ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.4, April 2009, pp.795−803
doi: 10.3724/SP.J.1001.2009.03268

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

一种高效非归并的 XML 小枝模式匹配算法*

陶世群+, 富丽贞

(山西大学 计算机与信息技术学院,山西 太原 030006)

An Efficient Algorithm of XML Twig Pattern Matching Without Merging

TAO Shi-Qun+, FU Li-Zhen

(School of Computer & Information Technology, Shanxi University, Taiyuan 030006, China)

+ Corresponding author: E-mail: tsq@sxu.edu.cn

Tao SQ, Fu LZ. An efficient algorithm of XML twig pattern matching without merging. *Journal of Software*, 2009,20(4):795−803. http://www.jos.org.cn/1000-9825/3268.htm

Abstract: In an XML database, finding all occurrences of a twig pattern is a core operation for XML query processing. In the past few years, many algorithms, such as Holistic Twig and TJFast, were proposed in the literatures. However, these algorithms are based on merging, with high computational cost. Recently Twig2Stack algorithm and TwigList algorithm are proposed to resolve this problem, but they are very complex. Aim at this problem, this paper considers the characteristic that most path expressions have only a few output nodes, and proposes two new algorithms without merging, named TwigNM and TwigNME, which use only a few stacks. Finally, the experimental results show that these algorithms are superior to the previous algorithms, especially for only ancestor-descendant relationship in XPath.

Key words: twig pattern matching; merging; main path; main node; predicate node

摘 要: 在 XML 数据库中,小枝模式查询是 XML 查询处理的核心操作.近几年,研究人员已提出许多种算法,如 Holistic Twig 和 TJFast 算法等.然而它们都是基于归并的,会有很高的计算代价.已提出的 Twig2Stack 和 TwigList 算法虽然可以克服这一点,但算法非常复杂.针对这一问题,尤其是考虑了通常查询表达式中只有少数几个结点是最终的输出结点这一特点,提出了 TiwgNM 算法及其扩展算法 TiwgNME 算法.算法不需要归并,且只用了少数栈来实现.实验结果表明,这些算法优于以前算法,尤其是对查询中只有祖先-后裔关系的表达式更有效.

关键词: 小枝模式匹配;归并;主路径;主结点;谓词结点

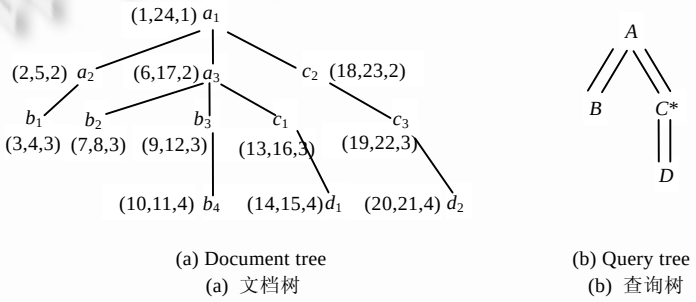
中图法分类号: TP301 **文献标识码**: A

随着 XML 的迅速发展,XML 的数据管理,特别是查询问题引起了众多学者的广泛关注,成为数据库领域的研究热点.在查询 XML 文档时,结构连接是最重要的查询方法之一.近年来,出现了直接归并连接算法,如 MPMGJN[1],EE-Jion/EA-Jion[2]等.但它们要重复扫描列表,这样会产生大量的中间结果.文献[3]提出了一种基于

* 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 (高等学校博士学科点专项科研基金)

Received 2007-07-22; Accepted 2008-02-04

栈的算法 Stack-Tree,文献[4]提出了一种基于区域划分的结构连接算法 Range Partitioning Jion,这些算法都是将小枝模式分解成一系列二元结构求解,然后再归并,这在一定程度上避免了重复扫描列表,但这样仍会产生大量的中间结果.为避免产生大量的中间结果,文献[5]提出了一种完整的小枝模式匹配算法 TwigStack,文献[6]提出了一种完整的小枝索引结构 XR-tree,利用 XR-tree 能够高效地跳过那些不参加连接的祖先和后裔结点.文献[7]提出了"流"的思想,利用流来加速小枝模式的处理过程.文献[8]提出的 TwigStackList 算法能够更好地处理小枝模式中的父子关系.文献[9]采用了一种扩展的 Dewey 编码方法,提出了 TJFast 算法,它只需访问叶子结点.但是,以上这些算法都是基于归并的,所以并不能避免大量的不必要的路径归并[10].例如,图 1(a)所示的是一个 XML 文档树 T,图 1(b)所示的是一个查询树 Q,它对应的查询表达式为 Q=A[//B]//C[//D].图中*所指的是输出结点(下同).在算法 TwigStack 中,(a_3,b_2)满足 Q_1=A//B,(a_1,c_2,d_2)满足 Q_2=A//C//D,但它们合起来并不满足 Q.这就产生了不必要的路径归并,使得算法的复杂度变大.文献[11]提出了一种算法,利用最近的共同的祖先结点来减少搜索空间,但该算法只有当输出结点是查询树中的叶子结点时才是高效的.2006 年,Chen 等人在文献[12]中提出了 Twig^{2}Stack 算法,2007 年,Lu 等人在文献[13]中提出了 TwigList 算法.这两种算法分别使用了层次栈和队列而不需要归并,因而优于 TwigStack 算法和 TJFast 算法等,但却会产生大量的随机访问,并且算法复杂.虽然它们都优于以前的算法,然而它们都没有考虑到对常用的 XPath 片段(包含/,//,[]和结点测试),通常查询只要求有一个结点输出,并不需要输出所有的结点.即使在复杂的 XPath 和 XQuery 的查询树中,也只有少数几个结点是最终的输出结点.本文针对小枝模式结构连接的有效处理问题进行了研究,特别是考虑了通常查询表达式中只有少数几个结点是输出结点这一特点,提出了 TiwgNM 算法以及其扩展算法 TiwgNME 算法,算法不需要归并,实现简单.在 TreeBank 标准文档集和生成的 XMark 文档集上进行了大量的实验,将我们的算法与典型算法 TwigStack 以及当前最新算法 Twiglist 进行了分析比较.实验结果表明,我们的算法优于它们,尤其是在只有祖先-后裔关系的 XPath 查询中,该算法效率很高.



