XML 查询的代数表示及其查询优化

Algebra Operations and Optimization Techniques for XML Query

李效东　顾毓清

（中国科学院软件所　北京100080）

Abstract　XML is emerging as de facto standard of data exchange and representation on the Web. A query language for XML and corresponding algebra operations and optimization techniques are imperative and very important under this circumstance. The paper proposes an algebra representation for XML query and addresses algebra rewriting-based preliminary optimization techniques in the context of data integration system. In the beginning of the paper, the author also presents an XML data model named XTree and describes an XML query language-AnXQL. The paper is an important part of the research on integrated query processing over hetergeneous data sources.

Keywords　XML Query, Algebra operations, Query optimization, Query tree

本文给出了一种 XML[7]查询的逻辑代数表示，基于这种代数表示作者探讨了在逻辑层面上进行 XML 查询优化的可能性和方法。文章按以下方式组织，第一部分给出了一种 XML 数据的模型表示，接着简单描述了在这个模型上的 XML 查询语言—AnXQL；第二部分是本文的重点，详细描述了 XML 查询的逻辑代数操作符，并举例说明；第三部分在逻辑层面上探讨了 XML 查询优化的方法；最后一部分介绍了相关研究工作，并与本文的方法进行了对比。本文是文[8]中基于半结构化数据模型的集成查询处理研究的重要组成部分。

1. XML 数据模型与 AnXQL 查询语言

数据建模是建立查询语言的基础，正像关系模型是 SQL 查询语言的基础一样。由于 XML 数据具有层次化、嵌套、自描述等半结构化数据的特点，因此必须设计新的数据模型进行 XML 数据表示，并在此基础上构造 XML 查询语言。本部分在半结构化数据模型的基础上，给出了一种 XML 数据模型表示，并在其上建立了一种 XML 查询语言 AnXQL。

半结构化数据既非原始数据，又非像关系型数据那样严格类型化了的数据[1]。半结构化数据一般没有分离的模式约束数据，数据和描述数据的模式纠缠在一起，具有自描述特征；半结构化数据的模式用于描述数据的结构信息，而不对数据结构进行强制性约束。由于半结构化数据组织的灵活性，越来越多的应用中使用半结构化数据，如人类基因组工程中非常流行的 AceDB[2]的数据组织方式，以及 Lotus Notes 的数据组织方式[3]等。XML 同样具有半结构化数据的诸多特征，而且研究界对半结构化数据的表示和查询的研究[1]为 XML 的研究奠定了坚实的理论基础。

将半结构化数据表示成某种图状（graph-like）或树状（tree-like）结构，是研究界较普遍的认识[4,5]。本文对文[6]提出的数据模型和查询语言加以改造和扩展以适应 XML 的数据表示。本文提出的半结构化数据模型实际上是一种“边加标签的带根有向图”（an edge-labeled rooted directed groph）。下面先给出它的形式化定义，然后在此基础上引申出 XML 数据模型 XTree 的定义。

定义1（边加标签的带根有向图）　一个带根、边加标签有向图是一个四元组 $G=(V,E,r,\lambda)$。其中 V 是一个非空的结点集合；$E\subseteq V\times V$ 是图中边的集合；(V,E) 代表一个有向多重图（directed multi-graph）；$r\in V$ 表示根，并且满足，任意一个 V 中的节点，从 r 开始经过 E 中的边都能够到达；$\lambda: E\rightarrow Label$ 是一个函数，表示边集合 E 到标签集合 Label 的映射。

定义2（XML 数据模型（简称 XTree））　论域 D 是包括所有字符串类型数据的集合，D 上的所有带根、边加标签有向图组成的集合用 T 来表示 $T\subseteq G$，T 中的元素 t，要么是一个原子数据，即有 $t=d\in D$；要么为 $t=d[t_1,\cdots,t_n]$，其中 $d\in D, t_1,\cdots,t_n\in T$，称 d 为标签（label），$t_1,\cdots,t_n$ 为 t 的子树。

$t=d[t_1,\cdots,t_n]$ 对应到 XML 文档数据，称 t 为 XML 树，简称 XTree，它有如下属性：

·树的每一个节点都有一个唯一的标识符（ID）。这个标识符可以显式地以 XML 文档某一元素的 ID 属性来标识，也可以为其分配一个唯一的 ID 来标识；

·以 XML 的术语，t 为元素（element），非叶标签 d 为元素类型（element type），$t_1,\cdots,t_n$ 为 t 的子元素（subelements）；叶标签 d 为原子对象时，为文字值或空元素；

·当考虑 $[t_1,\cdots,t_n]$ 中各子树的顺序时，称为顺序的 XTree（ordered XTree）模型，当不考虑 $[t_1,\cdots,t_n]$ 中各子树的顺序时，称为无顺序的 XTree（unordered XTree）模型。

图1是一个 XML 文档实例，图2是该文档的 XTree 模型表示的示意，因篇幅所限，此处没有给出该类文档的 XML DTD[9]，从下面的查询举例中可以看出该类文档组成的端倪。

```
〈bib〉
 〈book〉
  〈title〉A Great Book〈/title〉
  …
  〈publisher〉
   〈publishername〉Pt Hall〈/publishername〉
   〈address〉No8 Tree Street〈/address〉
  〈/publisher〉
 〈/book〉
〈book〉
 〈title〉A Good Start〈/title〉
 …
 〈publishername〉Wrox〈/publishername〉
```

