SCALER+：基于序列匹配的高效 XML 分支查询求解算法

刘乐　冯建华
(清华大学计算机科学与技术系　北京　100084)

SCALER+：A Efficient Algorithm for XML Twig Query Evaluation Based Sequence Matching

Liu Le and Feng Jianhua
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

Abstract XML query evaluation is a hot topic currently. The algorithm named SCALER[1] based sequence matching for XML twig query evaluation is proved very efficient, but it has also some drawbacks. In this paper two problems are solved, the first is definitely supporting wildcard * and descendant axis //, the second is about brother nodes' order in pattern trees. The improved algorithm is name SCALER+, it's efficient as SCALER.

Key Words Sequence Matching, Twig Query Evaluation, UDFTS

摘　要　XML 查询求解是目前的一个热点问题。基于序列匹配的 XML 分支查询求解算法 SCALER[1]被证明具有非常优异的性能，但它也有一些不足。本文解决了 SCALER 的两个问题，一是明确地对通配符*和后代轴//的支持，二是模式树中兄弟结点的顺序问题。改进后的算法命名为 SCALER+，它具有与 SCALER 一样优异的性能。

关键词　序列匹配，　分支查询求解，　UDFTS
中图法分类号　TP311.13

1　引　言

近年来，XML 迅速成为 Internet 上数据表示和数据交换的事实标准，海量的数据以 XML 文档的形式出现。如何对这些 XML 文档进行高效查询，是当前数据库研究领域的一大热点问题。

目前 XML 查询算法主要分为三类，即导航遍历算法，分支连接算法（包括结构连接算法），序列匹配算法。对于 XML 分支查询，导航遍历算法主要利用 DataGuide[2]，1-index[3]，A(k)[4]，F&B[5]等索引机制来裁剪掉多余的搜索空间。但这些算法都存在效率不高或索引有时候太大的缺点。而结构连接算法[6][7]则是将复杂的分支路径分解成多个单独的结构连接，然后不断地利用连接结果再进行结构连接，每次只处理一对结点，直到所有的路径和结点都处理完毕。这样的处理效率当然不会高，当然分支连接是目前整体解决 XML 分支查询最有效的方法。

序列匹配是求解 XML 分支查询问题的新方法，这方面比较有名的工作有 PRIX[8]和 ViST[9]。但 PRIX 没有统一对非叶结点和叶结点的处理，对于一次完整的分支路径匹配，需要单独处理叶结点的匹配问题。而 ViST 在处理查询时可能会产生错误的序列匹配结果。所以在 ViST 中需要检测并消除这种错误结果。为了解决 PRIX 和 ViST 的缺陷，高效且正确的 SCALER 算法被提出。

SCALER 算法中，采用 UDFTS（Unique Depth-First Traversal Sequence，惟一深度优先遍历序列）来描述 XML 文档树。由于 UDFTS 与 XML 文档树是一一对应的，这就可以对文档树的结构和内容结点进行统一

的处理。SCALER 把带有分支的 XML 查询作为一个整体来处理，利用 UDFTS 通过一遍扫描就解决了分支模式与 XML 文档集的匹配问题，避免了可能存在错误匹配的情况。

但 SCALER 算法也存在两个问题，一是虽然提出了解决方案但却没有明确地支持通配符*和后代轴//，二是没有解决模式树中兄弟结点的顺序问题。本文将对 SCALER 算法进行改进，解决以上两个问题。改进后的算法命名为 SCALER+。

本文组织结构如下：第 2 节简要介绍 SCALER；第 3 节详细描述 SCALER+；第 4 节是 SCALER+的效率分析；第 5 节是总结。

2 SCALER 分析

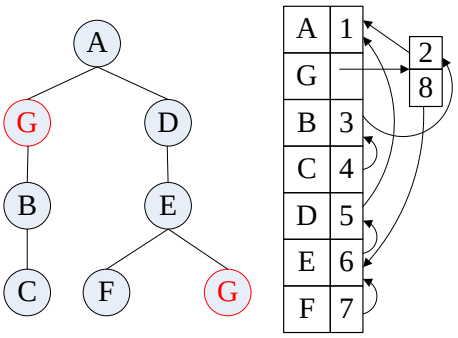
为了高效且正确地执行 XML 分支查询求解，文献[1]提出了 SCALER 算法。