(a) Document tree (a) 文档树　　(b) Query tree (b) 查询树

Fig.1 An XML document tree and a query tree

图 1 XML 文档树和查询树

本文第 1 节给出背景知识和相关定义.第 2 节中介绍 TwigNM 算法及其扩展算法 TwigNME,并进行代价分析.第 3 节对实验加以说明,并给出实验结果.第 4 节是结论以及进一步的工作.

1 背景知识和相关定义

在基于小枝模式匹配的 XML 查询处理中有两个关键的问题:(1) 如何快速判断 XML 文档树中任意两个结点之间的结构关系,包括父子关系和祖先-后裔关系;(2) 如何从给定的 XML 文档中高效地得到满足小枝模式结构关系的所有数据.在这两个问题中,结点间结构关系的判断依赖于对结点的合适编码,从结点的编码就可以判断出结点间的关系;而高效地得到所有满足结构关系的结点,则是所要解决问题的核心.

1.1 结点编码

对于 XML 文档树的编码方案,可以分为两大类:基于区间的编码和基于路径的编码.基于区间的编码方案是根据每一个元素结点在原 XML 文档中顺序的位置,给元素结点赋予一个编码;而基于路径的编码方案则是利

用 XML 文档的嵌套结构,给从文档根结点开始所能到达的每个路径和元素结点赋予一个编码.已经提出的编码方案主要包括位向量编码[14]、前缀编码[15]、区间(region)编码[1,11]和二叉树编码(PBiTree encoding)[16]等.

区间编码方法是目前所提出的编码方法中应用得最普遍的一种.它赋予文档树中每个结点一个编码(*start*,*end*,*level*),*start*,*end* 和 *level* 分别表示结点在树中前序遍历的起始和结束位置以及层次.对任意两个不同的元素结点 u 和 v,它们的编码(*start*,*end*)所覆盖的区间要么完全包含,要么不相交.本文采用区间编码方案.

1.2 小枝模式查询

XPath 是一种常用的 XML 查询语言,然而实际使用的 XPath 查询往往只用到了 XPath 语言的一个片段,其中较常用的 XPath 片段包含孩子轴(/)、后裔轴(//)、谓词和结点测试,这个片段通常表示为 XP{/,//,[]}.XPath 查询语句经常用树的形式来表达,这就叫小枝模式,XPath 查询也称小枝模式查询.下面给出其形式化定义.

定义 1. 一棵 XML 文档树 T 是一个四元组(r_T,N_T,E_T,λ_T),其中,$N_T\subseteq N$ 是所有结点的集合,$r_T\in N_T$ 是唯一的根结点,$E_T\subseteq N_T\times N_T$ 是树中所有边的集合,而$\lambda_T:N_T\rightarrow\Sigma$是一个映射函数,用来决定每个结点的类型.$\Sigma$表示 T 中结点的类型集.

定义 2. 一个小枝模式查询 Q 是$\langle t_Q,o_Q\rangle$,其中,$t_Q=(r_Q,N_Q,E_Q,\lambda_Q)$是一棵树;$E_Q$ 被分为两个互不相交的集合 C_Q 和 D_Q,它们分别表示所有孩子边(/)和后裔边(//);o_Q 是输出结点.

定义 3. 给定小枝模式查询 Q 和文档树 T,N_Q,N_T 分别表示 Q 和 T 的结点集,*lab*(v)表示结点 v 的类型标签,Q 在 T 上的匹配是一个映射 $f:N_Q\rightarrow N_T$,并且满足:

(1) 保持结点类型,即$\forall v\in N_Q$:*lab*(v)=*lab*(f(v));

(2) 保持边关系,即$\forall v,u\in N_Q$,若(u,v)是孩子边,则在 T 中 f(v)是 f(u)的孩子结点;若(u,v)是后代边,则在 T 中 f(v)是 f(u)的后代结点.