李效东　博士生，专注于与数据管理技术相关的领域的研究。博士阶段的学习和研究集中于 XML，半结构化数据，数据集成查询处理及优化等。

顾毓清　研究员，博导。

```
〈/book〉
〈article〉
  …
〈a/article〉
…〈/bib〉
```

图1　一个 XML 数据文档实例

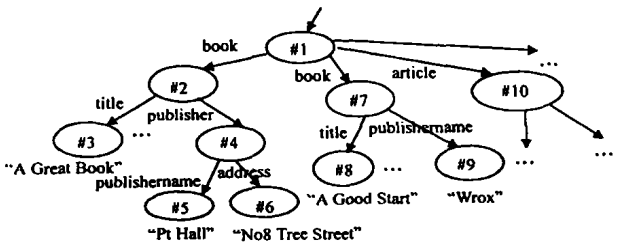


图2　一个 XML 文档 XTree 模型表示示意图

路径表达式是 XML 查询的基本建筑单元，针对模型 XTree 我们给出它的形式化定义：

定义3(路径表达式)　(1){}表示一棵空树；(2){l=>T}表示一棵树，它的根有一条标记为 l 的输出边，下面附着一棵子树 T；其中 T 可以表示叶节点，此时 T 等于某文字值或空元素。由于{l=>T}只表示单一一棵树，可省略括号，简写为 l=>T；(3){l_1=>T_1, l_2=>T_2, …, l_n=>T_n}也表示一棵树，它有 n 个由同一节点发出的、分别标记为 $l_1, l_2, \cdots, l_n$ 的输出边，每个输出边分别附着子树 $T_1, T_2, \cdots, T_n$；(4)无论是边标签 l，还是子树 T_i(包括叶节点)，都可以用变量表示；(5)2，3或2，3的组合称为路径表达式。

AnXQL 查询语句由两个重要的部分组成，一个称为构造部分(construction part)，另一个称为抽取部分(extraction part)。抽取部分由模式匹配条件表达式和称为过滤器(filter)的谓词条件表达式组成。构造部分由 select 子句来表达，而抽取部分由 where 子句来表达。所谓模式匹配就是用路径表达式来定义查询条件，将路径表达式做为一个模式去原文档中进行匹配，满足条件的 XML 文档片断被返回；而谓词表达式是指结果为布尔类型的条件表达式。

例如针对与图1的 XML 文档，可能有查询请求 Q1如下：

```
Q1:SELECT result=>{book=>{title=>
    $x,price=>$y,author=>$z}}
WHERE bib=>{book=>{title=>$x,
    price=>$y,author=>$z}}
    ADN $y<=50 IN "www.a.b.c/bib.xml";
```

其中，WHERE 子句由一个路径表达式和一个谓词表达式($y<=50)组成。如果进一步细分，查询 Q1实际上由三部分组成：构造部分、匹配部分和过滤器部分。三个部分之间，数据的传递可以视为元组(tuple)的传递，元组具有扁平和顺序无关的特性，这是 AnXQL 查询语言的一个重要特征。"IN"子句用来指定查询的文档所在的 URI (Unified Resource Identifier)，为简单起见，在下面的叙述中对单一数据源查询的 AnXQL 语句省略 IN 子句。首先，路径表达式到原文档中去匹配，结果是满足匹配模式的值绑定(下面给出变量绑定的定义)到元组集合($x, $y, $z)，然后元组集合被传递给过滤，变量 $y 小于等于50所对应的元组被保留，并被传递给构造部分，按 select 子句的路径表达式为每一组元组加标签(tagging)生成新的 XML 模型表示。作者在文[8]中，给出了 AnXQL 查询的详细语法功能特性描述。

2. XML 查询的代数表示

在传统的 DBMS 中，将一种声明型的查询语言翻译成一种等价的代数表示是通常的做法[10]，如对 SQL，OQL 等数据库查询语言的处理。代数表示的目的可以分为两方面，第一，代数表示给出了该查询语言的一种语义说明，因此代数的每一个操作必须被严密而准确地定义；第二，依靠这种代数表示，XML 查询的可操纵性(manipulable)更强，从而便于进行基于重写技术的查询优化。

2.1　变量绑定

所谓 XML 的查询代数是在 XTree 数据模型上的一系列操作的集合。这些操作或这些操作的结合能够用于对整个文档、文档的部分进行选择，并且从选择出来的部分中生成新的 XRL 模型表示。代数操作主要用于对查询的执行进行具体表示。在查询执行之时，一个说明型(declarative)的面向用户的 AnXQL 查询请求，将首先被翻译成它的相应的代数表示。

在引入代数操作符之前，先对一个重要的概念加以明确。

定义4(变量绑定，Variable Binding)　在查询处理过程当中，路径表达式中某个节点被赋给变量 x(或临时变量 x)，我们称变量 x 被绑定到该节点；也可称为该节点绑定到变量 x。

XML 查询处理的一个关键运算就是，给定一个由几个路径表达式组成的匹配模式，生成变量绑定的集合，即当路径表达式与文档的某一片断匹配的时候，就生成到某变量的绑定。AnXQL 中的 select 子句用来描述如何在这些变量绑定的基础上生成查询的应答结果。

下面我们举例说明。在图1的 XML 数据文档上，设有 AnXQL 查询 Q2如下：

```
Q2:
SELECT bib=>book=>{title=>$x,
        publisher=>$y}
WHERE bib=>book \$z AND
        $z=>{title=>$x,_*=>
            publishername=>$y};
```

这是一个带有正则路径表达式的查询语句，"_"表示通配符，"*"表示重复[8]，所以"_*"表示任意类型的零个或多个边(edges)。

当对一个 XML 数据文档实施查询的时候，XML 数据首先被解析(parse)，然后为该文档在内存中建立起一种树型的数据结构表示(当遇到大的 XML 文档的时候，相应的内存管理方法是必要的，如 LRU(Least Recent Used)等)，图1文档的数据结构示意图与图2类似，此处不再另行给出。其中 #i 表示系统为各个节点分配的唯一的标识符。在查询语句当中每一个路径表达式都对应着一个变量绑定列表。显然，满足查询 Q2的第一个路径表达式的节点分别为 #2和 #7，将它们分别绑定在变量 $z 上，得到绑定列表如下：