SCALER 算法分为两步，一是利用 DFT（Depth-First Traversal，深度优先遍历）方法序列化 XML 文档和分支查询模式树，生成 UDFTS，二是基于 UDFTS 进行序列匹配。

DFTS（Depth-First Traversal Sequence, 深度优先遍历序列）虽然能够保持原 XML 文档的一些结构信息，但并不能保持所有的结构信息。比如一个结点 P 在 DFTS 中出现在结点 R 的左边，但 P 到底是 R 的左兄弟还是 R 的父结点或二者没有任何关系是无法确定的。为了惟一地表示 XML 文档，在 DFT 方法的基础上引入了 UDFTS。

给定一个 XML 文档，假设它的 DFTS= $L_1, L_2, \ldots, L_n$, 其中 L_i 表示结点的标签名，则对应的 UDFTS=$L_1(P_1), L_2(P_2),\ldots, L_n(P_n)$。其中 L_i 同 DFTS 中含义一样，P_i 是一个整数，表示结点 L_i 的父结点在 UDFTS 中的位置编号。在 UDFTS 中结点 L_i 从 1 到 n 依次编号，比如 L_1 的位置编号为 1, $L_4(3)$表示结点 L_4 的父结点的位置编号为 3。规定 XML 文档的根结点的父结点的位置编号 $P_1=0$。如图 1 所示。

根据 DFT 方法很容易建立起 XML 文档的 UDFTS。在建立 UDFTS 的过程中，同时会构造一个一次扫描索引，简称 OSI（Once-Scan Index），它将 XML 文档结点的标签名和其在 UDFTS 中的位置编号关联起来，在序列匹配中用来快速找到具有某个标签名的所有结点。如图 1 所示。



UDFTS: A(0)G(1)B(2)C(3)D(1)E(5)F(6)G(6)
位置编号:1 2 3 4 5 6 7 8

图 1 UDFTS 和 OSI

建好 UDFTS 之后，接下来就进行序列匹配了。序列匹配的目标就是在 $UDFTS_{Doc}$（XML 文档 Doc 对应的序列）中找到至少一个有效子序列 S，它与 $UDFTS_Q$（查询模式树 Q 对应的序列）是一样的。基本做法是对于 $UDFTS_Q$ 中的每一个结点 qNextNode，通过检索 OSI 找到与其标签名相同的所有文档结点，并检查这些结点的父结点是否与 qNextNode 的父结点相同，丢弃不相同的，把相同的加到结果序列中。重复上述过程，直到 qNextNode 为 NULL，则算法结束，输出结果序列即可。

在文献[1]中，已经提出了支持通配符*和后代轴//的解决方法，不过没有具体实施到算法当中。同时，SCALER 算法没有解决模式树中兄弟结点的顺序问题。SCALER 把查询模式树也看成是完全有序的 XML 文档，而实际上模式树中兄弟结点之间是无序的。上述的两个问题将在第 3 节中解决。

3 SCALER+:改进的 SCALER

SCALER 算法已经含有了支持通配符*和后代轴//的分支查询，本节采用这些方法，并且具体应用到 SCALER+算法中。

3.1 UDFTS 的改进

为了惟一地表示 XML 文档中的结点，在 UDFTS 中为每个结点加上自身位置编

号，用一个整数来表示。并且为了能够表示祖先－后裔结点的情形，把 XML 树的惟一深度优先遍历序列表示为 UDFTS =L_1（N_1,#P_1），L_2（N_2,#P_2），…，L_n（N_n,#P_n），其中 L_i 表示结点的标签名，N_i 表示此结点在 UDFTS 中的位置编号，符号#既可以表示空也可以表示//，表示空时就说明 P_n 是此结点的父结点在 UDFTS 中的位置编号，表示//时就说明 P_n 是此结点的祖先结点在 UDFTS 中的位置编号。