对于 T 中的一个元素 e,若 e 在 Q 的一个匹配中,则称 e 满足于 Q.那么,最终查询的结果一定满足于 Q,且是与 o_Q 类型相同的元素的集合.例如,图 1 所示的查询 Q 最终的匹配结果是{c_1,c_2,c_3}.

1.3 相关的概念和定义

定义 4. 在查询树 Q 上,从根结点到输出结点的那条路径称为主路径.在主路径上的结点,叫做主结点;不在主路径上的结点,叫做谓词结点.

如果在 Q 中只有一个输出结点,则 Q 中只有一条主路径.例如,在图 1(b)所示的查询树所对应的表达式 Q=A[//B]//C[//D]中,C 是要求输出的结点,A//C 是该查询树的主路径.对于“[]”中的 D 结点,因为计算查询表达式的最终结果是不会被输出的,所以它一定不在主路径中.输出结点 C 是主路径的叶子结点.A 和 C 是主结点,B 和 D 是谓词结点.在此很容易得知,谓词结点的后裔结点一定是谓词结点.

定义 5(标签流[7]). 在查询树 Q 中的一个结点 A 的标签流 T_A 是指在文档树 T 中所有具有相同标签名的元素结点的集合.T_A 中的元素结点是有序的,流指针所指的元素为头元素.

例如,图 2 中 T_A,T_B,T_C 和 T_D 是结点 A,B,C,D 对应的标签流,它们分别与图 1(a)文档树的元素相对应.

为了方便起见,下面将本文中所用到的几个概念和记号说明如下:文中提到的结点概念是指查询树中的结点,用 V_k 表示;提到的元素结点是指 XML 文档树中的元素结点,用 e 表示.结点 V_k 的标签流 T_{V_k} 中头元素结点用 E_{V_k} 表示.*parent*(V_k)返回 V_k 在 Q 中的父结点,*descendant*(V_k)返回 Q 中 V_k 的所有后裔结点.*pchild*(V_k)返回 Q 中 V_k 孩子中是谓词结点的所有结点.*dchild*(V_k)返回 Q 中结点 V_k 在主路径上的孩子结点集合.*child*(V_k)返回 Q 中结点 V_k 的所有孩子结点.其中,*child*(V_k)=*dchild*(V_k)∪*pchild*(V_k).

定义 6. 当一个谓词结点 V_p 满足以下条件时,则称这个谓词结点 V_p 有扩展:

(1) T_{V_p} 不空;

(2) 在 Q 中 V_p 为叶子结点;

(3) 若在 Q 中 V_p 不是叶子结点,则$\forall V_k\in$*child*(V_p):V_k 谓词结点有扩展且 E_{V_p} .*start*< E_{V_k} .*start*< E_{V_p} .*end*;

例:查询树 Q_3 如图 3 所示,Q_3 各结点的标签流如图 2 所示.谓词结点 V_d 是叶子结点且 T_{V_d} 不空,所以 V_d 是有

扩展的;谓词结点 V_c 的 T_{V_c} 不空,且 $c_1.start<d_1.start<c_1.end$,所以此时谓词结点 V_c 也是有扩展的.

定义 7. 当一个主结点 V_t 满足以下条件时,则称这个主结点 V_t 有扩展:

(1) T_{V_t} 不空;

(2) 在 Q 中 V_t 为叶子结点;

(3) 若 V_t 不是叶子结点,则$\forall V_k \in child(V_t)$: $E_{V_t}.start< E_{V_k}.start< E_{V_t}.end$,且

① $\forall V_p \in pchild(V_t)$:$V_p$ 谓词结点有扩展;

② $\forall V_d \in dchild(V_t)$:$V_d$ 主结点有扩展.

例:查询树 Q 如图 1(b)所示,在 Q 中,$child(V_c)=\{V_d\}$,$pchild(V_c)=\{V_d\}$,$dchild(V_c)$为空.Q 各结点的标签流如图 2 所示:T_{V_d} 不空,且 V_c 的孩子结点 V_d 是叶子结点,所以谓词结点 V_d 是有扩展的.与图 3 不同,这里的 V_c 是主结点而不是谓词结点.T_{V_c} 不空且 $c_1.start<d_1.start<c_1.end$,所以此时主结点 V_c 也是有扩展的.

引理 1. 一个结点 V 有扩展,则 E_V 一定满足于以 V 为根的子查询树 Q_V.

给定结点 $V_i \in descendant(V)(1\le i\le n)$,若 V 有扩展,由定义 3 可知,以 V 为根的子查询树 Q_V 在文档树 T 上的匹配一定是一个既保持结点类型又保持边关系的映射.而由定义 6 和定义 7 可知,这时,由各结点的头元素组成的元组$(E_V, E_{V_1}, E_{V_2}, \ldots, E_{V_n})$正是 Q_V 的一个匹配,所以 E_V 满足 Q_V.