式1:[〈$z/#2〉,〈$z/#7〉]

第二个路径表达式依赖于变量 $z，显然根据图2我们可以得到与其对应的变量绑定列表如下：

式2:[〈$x/#3, $y/#5〉,〈$x/#8, $y/#9〉]

必须说明的一点是，查询语句中的变量到 XML 树中某一节点的绑定不仅代表着该节点本身，而且包括该点向外发散的边所连结的所有子树，下节将给出变量绑定的更一般的表达方式，该种表达方式将变量绑定到节点和从该节点出发的边所连结的所有子树。

2.2　查询的代数表示

每一个 AnXQL 查询都有一个等价的代数表达式与之对应。代数操作符以一系列变量绑定的列表为输入，产生一系列新的变量绑定列表做为输出。把上面的变量绑定以一种更方

便的树型结构来表达[11,12]，便于代数操作符的说明。例如，我们可以将如图1的整个 XML 文档看成是绑定在根节点变量 $root 上，则将这种变量绑定可以以树型结构表示成图3(A)的形式，而上小节的式子1和式子2的变量绑定可以分别表示成图3的(B)和(C)的形式。

```
bs[b[ $root[
 book[
   title["A Great Book"],
   …,
   publisher[
       publishername["Pt Hall"],
       address["No8 Tree Street"]
           ]
       ],
 book[
   title["A Good start"],
   …,
   publishername["wrox"],article[
   …,
   ]
       …]]]
```

(A)

```
bs[b [ $z[
     title["A Great Book",]
     …,
     publisher[
     publishername["Pt Hall"],
         address["No8 Tree Street"]]]],
     b[ $z[
         title["A Good Start"],
         …,
         publishername["Wrox"]]]]
```

(B)

```
bs[b [ $x["A Great Book"], $y["pt Hall"]],
     b[ $x["A Good Start"], $y["Wrox"]]]
```

(C)

图3　变量绑定的树型结构表示

从图3可以看出这种树型结构以 bs[…]包含若干个变量绑定 b[…]的形式给出。可以用如下的一般形式来表示：

```
bs[b[ $x[A[…[x1]]], $y[B[…y1]]],…],
     b[ $x[A[…[x2]]], $y[B[…[y2]]],…],…]
```

其中，$x 和 $y 是节点及其连结的子树所绑定的变量，A 和 B 是 XML 的元素名，或者称模型表示中的边标签(edge label)，x1，y1，x2，y2是 XML 元素所包含的文字值(#PCDATA)。当文字值直接绑定在变量上的时候，则不会有一般形式中类似 A，B 这样的元素名出现(如图3中的(C))。

本文所采用的代数表示，基本类似于嵌套关系代数[13]，我们称其为 AnXQL 代数。在关系型数据库方法中，“一切”都可以表示成 table，而在嵌套关系方法中，数据由元组(tuples)和列表(lists)组成，可以任意地嵌套。例如一个表示书目信息的数据由一个元组列表组成，每个元组表示一本书，而元组的某一个域(field)本身可能又是多个作者组成的列表。

AnXQL 代数操作包括与传统的关系运算，如选择(σ)、投影(π)、连接(▷◁)等，类似的运算符，只不过它们目前操作的对象是一系列变量绑定列表。除此之外，还增加了适合 XML 数据嵌套、层次结构特点的其它代数操作运算符。下面给出核心代数操作符的语义描述，并对各个操作符举实例加以说明。

·**bind**$_{vb,re\to nb}$　bind 操作以变量绑定列表 vb 为输入，用路径表达式 re 去匹配 vb，被匹配上的部分生成新的变量绑定列表 nb。允许以“_”，“|”，“*”作为路径表达式的组成，即 re 可以是正则路径表达式。

如针对图1的 XML 文档，运算 bind$_{\$root,bib=>book\to\$z}$理解为从整个文档(绑定在根节点变量 $root 上)的根节点开始去匹配路径表达式 bib=>book，将满足条件的节点(及其下的子树)帮定在变量 $z 上。实际上是以图3的变量绑定 A 为输入，输出变量绑定 B。bind 运算实现了投影运算的功能。

·**extract**$_{vb,re\to nb}$　extract 操作以变量绑定列表 vb 为输入，“抽取”路径表达式 re 所指定的 vb 中的文字值。从 extract 所完成的功能可以看出来，路径表达式 re 一定是指向某一(几)个文字(叶)节点的路径表达式，而在 bind 运算中没用这个要求。在 bind 运算中，re 既可以是指向文字节点的路径表达式，也可如 bib=>book 那样，并非指向某文字节点。extract 运算的结果是变量直接绑定在文字节点上(而非子树上)形成的变量绑定列表。

如运算 extract$_{\$x,\{title=>\$x,_*=>publishername=>\$y\}\to[\$x,\$y]}$可以理解为以变量绑定 $z 为输入，在其中抽取满足路径表达式{title=>$x,_*=>publishername=>$y}的 $x 和 $y 的值，生成新的变量绑定列表。如图3所示，这个操作实际上是以图中的 B 为输入生成变量绑定列表 C。

·**elementCreate**$_{label,vb\to nb}$，**elementCreate**$_{[label1,label2,\cdots],[b1,b2,\cdots]\to nb}$　elementCreate 运算分为两种形式，第一种形式为输入的变量绑定列表 vb 中的每一个变量绑定生成一个新的元素。参数 label 指定新元素的名字，可以是一个常数，也可以是一个变量，结果生成新的变量绑定列表 nb；第二种形式的输入是若干个变量绑定组成的列表，参数 label1、label2，…为每一个变量绑定(b1，b2，…)从左到右对应，依次指定一个新的元素名字，同样，labeli 可以是变量，也可以是一个常数，结果生成新的变量绑定列表 nb。

