 interconnection 方法，但显然，interconnection 方法只解决了 XML 文档不完整性问题，而对于如图 2(b)所示的 XML 文档，则其查询精度大幅下滑. mlca 方法实际上是一种基于结构相似性的确定等价模式. 但是这种结构相似性定义与本文的方法相比，显然是不够准确的. 当数据缺失非常严重的时候，如图 2(c)所示，mlca 会产生判断错误. 当然，如果单独对两个实体的模式进行判断，那么本文的方法也存在这个问题，但是如第 4 节所示，本文通过合并模式解决了数据缺失的问题.

从执行效率上，XSEarch 建立索引是非常耗时的，对关键字查询有很高的效率，但对诸如测试用例 $Q1$～$Q12$ 的查询，由于需要对查询项所有候选结果做笛卡儿积，导致其判断效率非常低. 在文献[12]中，我们提出一种改进倒排索引来解决基于标签和关键字的 XML 检索，但无法有效处理路径查询，本文通过 PE 索引在一定程度上解决了这个问题. Timber 效率较高，但对于用例 $Q13$～$Q20$ 的查询以及含有其他谓词的查询，Timber 在执行关键字过滤和其他操作之前进行有意义判断，不利于查询计划的优化，降低了执行效率. 以用例 13 为例，一种有效执行方法是对根据路径信息和关键字信息返回的节点集合做交集，然后执行判断，而 Timber 必须先返回所有符合“//location”和“//quantity”的节点集合，并在此集合上执行判断，然后才能利用关键字信息返回的节点集合过滤. 而本文中的方法只是对标签路径作笛卡儿积，对每个候选集采用了类似结构连接的操作，提高了执行效率，并且不妨碍查询计划的优化.

7 结　语

本文着重研究在大规模 XML 文档下，如何保证查询质量的同时，高效地执行 NFS 查询. 主要贡献有：

(1) 提出了一种普遍适用、便于扩展的 NFS 查询结果的有意义判断模型，并根据此模型，设计了一种有效的等价模式和等价查询项判断方法，其 NFS 查询质量要高于 XSEarch 和 Timber 系统.

(2) 根据提出的等价模式和等价项判断方法，设计了 PE 结构索引和 I2P 倒排索引，它们不仅可以支持传统的路径查询和关键字查询，而且可以有效支持本文提出的 NFS 查询结果判断方法，有效地利用了现有 XML 数据库的索引资源

(3) 提出一种高效的 NFS 查询算法，其时间复杂度要远远小于 XSEarch 和 Timber 系统，适用于大部分 XML 编码方案，适用于大规模数据集.

参 考 文 献

[1] Cohen S, Mamou J, Kanza Y, Sagiv Y. XSearch: A semantic search engine for xml//Proceedings of the 29th International Conference on Very Large Databases (VLDB'03), 2003: 45-56

[2] Li Y Y, Yu C, Jagadish H V. Schema-free XQuery//Proceedings of the 13th International Conference on Very Large Data Bases, Toronto, Canada, 2004: 72-83

[3] Chun Z, Jeffery F. On supporting containment queries in relational database management system. ACM SIGMOD Record, 2001, 30(2): 425-436

[4] Kna D D, Yoshikawa M, Uemura S. An XML indexing structure with relative region coordinate//Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001: 313-320

[5] Li Q Z, Moon B. Indexing and querying XML data for regular path expressions//Proceedings of the 27th VLDB International Conference on Very Large Databases, Rome, Italy, 2001: 361-370

[6] Wang W, Jiang H F, Lu H J, Jeffery X Y. PBiTree coding and efficient processing of containment joins//Proceedings of the 19th ICDE International Conference on Data Engineering, Bangalore, India, 2003: 391-402

[7] Schmidt A, Kersten M, Windhouwer M. Querying xml document made easy: Nearest concept queries//Proceedings of the 17th International Conference on Data Engin, Rome, Italy, 2001: 321

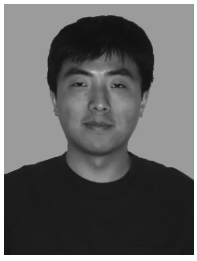
[8] Guo L, Shao F, Botev C, Shanmugasundaram J. XRank: Ranked keyword search over xml documents//Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, 2003: 16-27

[9] Goldman R., Widom J.. DataGuides: Enabling query formulation and optimization in semistructured databases//Proceedings of the 23th VLDB International Conference on Very Large Databases, Athens, Greece, 1997: 436-445

[10] Milo T, Suciu D. Index structure for path expressions//Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel, 1999: 277-295

[11] Hatano K, Kinutani H, Watanabe M et al. Determining the unit of retrieval results for XML documents//Proceedings of the 1st Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), Schloss Dagstuhl, 2002: 57-64

[12] Li X G, Gong J, Wang D L, Yu G. An effective and efficient approach for keyword-based XML retrieval//Proceedings of the 6th International Conference on Advances in Web-Age Information Management (WAIM), Hangzhou, China, 2005: 56-67



LI Xiao-Guang, born in 1973, Ph.D.. His research interests include databases, Web mining and information retrieval.

YU Ge, born in 1962, professor, Ph. D. supervisor. His research interests include theory and technique of databases.

GONG Jian, born in 1981, master. His research interests include XML databases.

WANG Da-Ling, born in 1962, Ph. D., professor. His research interests include data mining and Web mining.

BAO Yu-Bin, born in 1968, Ph. D., associate professor. His research interests include data warehouse and OLAP.

Background

This research belongs to the project of "Study on Deduction Model of Users Motivation Oriented to New Generation Search Engine", supported by the National Natural Science Foundation of China (No. 60573090). This project studies on the deduction model of user's motivation based on the analyses of user's behavior, intention and motivation. In the past, research team has developed some advanced techniques of XML retrieval, data stream mining, and text similarity as the fundamental works for the project. The study on XML querying process, as the core technique of Web information distribution and exchange, facilitates retrieval analyses and quality improvement. As a conclusion of author's previous work on keyword-based XML retrieval, this paper give a comprehensive review of non-fully structured XML query.