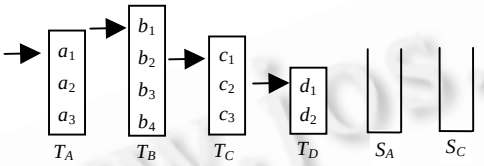


Fig.2 Tag streamings and stacks

图 2 标签流和结点栈

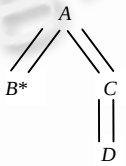


Fig.3 A query tree Q_3

图 3 查询树 Q_3

2 一种新的小枝模式匹配算法 TwigNM

该算法的主要思想是:考虑到实际小枝模式查询时通常只有少数几个结点是要输出的,且在多数情况下只要求一个输出结点;又因为谓词结点不会是输出结点,所以不需要求出所有满足 Q 的元素结点.因此,只需考虑那些能使主结点有扩展的元素结点.该算法为每个主结点 V_t 建立一个栈 S_{V_t} 和一个标签流 T_{V_t}.只为每个谓词结点 V_p 建立标签流 T_{V_p},而无须为其建立栈.在该算法中对于一个结点 V,如果它有孩子结点 V_1,在考虑 V 是否有扩展时则只需考虑 T_{V_1} 的头元素,不再需要考虑 S_{V_1} 中的元素,最后也不需要归并.

2.1 TwigNM算法

算法 1. *TwigNM*(Q,T).

输入:查询树 Q 有 n 个结点$\{r_Q,V_1,V_2,\ldots,V_{n-1}\}$,以 r_Q 为根,T 为 Q 中各结点对应的标签流的集合

$$\{T_{r_Q}, T_{v_1}, \ldots, T_{v_{n-1}},\};$$

输出:满足 Q 的元素结点 e_{o_Q}.

```
1.  while (¬empty(oQ)) do          //当输出结点及其后裔结点对应的标签流都不为空时,执行循环
2.      Vk=TwigNM_getnext(rQ);     //得到的 Vk 是有扩展的主结点
3.      if (Vk is not rQ) then cleanStack( S_parent(Vk) ,Vk);
4.      if (Vk is rQ or  S_parent(Vk)  is not empty) then
5.          clearStack( S_Vk ,Vk);
6.          moveToStack(Vk, S_Vk );    //将 Vk 压栈
7.          if (Vk is oQ) then
```

8. output(V_k); //输出满足小枝模式的元素结点
9. pop(S_{V_k}); //弹出 S_{V_k} 栈顶元素
10. proceed(T_{V_k}); //主结点 V_k 的流指针后移
11. **end while**

Function empty(V_l) //当 V_l 或其后裔结点的标签流为空时,返回 TRUE,否则返回 FALSE
1. **if** ($\exists V_j \in descendant(V_l)$: T_{V_j} is empty) **then** return TRUE;
2. **else** return FALSE;

Procedure moveToStack(V_m, S_{V_m})
1. push (E_{V_m}) to stack S_{V_m} // E_{V_m} 为结点 V 标签流中头结点元素

Procedure clearStack(S_{V_m} ,V_n)
1. **while** (S_{V_m} is not empty)$\wedge$($E.end < E_{V_n}.start$)) do //E 为 S_{V_m} 的栈顶元素
2. pop(S_V)

算法 1 是 TwigNM 算法的主算法.第 2 行返回一个有扩展的主结点 V_k,第 3 行和第 5 行是分别将 $S_{parent(V_k)}$ 和 S_{V_k} 中不是 E_{V_k} 的祖先的元素结点弹出.第 6 行是将在 $S_{parent(V_k)}$ 中有祖先元素结点的 E_{V_k} 压栈.第 8 行、第 9 行是将满足条件的元素结点输出.第 10 行 V_k 的流指针后移,算法中,proceed(V_k)函数是将 V_k 对应的标签流指针往后移动,指向下一个元素结点.执行循环直至将满足 Q 的所有 o_Q 类型的元素输出.

算法 2. *TwigNM_getnext*(V_t).

输入:V_t //V_t 为查询树 Q 中主结点;

输出:有扩展的主结点.

1. **if** (V_t is leaf node) **then** return (V_t); //若是叶子,则返回
2. **if** (V_t is not o_Q) **then** //考虑到输出结点不一定是叶子结点
3. V_d=*dchild*(V_t); //在主路径上 V_t 的孩子结点
4. g=*TwigNM_getnext*(V_d);
5. **if** (g!=V_d) **then** return g;
6. **while** ($\neg$empty(V_t)) do
7. **for** node V_i in *pchildren*(V_t) //除主路径以外其他路径上的孩子结点
8. g_i=*TwigNM_getnextp*(V_i); //有扩展的谓词结点 g_i
9. V_{max}=maxarg($V_j \in child(V_t)$); //maxarg 函数返回对应标签流中头元素 *start* 值最大的结点
10. V_{min}=minarg($V_j \in child(V_t)$); //minarg 函数返回对应标签流中头元素 *start* 值最小的结点
11. **while** ($E_{V_t}.end < E_{V_{max}}.start$) do proceed(T_{V_t}); //跳过使结点没有扩展的元素
12. **if** ($E_{V_t}.start < E_{V_{min}}.start$) **then** return ($V_t$);
13. **if** ($E_{V_d}.start < E_{V_t}.start$) **then** return V_d;
14. **for** ($\forall V_j \in pchild(V_t)$: E_{V_j} .start< $E_{V_t}.start$) **while** ($E_{V_j}.start < E_{V_t}.start$) do *proceed*(T_{V_j});
15. **end while**
16. return V_d //若 empty(V_t)返回 TRUE,则 V_t 将不可能再有扩展,算法返回有扩展的结点 V_d