如查询 Q2的 select 部分可以表达成如下的两个 elementCreate 运算：

$$\text{elementCreate}_{[title,publisher],[\$x,\$y]\to\$temp}$$
$$\text{elementCreate}_{book,\$temp\to\$result}$$

以上两个运算的结果可以分别表示成图4的(A)和(B)。

可以看出，第一个 elementCreate 运算实际上是在 extract$_{\$z,\{title=>\$x,_*=>publishername=>\$y\}\to[\$x,\$y]}$运算的结果上实施，而第二个 elementCreate 运算又在第一个 elementCreate 运算的结果上实施。

```
bs[b [ $temp[title["A Great Book"],publisher["Pt Hall"]]],
     b[ $temp[title["A Good Start"],publisher["Wrox"]]]]
```

(A)

```
bs[b [ $result[book[title["A Great Book"],
      publisher["Pt Hall"]]]],
     b[ $result[book[title["A Good Start""],
       publisher["Wrox"]]]]]
```

(B)

图4　elementCreate 运算的输出结果举例

查询 Q2中，select 子句中的 bib 元素不作为新加入的元素由代数来处理，按照 XML 语法的要求，一个 XML 文档只能有一个顶层(top level)元素，因此 bib 只是做为一个顶层元素加入到查询结果当中形成一个完整的 XML 文档(见下面的操作符 xmlConstruct)。

·**select**$_{vb,pred\to nb}$　选择运算以变量绑定列表 vb 为输入，选择文字值满足某谓词表达式 pred 的变量绑定。

例如，把查询 Q2更改为如下的查询 Q3(在图1的文档中相应地加入图书出版的 year 元素)：

```
Q3:
SELECT bib=>{book=>{title=>$x,publisher=>$y}}
WHERE bib=>book\$z AND
        $z=>{title=>$x,_*=>publishername=>$y,
            year=>$w}
    AND $w>=1999;
```

则有以下的选择运算

$select_{[\$x,\$y,\$w],\$w>=1999\rightarrow[\$x,\$y,\$w]}$

其中，$w>=1999是一个谓词表达式，从变量绑定列表[$x，$y，$w]中选择那些1999年以后出版的图书组成新的变量绑定列表。

$\cdot groupBy_{\{v1,\cdots vk\},v\rightarrow[\{v1,\cdots vk\},[v]]}$ 分组运算与关系模型下的分组运行有很大不同，具有典型的嵌套关系模型或者面向对象模型的特点。在关系模型的带有分组运算的SQL查询中，select子句中一定有一个(或几个)在某一个属性上的聚集(aggregation)函数[10]。而在对XML模型表示的查询中却没有这种限制。$groupBy_{\{v1,\cdots vk\},v\rightarrow nb}$表示按照变量绑定v1，…，vk(称为group-by变量)对变量绑定v进行分组，对于每一个符合group-by变量的变量绑定，都生成一个新的变量绑定，最后生成列表[{v1，…vk}，[v]]。例如针对图1中的XML文档有这样的查询请求"给出文档中所出现的所有出版社的列表，并在每个出版社下面列出该出版社出版的图书。"很显然这个查询请求需要对所有的图书按照它所属的出版社进行分组，有AnXQL查询如下：

```
Q4:
SELECT bib=>{[$y] book=>{publisher=>
        $y,[$x] title=>$x}}
WHERE bib=>book \$z AND
      $z=>{title=>$x,_*=>publishername=>$y};
```

AnXQL中分组查询语义在文[8]中有较详细的说明，[$y]表示为元组集合中每一个不重复的绑定到$y变量的值生成一个树结构book=>{publisher=>$y,[$x]title=>$x}，而树结构中的[$x]表示为与这样的$y值对应的每一个$x值生成一个树结构title=>$x。此处我们来看看代数groupBy操作符。Q4查询中的分组运算表示成如下的代数操作：

$GroupBy_{[\{\$y\},\$x]\rightarrow[\$y,[\$x]]}$

将变量绑定$x按照变量绑定$y来分组。一个[$y，[$x]]变量绑定列表的结果可能如图5所示。

```
bs[b [$y["Pt Hall"], $x["A Great Book"], $x["XML Query"],
      $x["Wrapper Generation"]],
   b[ $y["Wrox"], $x["A Good Start"], $x["Inside XML"],
      $x["Data on the Web"]]]
```

图5 分组运算结果举例

$\cdot join_{vb1,\cdots,vbk},\{vb1.\$x=vb2.\$y,\cdots\}\rightarrow nb$ 与在关系系统中的查询一样，连接运算是一个非常重要和常用的代数操作。它在来自一个或多个XML文档的某值上做连接。与其它运算一样，join运算以多个变量绑定列表{vb1，…，vbk}做为输入，按照条件{vb1.$x=vb2.$y，…}生成新的变量绑定列表。如图6，给出了一个有关书籍和对应书评的新的XML文档，它与图1所示的文档按照书名进行连接，并列出书名、出版社和相应书评，查询语句如Q5所示：

```
Q5:
SELECT result=>book=>{title=>$x,
 publisher=>$y,review=>$w}
WHERE bib=>book\$s AND
      $s=>{title=>$x,_*=>publishername=>$y}
      IN S1 AND
      reviews=>book\$t AND
      $t=>{title=>$v,review=>$w} IN S2 AND
      $x=$v;
```

表示成join运算如下：

$join_{[\{\$x,\$y\},[\$v,\$w]],\$x=\$v\rightarrow[\$x,\$y,\$w]}$

```
<reviews>
  <book>
    <title>A Great Book</title>
    <review>A fine book.</review>
  </book>
  <book>
    <title>A Good Start</title>
    <review>This is interesting.</review>
  </book>
  <book>
    ...
  </book>
  ...
</reviews>
```