从上面 UDFTS 的定义可知，L_i 的位置编号就是 i，即 N_i=i，这样看起来好像 N_i 是多余的。其实不然。因为在一个 XML 文档树中可能有多个结点的标签名是一样的，仅凭标签名无法判断到底是指哪一个结点，但位置编号（即 N_i）却可以惟一地确定一个结点。引进 N_i 对于后面将要讨论的支持兄弟结点无序的模式树序列匹配意义重大。但在不引起歧义的情况下可以省略 N_i。如图 2 所示。

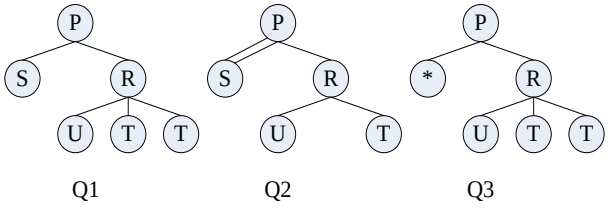


图 2 几个模式树实例

$UDFTS_{Q1}$=P(1,0)S(2,1)R(3,1)U(4,3)T(5,3)T(6,3)

$UDFTS_{Q2}$=P(1,0)S(2,//1)R(3,1)U(4,3)T(5,3)

$UDFTS_{Q3}$=P(1,0)*(2,1)R(3,1)U(4,3)T(5,3)T(6,3)

在 Q1 中有两个 T 结点，仅从标签来看是无法分辨这两个 T 结点的，但加上位置编号后就可以轻易地分辨这两个结点了。在 Q2 中 P//S 是祖先－后裔边，于是 S 在 UDFTS 中得表示成 S(2,//1)的形式。在 Q3 中出现了通配符*，在生成 UDFTS 时把*当作普通的结点对待，只是在序列匹配的时候要把*的父结点 P 的所有子结点都当作*的候选结点。

3.2 SCALER+

XML 文档树中的兄弟结点是有序的，但查询模式树中的兄弟结点却是无序的。SCALER 中把查询模式树也看成是有序的 XML 文档树，这大大限制了 SCALER 的应用。如图 3 所示。

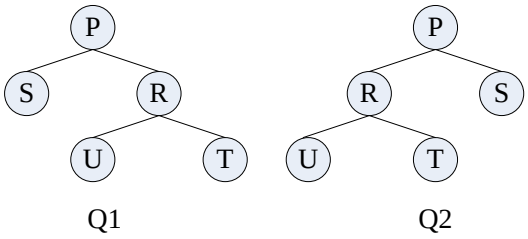


图 3 无序的模式树

图 3 中的两个模式树，其 UDFTS 显然是不同的：

$UDFTS_{Q1}$=P(1,0)S(2,1)R(3,1)U(4,3)T(5,3)

$UDFTS_{Q2}$=P(1,0)R(2,1)U(3,2)T(4,2)S(5,1)

虽然 Q1,Q2 的 UDFTS 不同，但应用于查询时产生的结果却应是完全一致的。可以看到，Q1 和 Q2 的区别就在于 P 的两个子结点 S、R 的顺序不同，在 Q1（Q2）中只要把 S、R 两个结点的位置互换一下就变成了 Q2（Q1）。更确切地讲，是将分别以 S、R 为根结点的两棵子树互换位置。通过将兄弟结点的位置互换而得到的新的树与原来的树是相似的。下面给出相似树的一般定义。

定义 1 相似树：

假设 T1 和 T2 是两棵 XML 树，如果它们满足以下条件则称 T1 和 T2 互为相似树：对于 T1 中的任意一条 P-C(或 A-D)边<u , v>，在 T2 中也存在一条相应的边<u , v>；并且，对于 T2 中的任意一条 P-C(或 A-D)边<r , s>，在 T1 中也存在一条相应的边<r , s>。