算法 2 是要返回有扩展的主结点.第 1 行判断结点 V_t 是否是叶子结点,如果是叶子,则返回该结点.第 2~5 行判断 V_t 的孩子中是主结点的结点是否有扩展,若没有,则返回有扩展且是主结点的 V_t 的后裔结点.第 6~14 行是判断 V_t 是否有扩展,若没有,则返回 V_t 孩子中有扩展的主结点.其中,第 7 行、第 8 行计算 V_t 的孩子结点中有扩展的谓词结点.第 11 行、第 12 行是判断 V_t 对$\forall V_k \in child(V_t)$是否满足条件 $E_{V_t}.start < E_{V_k}.start < E_{V_t}.end$,若满足,则返回 V_t;否则,跳过那些使 V_t 不满足条件 T_{V_k} 中的元素,直到 $E_{V_t}.start < E_{V_k}.start$.第 13 行若 $V_d \in dchild(V_d)$满足条件 $E_{V_d}.start < E_{V_t}.start$,则返回该结点 V_d;否则,说明在 V_t 的孩子结点中存在一些使 V_t 不能有扩展的谓词结点,则

在第 14 行跳过那些使 V_t 不能满足条件的 V_j 类型元素,继续执行循环寻找有扩展的主结点.

算法 3. *TwigNM_getnextp*(V_p).

输入:V_p //V_p 为查询树 Q 中谓词结点;

输出:有扩展的谓词结点 V_p.

1. **if** (V_p is leaf node) **then** return V_p
2. **while** (¬empty(V_p)) do
3. **for** all node V_i in *child*(V_p),g_i=*TwigNM_getnextp*(V_i);
4. $V_{\max}$=maxarg($g_i \in$ *child*(V_p)); //maxarg 函数返回对应标签流中头元素 *start* 值最大的结点
5. $V_{\min}$=minarg($g_i \in$ *child*(V_p)); //minarg 函数返回对应标签流中头元素 *start* 值最小的结点
6. **while** (E_{V_p} .*end*< $E_{V_{\max}}$.*start*) do proceed(T_{V_p}); //跳过使 V_p 不能有扩展的元素结点
7. **if** (E_{V_p} .*end*< $E_{V_{\min}}$.*start*) **then** return V_p; //此时 V_p 是有扩展的
8. **for** ($\forall V_j \in$ *pchild*(V_p): E_{V_j} .*start*< E_{V_p} .*start*) **while** (E_{V_j} .*start*< E_{V_p} .*start*) do proceed(T_{V_j});
9. **end while**
10. return NULL //若 *empty*(V_p)返回 TRUE,则 V_p 将不会再有扩展,算法返回空

算法 3 是返回有扩展的谓词结点 V_p.第 1 行判断结点 V_p 是否是叶子结点,若是,则返回结点 V_p;若不是,则第 2~9 行找到有扩展的谓词结点.执行完第 3 行后,V_p 的所有孩子结点都会有扩展.第 6~8 行是判断 V_p 对$\forall V_k \in$ *child*(V_p)是否满足条件 E_{V_p} .*start*< E_{V_k} .*start*< E_{V_p} .*end*,如果满足,则返回 V_p;否则,在第 8 行中,跳过那些使 V_p 不满足条件的 V_j 类型元素.执行循环直至使得 V_p 有扩展.例如图 1 所示的 XML 文档和小枝模式查询树,算法 TwigNM 执行过程如下:

初始状态如图 2 所示,执行过程如图 4 所示,其中,箭头指向各标签流的头元素和结点栈的栈顶元素.在第 2~5 步,T_A 的流指针指向空,在第 5 步,T_C 的流指针指向空.

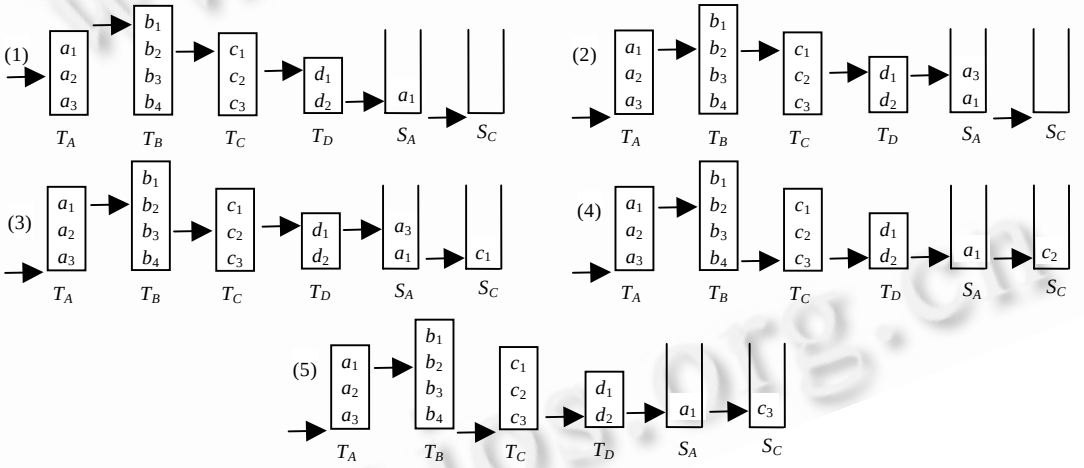


Fig.4 An example of TwigNM algorithm

图 4 TwigNM 算法实例

执行后,各标签流和栈的状态如图 4 所示.