图6 一个XML文档

一个[$x，$y，$w]绑定列表的可能结果如图7所示。

```
bs [b [ $x["A Great Book"], $y["Pt Hall"], $w["A fine
       book."]],
     b[ $x["A Good Start"], $y["Wrox"], $w["This is inter-
       esting."]]]
```

图7 可能的连接运算的结果

$\cdot xmlConstruct_{vb,label\rightarrow xmlDoc}$ xmlConstruct运算将形如bs[b[],b[],…]的变量绑定列表拆散，返回每一个绑定的元素和文字值，并在它们上面加一个顶层元素label，生成新的XML文档xmlDoc。如在图4运算elementCreate的结果B上实施xmlConstruct运算，有：

$xmlConstruct_{\$result,bib\rightarrow xmlDoc}$

xmlDoc表示最终的查询Q2的select语句生成的XML结果。

$\cdot source_{url\rightarrow y}$ source操作符为在指定url处的文档的根节点元素e生成由一个单一变量绑定构成的列表：bs[b[v[e]]]。

每一个AnXQL查询都对应着一个等价的代数表达式，这个代数表达式实际上就构成该查询的一个初始的查询计划(initial plan)。与其它数据库系统中一样，我们用查询树来表示查询计划，则前面给出的查询Q2，Q3，Q4和Q5可以被分别表示为如图8的A，B，C和D所示的查询树。

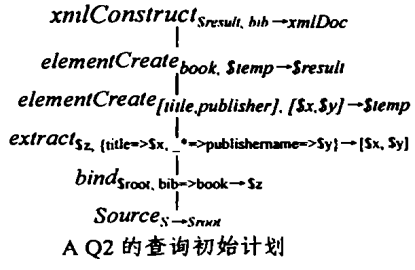
以上是我们给出的XML查询的一个核心代数操作符集合，它不是一个全功能的代数表达集合，这些代数操作能够很好地适应我们的母课题—基于半结构化数据模型下的数据集成系统[8]的研究与开发，在功能上是其它XML查询代数系统的子集。

除此之外，本文所给出的代数表示是逻辑层(logical level)的代数表示，而非物理层(physical level)的代数表示[10]，例如我们不能够给出这样的操作符，利用它可以借助索引结构来更高效地计算连接操作。这是集成系统的特点造成的。

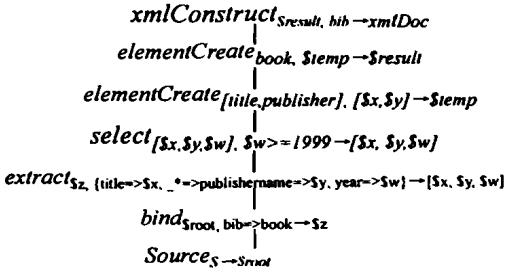
3. XML查询优化

由于本文的研究是文[8]研究的一部分，因此对查询优化的讨论主要在异构集成系统的前提下进行。XML的查询优化方面，虽然有一些研究项目和论文涉及文[14]，但是在异构数据源集成系统这个特殊的前提下进行的查询优化研究却很少。这是集成系统的特殊性所造成的，由于数据源的自治性，集成系统不可能知道数据源XML文档的物理存储方式，从而也就不可能利用数据源对XML的索引来高效地实施XML查询。而且大多数数据源并不直接是XML数据源，这时是在wrapper输出的结果上实施查询，wrapper输出的中间结果以文本文件存储，或以数据流的方式直接输入到某个查询操作当中。本节主要从代数重写角度探讨查询优化方法和可能性，主要原理是尽量减少扫描原文档的次数，或者减少不必要的中间层次(intermediate)的变量绑定，达到物理查询

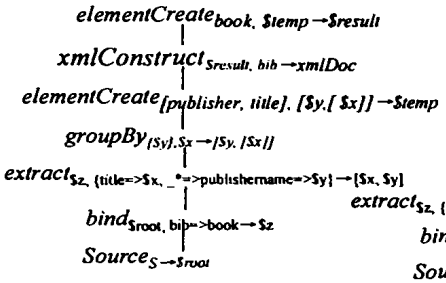
的效率最高。文[8]还研究了使用 XML 的 DTD 模式信息来优化带有路径表达式的 AnXQL 查询，由于与本文代数操作符的主题不符，此处不再赘述。



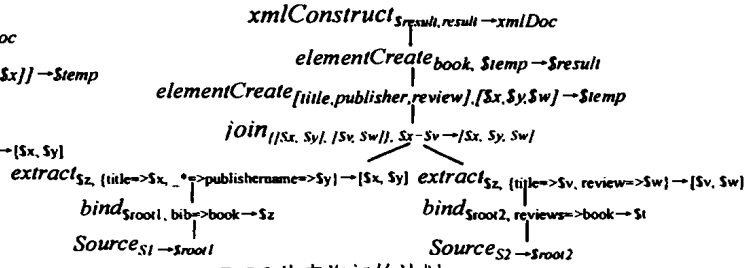
A Q2 的查询初始计划



B Q3 的查询初始计划



C Q4 的查询初始计划



D Q5 的查询初始计划

图8　查询初始计划(查询树)

我们先介绍基于代数重写的 XML 查询优化方法。

①通过合并 bind 和 extract 操作减少中间层次的变量绑定。如图8中的形如 bind$_{\$root,bib=>book=>\$z}$和 extract$_{\$x,\{title=>\$x,_*=>publishername=>\$y\}\to[\$x,\$y]}$可以合并为一个 extract 运算，为 extract$_{\$root,bib=>book=>\{title=>\$x,_*=>publishername=>\$y\}\to[\$x,\$y]}$，从而减少了中间层次的变量绑定 $z。但是必须注意的是，这种方法只有在 extract 运算依赖某一 bind 运算的结果的时候才起作用。当只有 bind 运算或只有 extract 运算的时候，则不可能实施这样的简化操作。