UDFTS 与 XML 树是一一对应的，是等价的，既然 XML 树存在相似的概念，那么在 UDFTS 中也存在同样的概念。

定义 2 相似序列：

假设 S1 和 S2 是两个 UDFTS，其中 S1＝L_1（N_1,#P_1），L_2（N_2,#P_2），…，L_n（N_n,#P_n），S2＝l_1（n_1,#p_1），l_2（n_2,#p_2），…，l_n（n_n,#p_n），如果它们满足以下条件则称 S1 和 S2 互为相似序列：对于 S2 中的每个 l_i（n_i,#p_i），在 S1 中都存在 L_j（N_j,#P_j），使得 l_i=L_j，l_{pi}=L_{Pj} 而且两个#要么都表示空要么都表示//；并且对于 S1 中的每个 L_i（N_i,#P_i）,在 S2 中都存在

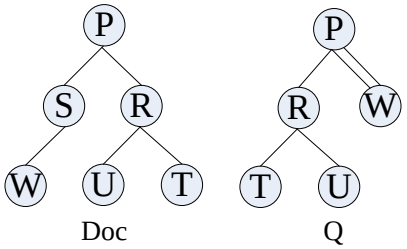
l_j（n_j,#p_j），使得 $L_i=l_j$，$L_{pi}=l_{pj}$ 而且两个#要么都表示空要么都表示//。

在序列匹配的过程中， XML 文档树中只有父子边（P－C 边），没有祖先－后裔边（A－D 边），于是它对应的序列 $UDFTS_{Doc}$ 中只存在 L（P）而不存在 L（//P）。只有在模式树中才有 A－D 边，对应的序列 $UDFTS_Q$ 中存在 L（//P），并且都是 $UDFTS_Q$ 作为子序列去匹配 $UDFTS_{Doc}$。为了简化问题的描述，以下的讨论都基于这种简单情况给出。

定义 3 相似子序列：

假设 S_1 是一个 XML 文档的 UDFTS，$S_1=L_1$（P_1），L_2（P_2），…，L_n（P_n），S_2 是一个模式树的 UDFTS，$S_2=l_1(\#p_1), l_2(\#p_2), \ldots, l_m(\#p_m))$ $(m\le n)$。如果满足以下条件，就称 S_2 是 S_1 的相似子序列：对于 S_2 中的每个 $l_i(\#p_i)$ $(0<i\le m)$，如果 $l_i(\#p_i)$表示的是 $l_i(p_i)$，则在 S_1 中存在 L_j（P_j）（$0<j\le n$），使得 $l_i=L_j$ 而且 $l_{pi}=L_{Pj}$；如果 $l_i(\#p_i)$表示的是 $l_i(//p_i)$，则在 S_1 中存在 L_j（P_j），L_k（P_k）（$0<j,k\le n$），使得 $l_i=L_j$ 而且 $l_{pi}= L_{Pk}$ ，其中 L_k（P_k）是 L_j（P_j）的祖先结点。

此处的相似子序列的定义与［1］中的子序列相比较，实际上去掉了对兄弟结点的有序性的要求，并且加上了祖先－后裔边的情形。不过当在 S2 中存在 A－D 边<u,v>时，这里并不要求 S1 中也存在同样的 A－D 边<u,v>，而只是要求从结点 u 到结点 v 存在一条祖先－后裔路径（即 u 是 v 的祖先），由于 S1 中不存在 A－D 边。当把 S2 作为模式树抽取 S1 中的数据的时候，产生的是 S2 的相似序列，如图 4 所示。



$UDFTS_{Doc}$=P(1,0)S(2,1)W(3,2)R(4,1)U(5,4)T(6,4)
$UDFTS_Q$=P(1,0)R(2,1)T(3,2)U(4,2)W(5,//1)