(1) 执行 *TwigNM_getnext* 后,返回 a_1,a_1 压栈;(2) 执行 *TwigNM_getnext* 后,返回 a_3,a_3 压栈;(3) 执行 *TwigNM_getnext* 后,返回 c_1,输出 c_1;(4) 执行 *TwigNM_getnext* 后,返回 c_2,a_3 出栈,输出 c_2;(5) 执行 *TwigNM_getnext* 后,返回 c_3,输出 c_3.

最终结果返回 c_1,c_2,c_3.由此例可以发现,本算法不需要扫描谓词结点流中的所有结点.同时,当 T_{o_Q} 为空时就不再扫描其他标签流.

2.2 扩展算法TwigNME

TwigNM 算法在执行完第 1 步后,已经知道主结点 V_C 有扩展,但在执行第 2 步时,还要重新考虑 V_C 是否有扩展.针对这个问题,本文在 TwigNM 算法基础上提出了一种扩展算法 TwigNME.算法为每个结点设置了一个扩展指针来表示一个结点是否有扩展,这样可以避免重复地检测点是否有扩展.限于篇幅,这里只说明 TwigNME 算法的思想,不再详细描述该算法.由实验结果可以看出,其执行效率比 TwigNM 有较大的提高.

2.3 算法分析

2.3.1 算法的正确性分析

根据引理 1,算法 2 得到有扩展的主结点 V,其对应的头元素 E_V 一定在一个 Q_V 的匹配(E_V, E_{V_1},…, E_{V_n})之中,其中,$V_i(1\leq i\leq n)\in descendant(V)$.因此,栈 S_V 中的元素也一定都在某个满足 Q_V 的匹配中.

由算法 1 得知,因为对于一个 V 类型的元素 e_V,只有在 $S_{parent(V_k)}$ 中存在一个元素与满足祖先-后裔关系时,e_V 才会被压入栈中.所以,对于一个被算法输出的元素 e_{o_Q},一定存在一个匹配 $M=(e_{r_Q},...,e_{o_Q})$ 满足于主路径.M 中 $\forall e_{V_i}$ 一定会在一个满足的匹配中(V_i 是该主路径上的结点).由上面这些匹配很容易构造出一个新的匹配 $(e_{r_Q},...,e_{o_Q})$ 满足于 Q,所以由算法得到的元素一定是满足于 Q 的.

例如图 1 所示,a_1 在满足 Q 的匹配(a_1,b_1,c_1,d_1)之中,c_2 在满足 Q_C 的匹配(c_2,d_2)之中.又因为 a_1 与 c_2 满足祖先-后裔关系,则很容易构造一个新的匹配(a_1,b_1,c_2,d_2)满足 Q.

2.3.2 算法复杂度分析

TwigNM 算法为每个结点建立一个标签流,为每个主结点建立一个栈.因为每个主结点的栈的大小不会超过该结点的标签流的大小,所以其算法的空间复杂度在最坏情况下为 $O(|input|)$,其中,$|input|$表示输入标签流的长度总和.TwigNME 算法还建立了一个长度为 Q 的结点个数的扩展标志数组 *Extentive*,但对算法的空间复杂度影响很小,可以忽略.因此,这两种算法的空间复杂度相同.子算法 *TwigNME_getnextp*(V_p)只需要找到一个使 V_p 有扩展的实例$(V_1,V_2,\ldots,V_n):V_i\in descendand(V_p)(0<i\leq n)$,并且会快速地跳过不在此实例中的结点.所以,该算法受后裔中其他结点的标签流大小的影响很小.因此,在最坏情况下,*TwigNME_getnextp*(V_p)的算法复杂度为 $O(n)$,TwigNME 算法整体算法复杂度为 $O((d-1)n|T_m|)$,其中,d 为查询树 Q 的度减 1,$|T_m|$表示 Q 中所有主结点的标签流长度的总和.对于 TwigNM 算法,时间复杂度比 TwigNME 算法要大.但该算法主要受$|T_m|$的影响也最大,受$|input|$的影响不大.