如针对图1的 XML 文档，有查询请求“输出所有的有关 articles 的信息”，相应 AnXQL 查询语句如下：

```
select result=>{$x}
where bib=>article/$x;
```

则此查询的初始查询计划中会有代数表达式 bind$_{\$root,bib=>article=>\$x}$。这个简单的查询不可能利用上面的简化方法。

②当查询请求有在一个数据源的 XML 文档上就某两个变量进行连接运算的时候，通过代数变换将对同一文档的多次扫描，变为一次扫描多个输出。这时必须引入一个新的代数运算符 scan。scan 操作符以一个变量绑定(一般是根节点的变量帮定 $root)和两个路径表达式为输入，输出两个分别与不同的路径表达式对应的变量绑定列表。它的一般表达形式如下：

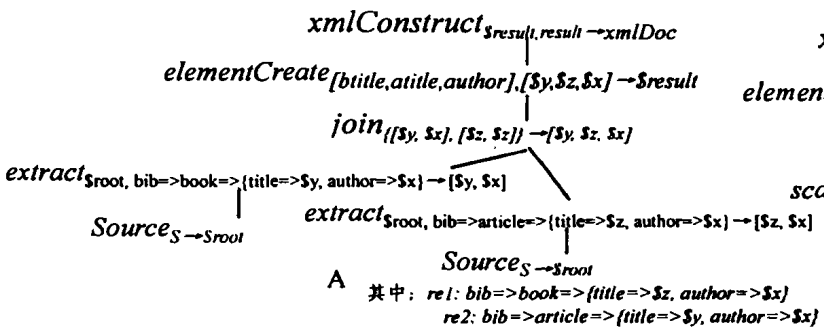
$$scan_{vb,re1,re2\to nb1,nb2}$$

其中 vb 代表输入的变量绑定列表，re1和 re2代表两个路径表达式，nb1和 nb2是分别与 rel 和 re2对应的新输出的变量绑定列表。

设针对图1的 XML 文档有这样的查询请求“找出同一作者参与写作的图书(books)和文章(articles)。”，有 AnXQL 查询语句 Q6如下：

```
Q6:
SELECT result=>{btitle=>$y,atitle=>$z,author=>$x}
WHERE bib=>book=>{title=>$y,author=>$x} AND
      bib=>article=>{title=>$z,author=>$x}
```

显然，这个查询的实施需要在变量 $x 上的连接运算。这个查询的初始计划与加入 scan 操作符的简化查询计划分别如图9的 A 和 B 所示(图中的连接运算是自然连接 natural join 运算)。



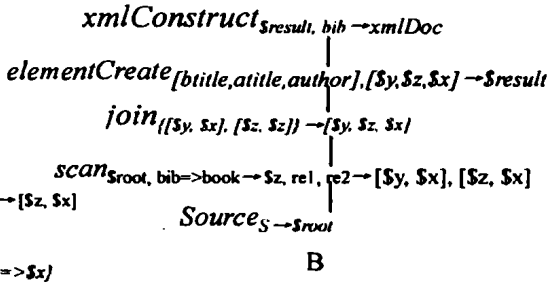


图9　查询 Q6的初始计划和简化查询计划

将图9中的 A 与 B 加以比较，会发现图 B 中的 scan 运算代替了图 A 中的两个 extract 运算，从而将对文档的多次扫描减少为一次扫描。scan 运算设计的出发点在于，在 XML 数据文档中，不同类型的数据项出现的顺序完全是随机的，如例子中的 book 数据项和 article 数据项，你不能预测它们出现的前后顺序，这时引进 scan 操作符将会有效提高与例子类似

的具有文档内部连接的查询的处理效率。

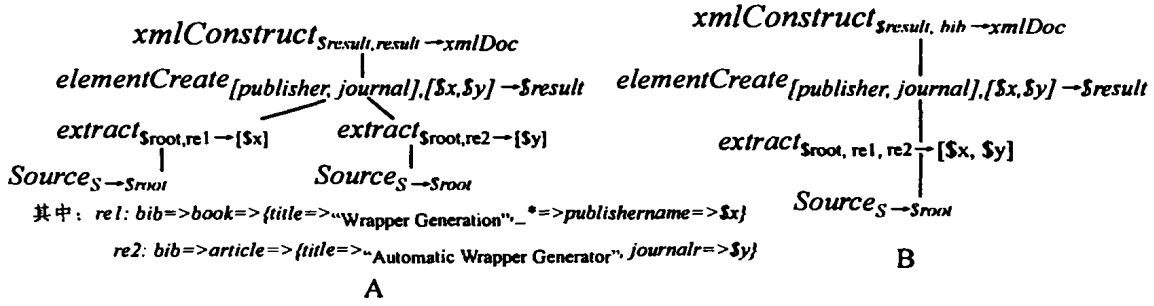


图10 查询Q7的初始计划和简化查询计划

③利用以下两个等价变换公式，可以与 scan 运算同理减少对原文档的扫描次数：

$$extract_{vb.re1\to nb1}, extract_{vb.re2\to nb2} \equiv extract_{vb.re1.re2\to[nb1.nb2]}$$

$$select_{vb.pred1\to nb1}, select_{vb.pred2\to nb} \equiv select_{vb.pred1.pred2\to[nb1.nb2]}$$

设针对图1的 XML 文档有查询请求"查询图书 Wrapper Generation 的出版社，文章 Automatic Web Wrapper Generation 所发表的刊物名。"相应的 AnXQL 查询语句 Q7如下：

```
Q7:
SELECT result=>{publisher=>$x,journal=>$y}
WHERE bib=>book=>{title=>"Wrapper Generation",
        _*=>publishername=>$x} AND
      bib=>article=>. {title=>"Automatic Wrapper Generator",
      journal=>$y};
```