3 实验结果和分析

3.1 实验设置

为了评价算法的性能,我们进行了大量的实验.所有的实验都在一台 Intel(R) Core(TM)2 Duo CPU E4400 @ 2.00 GHz,1.99GHZ,1G RAM,160G 硬盘的 PC 上运行,底层操作系统是 Windows XP.本实验使用 C++编程语言实现了 4 种算法:TwigStack,TwigList,TwigNM 和 TwigNME.之所以选择了 TwigStack 和 TwigList 算法与我们的算法进行对比,是因为 TwigStack 算法是一种典型的完整小枝模式匹配算法,很具代表性.而 2007 年提出的 TwigList 算法则是截至目前一种最新的小枝模式匹配算法[13],它比其他一些同类算法的效率要高.实验都是在两个著名的数据集:生成数据集 XMark[17]和真实的数据集 TreeBank[18]上进行的.其中,XMark 数据集是由 XMark 文档生成器生成的,大小为 113MB,有 1 666 315 个元素结点,最大深度为 12;TreeBank 数据集的大小为 82MB,有 2 437 666 个元素结点,最大深度为 36.XMark 数据集相对来说深度较小,结点类型也较少;TreeBank 数据集深度则很大,结点类型也很多.这两个数据集的规模都较大.在这两个数据集上进行了两组、每组 5 个小枝模式匹配查询(查询树)测试,这两组小枝模式查询树也和文献[5,13]中使用的查询树基本相同.实验中用到的小枝模式查询树及得到的结果结点个数见表 1.

Table 1 Twig queries used for the experimental evaluation

表 1 实验中所用的小枝模式查询树

Name	Dataset	QueryTrees	Result
XQ1	XMark	//item[//description//listitem//text//bold]//name	5 353
XQ2	XMark	//item[//description//text//bold]//mailbox//mail//date	11 585
XQ3	XMark	//site[//regions//parlist//text//keyword]//closed_auction//date	9 750
XQ4	XMark	//open_auctions[//reserve]//bidder[//time]//personref	59 486
XQ5	XMark	//site[//africa//shipping][//asia//mailbox][//europe//parlist]//text	105 114
TQ1	TreeBank	//S[//VP][//NP]//PP[//IN]//NP//VBN	5 642
TQ2	TreeBank	//S[//VP][//NP]//VP//PP[//IN]//NP//VBN	4 790
TQ3	TreeBank	//S//VP//PP[//NP//VBN]//IN	10 942
TQ4	TreeBank	//S//VP//PP[//NN][//NP[//CD]//VBN]//IN	3 281
TQ5	TreeBank	//EMPTY[//VP//PP//NNP][//S[//PP//JJ]//VBN]//PP//NP//_NONE_	2 278

3.2 实验结果与分析

图 5 给出了 4 种算法在表 1 所示的查询模式上的执行时间,它们都不包括输入标签流的时间,其输入标签流的时间是相同的.由图 5 可以看到,TwigNM 和 TiwgNME 在大多数情况下都比 TwigStack 和 TwigList 的执行效率高很多,而 TwigNME 在大多数情况下又比 TwigNM 的效率高.在 XMark 数据集中,TwigStack 所用的时间是 TiwgNM 的 3~6.9 倍,是 TiwgNME 的 3.5~94 倍;TwigList 是 TiwgNM 的 1.4~3 倍,是 TiwgNME 的 1.5~46 倍.在 TreeBank 中,TwigStack 所用的时间是 TiwgNM 的 2.0~8.8 倍,是 TiwgNME 的 2.0~10.1 倍;TwigList 是 TiwgNM 的 2.7~5.9 倍,是 TiwgNME 的 2.8~6.8 倍.由图 5 可以分析出,合并操作对 TwigStack 的执行效率有很大影响.另外,由于 TwigList 和 Twig2Stack 一样都是 Bottom-Up 算法,所以需要保存很多并不在最终结果中的元素结点,在只有祖先-后裔关系的情况下效率并不高.因此在图 5(a)中,对于 TQ3,TQ4 和 TQ5,TwigList 算法效率还不如 TwigStack.值得说明的是,这 4 种算法在实验中所得到的最后查询结果也完全相同,这也正好说明了我们的算法是正确的.