图10的 A 与 B 分别是查询 Q7的初始查询计划和代数等价变换后的简化形式。显然这种变换减少了扫描原文档的次数。

select 操作与 extract 操作的原理完全相同，此处不再举例说明。

考虑查询语句和代数表达式，会发现代数表示与查询语句的子句或路径表达式都有一一对应的关系，所以查询语句到"代数表达式"的转换是很直接的转换。值得注意的是正因为是一种直接的转换，所以初始的代数表达式查询计划往往不是效率最优的，这时优化器应根据变换规则进行必要的优化处理。

4. 相关研究工作

XML 正逐渐成为 Web 上数据交换和表示的事实标准，围绕 XML 数据的模型表示和查询语言，代数操作符和查询优化方面的研究成为研究界和产业界的热点。

基于半结构化的数据模型表示，有多种 XML 查询语言提出，包括 XML-QL[15]，Lore[14]，YATL[16]，XQL[17] 等。AnXQL 查询语言的设计力求满足 XML 查询语言的设计需求[8]，并为自治异构数据源集成查询处理这个整体目的服务。

XML 查询代数表示的设计方案见诸于多篇论文，著名的有 David Beech 等[18]，Mary Fernandez 等[19]，以及 Catriel Beeri 等[20]。前两者的设计目标都是全功能的 XML 查询的代数表示，其中文[19]及其后续工作最有可能被 W3C[21]用作为草案或推荐标准。文[19]以函数式程序设计语言(functional programmig language)[22]来表示各种 XML 查询的代数操作符，完全区别于传统的代数操作符表示。AnXQL 代数表示沿用传统的查询语言的代数表示方式，表达方式简单便于理解和实现；功能上是文[19]代数表示的一个子集。

专门针对 XML 数据查询优化方面的研究论文较少，特别是在异构数据源集成系统的情境下研究 XML 查询优化的则更少。这是集成系统的特殊性所造成的，由于数据源的自治性，集成系统不知道数据源 XML 文档的物理存储方式，不可能利用数据源对 XML 的索引。Xscan[23]是所见不多的、致力于 Web 数据集成系统下 XML 查询优化研究的系统之一。在文[23]中，作者研究了在数据集成系统下，如何对非物化(non-materialized)的 XML 数据如何高效地实施查询处理，SAX API 设计了专门操作符 Xscan。Xscan 在"跨过网络而来"的数据流中匹配来自查询的正则路径表达式，并以管道的(pipelined)方式返回。Xscan 的设计获得了很高的查询效率。Xscan 的研究与本文的出发点有所不同，本文力图基于代数重写设计一个通用的 XML 查询的优化方法。

Lore[14]项目的目标是设计一个类似传统 DBMS 的 XML 数据管理系统，Lore 的设计基于面向对象型的半结构化数据模型 OEM[24]，文[14]中研究了 Lore 中 XML 数据的索引和查询优化方法。文[25]提出的基于图形模式(graph schema)的半结构化的查询方法用于对结构已知的 Web 数据源的查询，但是该方法不适用于对 XML 数据的查询优化处理。类似本文基于逻辑代数变换的 XML 查询优化方法的研究，少见文献论及。

参 考 文 献

1 Abiteboul S. Semi-Structured Data. In: Proc. of ICDT. Delphi, Greece, 1997. 1～18

2 http://www.acedb.org/

3 http://www.lotus.com/home.nsf/welcome/notes

4 Buneman P, et al. A Query Language and Qptimization Techniques for Unstructured Data. In: H. V. Jagadish, I. S. Mumick eds. Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data. Montreal: ACM Press, 1996. 505～516

5 Buneman P, et al. Adding structure to unstructured data. In: Proc. of ICDT, Jan. 1997

6 Buneman P, et al. A query language and optimization techniques for unstructured data. In: Proc. of ACMSIGMOD Intl. Conf. on Management of Data, Montreal, Canada, June 1996. 505～516

7 Bray T, Paoli J, Sperberg-McQueen C. Etensible Markup Language (XML) 1.0. Http://www.w3.org/xml/1998/06/xmlspec-reprot-19980910.htm

8 李效东. 自治异构数据源的集成查询处理：[中科院软件所博士论文]. (手稿)

9 Bosak J, et al. W3C XML Specification DTD. http://www.w3.org/XML/1998/06/xmlspec-report.htm

(下转第51页)

1)非合作的决策过程 在实验二中,中间商1采用Q学习策略,从而预先考虑到自己的定价给对手带来的影响,从长远出发,以自己长期赢利和最大为目标,定出了合理的价格,减小了价格战的幅度,故W_1比实验一时增加了30.2%,可见Q算法在定价过程中,很好地发挥了效果;另外,我们还发现,作为中间商1的对手,虽然中间商2依然采取myopic策略,但是它的赢利较实验一时也有提高,这也是由于价格战幅度减小的缘故,可以说,中间商1采取Q学习策略是一种对整个集体都有利的行为。同时,由于价格战的减弱,整个系统的交易率也随之提升,交易率的提高代表了顾客满意度的提高。当然,在交易率变化不大的前提下,中间商赢利提高,必然导致顾客总赢利的下降,但这并不是一件坏事,因为在市场中,对于顾客而言,需求被满足是更关键的事情。

在实验三中,与实验一相比,Q学习策略使得W_1提高了20.4%,W_2提高了33.4%,整体赢利提高了26.9%,更加有效地克服了价格战的危害。此外,实验三与实验二相比,整体赢利和交易率也都有提高。

综上可见,在非合作的决策过程中,强化学习可以有效地遏制价格战,稳定市场,既增加了中间商的盈利,也提高了顾客的满意度。

2)合作的决策过程 在实验四中,与实验三相比,由于中间商2采取了非合作策略,中间商1从它这里得不到任何好处,相反,由于中间商1采取了合作策略,把本属于自己的一部分顾客群也奉献给了中间商2,所以,W_1下降了24.8%,W_2提高了68.1%。此外,中间商1的合作行为使得整体赢利有所提高,但我们发现,这也导致了交易率的下降。

在实验五中,由于两个中间商均采取合作策略,因此每个中间商的赢利以及整体赢利都大大提高了,即使相对于实验四,整体赢利也提高了14.3%,这充分体现了合作的优点。但是,中间商们的合作行为导致交易率下降到19.7%,这是因为这里的合作类似于现实经济中的垄断行为。由于中间商的垄断,使得平均价格比以前四种情况的平均价格高出了许多(见图9),从而使得交易率下降。从图9还可以看出,中间商采取了合作的Q学习策略之后,价格战的幅度与周期比前面三种情况都有所减小,即更有效地削弱了价格战现象。此外,由于每个中间商的目标是以自己赢利最大化为主,兼顾考虑对手的赢利,也就是说,中间商们的目标并不一致,所以价格战仍然存在。

在实验六中,两个中间商的目标一致,价格稳定在定值,彻底结束了价格战,使得整体赢利达到最优。但是$W_2<0$,这会使得中间商2不愿意与中间商1合作,所以在这种情况下,往往需要合作双方事先定下规则,交易结束后,按照规则重新划分赢利。但在现实世界中,这种完全合作很难达成。

综上可见,在合作的决策过程中,强化学习可以进一步遏制价格战,稳定市场,大幅度地提高中间商的盈利。但由于中间商的合作是一种垄断行为,因此导致消费者的满意度有所降低。

另外,值得强调的是,无论是合作情况还是非合作情况,通过在不同的折扣因子γ下进行仿真实验,我们发现:随着γ的增大,若Q-函数收敛,整体赢利单调递增。

结束语 本文针对电子商务中信息资源交易的价格战问题展开讨论,在中间商Agent的定价过程中,利用强化学习算法,将中间商Agent的赢利目标从短期改为长期,并提出中间商非合作、合作两种情况下的强化学习模型。仿真试验表明了该方法的有效性,在非合作情况下,学习算法有效地遏制了价格战,提高了中间商的赢利与顾客的满意度;在合作情况下,学习算法可以为中间商进一步提高赢利,但顾客的满意度有所下降。

本文所依据的信息过滤经济模型在应用时,也有一定的限制。主要原因是:信息过滤经济模型假设中间商能够清楚地知道全局信息,而在现实世界中,实现起来有所困难,需要花费一定的咨询费用。因此,在下一步的研究中,可能包括的研究内容有:①在中间商仅了解到非完全信息的情况下,如何设计最优定价策略;②考虑咨询费用的情况下,如何设计最优定价策略。

参考文献

1 Kephart J O, Hanson J E, Skirmish J. Price-War Dynamics in a Free-Market Economy of Software Agents. Proceedings of ALIFE VI, UCLA, Los Angeles, CA, USA, MIT Press, Cambridge, MA, USA. UCLA, 1998. 26～29

2 Tirole J. The Theory of Industrial Organization. The MIT Press, Cambridge, MA, 1988

3 Kephart J O, Hanson J E, Skirmish J. Price and Niche Wars in a Free-Market Economy of Software Agents, MIT Press, Sep. 1998

4 Kaelbling L P, Littman M L, Moore A W. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 1996, 4: 237～285

(上接第62页)

10 Garcia-Molina H, Ullman J, Widom J. Database System Implementation Prentice-Hall, 2000

11 Hosoya H, Pierce B C. XDuce: A Typed XML Processing Language. http://www.cis.upenn.edu/～hahosoya/xduce.html, 2000. Full version

12 http://www.w3.org/TR/query-algebra/

13 Paredaens J, Gucht D V. Converting nested relational algebra expressions into flat algebra expression. ACM Transaction on Database Systems, 1992, 17(1): 65～93

14 Quass D, et al. Lore: A lightweight object repository for semistructured data. In: Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, Montreal, Canada, June 1996. 549

15 Deutsch A, et al. XML-QL: a query language for XML. W3C Note NOTE-xml-ql-19980819. See http://www.w3.org/TR/NOTE-xml-ql.

16 Christophides V, Cluet S, Simeon J. On wrapping query languages and efficient XML integration. In: Proc. of ACM Conf. on Management of Data(SIGMOD), Dallas, Texas, May 2000

17 Robie J, Lapp J, Schach D. XML query Language (xql). Workshop on XML Query Languages, W3C Dec. 1998

18 Beech D, Malhotra A, Rys M. A Formal Data Model and Algebra for XML. W3C XML Query owrking group note, Sep. 1999

19 Fankhauser P, et al. The XML Query Algebra, W3C Working Draft, 15 February 2001. http://www.w3.org/TR/2001/WD-query-algebra-20010215/

20 Beeri C, Tzaban Y. SAL: An algebra for semistructured data and XML. In: Informal Proc. of Workshop on The Web and Databases, ACM SIGMOD, June 1999

21 World Wide Web Consortium. http://www.w3.org/

22 http://www.haskell.org/

23 Ives Z G, Levy A Y, Weld D S. Efficient Evaluation of Regular Path Expressions on Streaming XML Data: [Technical Report UW-CSE-2000-05-02]. University of Washington

24 Goldman R, Chawthe S, Crespo A, McHugh J. A Standard Textual Interchange Format for the Object Exchange Model (OEM). Department of Computer Science, Stanford University, California, USA, 1996

25 Fernandez M, Suciu D. Optimizing Regular Path Expressions Using Graph Schemas. In: Proc. ICDE, 1998