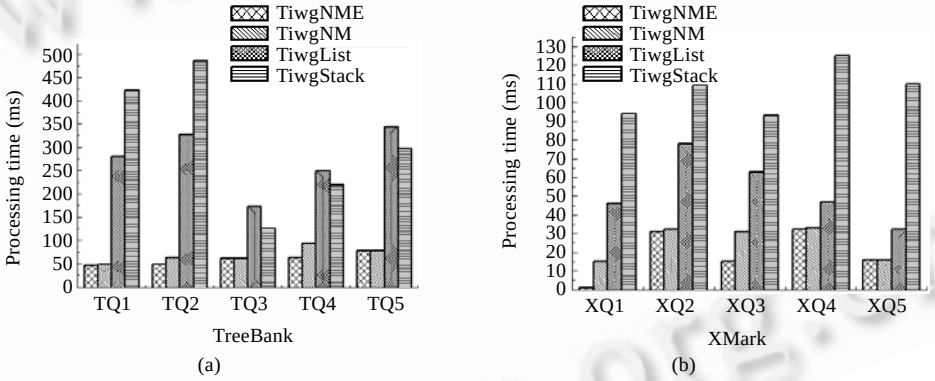


Fig.5 Processing time

图 5 算法的执行时间

4 结论和进一步的工作

本文针对有效地计算 XML 查询中的核心操作——小枝模式结构连接的有效处理问题进行了研究.尤其是考虑到通常 XML 的查询表达式中只有少数几个结点是最终的输出结点这一特点,提出了 TiwgNM 算法及其扩展算法 TiwgNME.实验结果表明,该算法优于以前的算法,尤其是对查询中只有祖先-后裔关系的表达式.实验虽然只考虑了只有少数输出结点的情况,但对于有多个输出结点的情况,可由输出结点构成的新的小枝,则可以采用 TwigList 算法中的方法,用队列来保存这些结果,避免合并操作.今后还有不少工作要做,本文所研究的只是简单的小枝模式查询和 XPath 的一个片段,对于有其他轴的更为复杂的情况还有待于进一步研究.此外,本文没有用到索引,若加上索引将会使查询效率有更大的提高,这一点可以参考文献[6,7].

References:

[1] Zhang C, Naughton J, De Witt D, Luo Q, Lohman G. On supporting containment queries in relational database management systems. In: Timos S, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2001. 425−436.

[2] Li QZ, Moon B. Indexing and querying XML data for regular path expressions. In: Apers PMG, Atzeni P, Ceri S, Paraboschi S, Ramamohanarao K, Snodgrass RT, eds. Proc. of the 27th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2001. 361−370.

[3] AI-Khalifa S, Jagadish HV, Koudas N, Patel JM, Srivastava D, Wu Y. Structural joins: A primitive for efficient XML query pattern matching. In: Agrawal R, Dittrich K, Ngu AHH, eds. Proc. of the 18th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2002. 141−152.

[4] Wang J, Meng XF, Wang S. Structural join of XML based on range partitioning. Journal of Software, 2004,15(5):720−729 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/720.htm

[5] Bruno N, Koudas N, Srivastava D. Holistic twig joins: Optimal XML pattern matching. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 310−321.

[6] Jiang H, Lu H, Wang W. XR-Tree: Indexing XML data for efficient structural joins. In: Dayal U, Ramamritham K, eds. Proc. of the 19th Int'l Conf. on Data Engineering (ICDE). Bangalore: IEEE Computer Society, 2003. 253−264.

[7] Chen T, Lu J, Ling TW. On boosting holism in XML twig pattern matching. In: Ozcan F, ed. Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: ACM Press, 2005. 455−466.

[8] Lu J, Chen T, Ling TW. Effcient processing of XML twig patterns with parent child edges: A look-ahead approach. In: Proc. of the ACM Conf. on Information and Knowledge Management (CIKM). Washington: ACM Press, 2004. 533−542.

[9] Lu J, Ling TW, Chan CY, Chen T. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In: Bohm KK, Jensen CS, Haas LM, Kersten ML, Larson P, Ooi BC, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB). Trondheim: ACM Press, 2005. 193−204.

[10] Choi B, Mahoui M, Wood D. On the optimality of holistic algorithms for twig queries. In: Marik V, *et al*., eds. Proc. of the Database and Expext Systems Application (DEXA). LNCS 2736, Berlin: Springer-Verlag, 2003. 28−37.

[11] Aghili S, Li, HG, Agrawal D, Abbadi AE. Twix: Twig structure and content matching of selective queries using binary labeling. In: Proc. of the first Int'l Conf. on Scalable Information Systems (INFOSCALE). Hong Kong: ACM Press, 2006. 411−420.

[12] Chen S, Li HG, Tatemura J, Hsiung WP, Agrawal D, Candan KS. Twig2stack: Bottom-Up processing of generalized-tree pattern queries over XML documents. In: Dayal U, Whang KY, Lomet DB, *et al*., eds. Proc. of the 32nd Int'l Conf. on Very Large Data Bases (VLDB). Seoul: ACM Press, 2006. 283−294.

[13] Liu Q, Jeffrey XY, Ding BL. TwigList: Make twig pattern matching fast. In: Proc. of the 12th Int'l Conf. on Database Systems for Advances Applications (DASFAA). Bangkok: 2007. 850−862. http://www.se.cuhk.edu.hk/~lqin/files/TwigList.pdf

[14] Wirth N. Type extensions. ACM Trans. on Programming Languages and Systems, 1988,10(2):204−214.

[15] Online Computer Library Center. Dewey decimal classification. http://www.oclc.org/dewey/

[16] Wang W, Jiang HF, Lu HJ, Jeffrey XY. PBiTree coding and efficient processing of containment joins. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. Los Alamitos: IEEE Press, 2003. 391−402.

[17] [17] 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

[18] University of Washington XML Repository. http://www.cs.washington.edu/research/xmldatasets/

附中文参考文献:

[4] 王静,孟小峰,王珊.基于区域划分得 XML 结构连接.软件学报,2004,15(5):720−729. http://www.jos.org.cn/1000-9825/15/720.htm

陶世群(1946－),男,安徽寿县人,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.

富丽贞(1982－),女,硕士生,主要研究领域为数据库技术,XML 数据管理.