图 4 相似子序列

图 4 中的 $UDFTS_Q$ 是 $UDFTS_{Doc}$ 的相似子序列。用 Q 作为模式树抽取 Doc 中的数据的时候，产生的就是 Q 本身。如果仍然使用 Doc 中的位置编号的话，则 UDFTS=P(1,0) R(4,1)T(6,4)U(5,4)W(3,//1)。按照位置编号的大小重排结点，则得到 UDFTS=P(1,0) W(3,//1)R(4,1)U(5,4)T(6,4)，此子序列保持着与 $UDFTS_{Doc}$ 相同的结点顺序，重新分配位置编号则得到 UDFTS=P(1,0) W(2,//1)R(3,1) U(4,4)T(5,4)，这正是我们所要的查询结果。

下面给出有效局部项和有效相似子序列的概念。

定义 4 有效局部项：

假设 S 是一个 UDFTS 且 S=L_1（P_1），L_2（P_2），…，L_n（P_n），并且给定一个候选结点 L（#P）。如果在 S 中存在 L_i（$1\le i\le n$），使得 i=P，那么就称 L（#P）为有效局部项。

定义 4.4 实际上是说结点 L（P）的父亲结点在序列 S 中，或结点 L（//P）的祖先结点在序列 S 中。

定义 5 有效相似子序列：

假设一个 XML 文档 Doc 对应的序列为 $UDFTS_{Doc}$，而查询序列 S=L_1（#P_1），L_2（#P_2），…，L_n（#P_n）是 $UDFTS_{Doc}$ 的相似子序列。如果对序列 S 中的每个结点 L_i（#P_i）（$1<i\le n$）来说，它都是子序列 L_1（#P_1），L_2（#P_2），…，L_{i-1}（#P_{i-1}）的有效局部项，则称序列 S 是 $UDFTS_{Doc}$ 的一个有效子序列。

实际上，定义 5 给出了判断 UDFTS 中每个结点是否连通的方法。

引理 1 给定一个分支查询 Q 和一个 XML 文档 Doc，如果 Doc 是满足 Q 的文档，则至少存在一个 $UDFTS_{Doc}$ 的相似子序列 S 满足以下两个条件：

（1） S 是 $UDFTS_{Doc}$ 的有效子序列；

（2） S 与 $UDFTS_Q$ 是相似序列。

根据定义 2 和定义 5 容易证明引理 1。根据引理 1,很容易给出改进的基于序列匹配的 XML 分支查询求解算法 SCALER+，算法的详细步骤描述如下。

算法 1　SCALER+（$UDFTS_{Doc}$，$UDFTS_Q$，OSI）

输入：XML 文档 Doc 对应的 $UDFTS_{Doc}$；

XML 分支查询 Q 对应的 $UDFTS_Q$；

一次扫描索引 OSI。

输出：与查询 Q 匹配的 XML 文档 Doc 的所有有效相似子序列。

```
Begin
1.  qNextNode=FirstNodeOf（UDFTS_Q）;
2.  resultSequenceSet=∅;
3.  while（qNextNode!=NULL）
4.      CurrentMatchedNodeSet=FindMatchedNode（UDFTS_Doc，qNextNode，OSI）;
       /*利用 OSI 在 UDFTS_Doc 中找到所有与当前结点 qNextNode 匹配的
                                                    所有候选结点*/
       /*如何处理通配符*与//也由过程 FindMatchedNode 来完成*/
5.    if qNextNode==FirstNodeOf（UDFTS_Q）then
6.      for each appendItem in CurrentMatchedNodeSet do
7.        AddSequence（resultSequenceSet，appendItem）;
8.      end for
9.    else
10.     tempSequenceSet=∅;
11.     for each CurrentSequence in resultSequenceSet do
12.       for each appendItem in CurrentMatchedNodeSet do
13.         if appendItem is valid for CurrentSequence then
   /*检查 appendItem 是否是有效局部项；检查 appendItem（候选结点）在
       CurrentSequence 中的父结点是否与 qNextNode（当前结点）在 UDFTS_Q 中的
       父结点一致，或是 appendItem 在 CurrentSequence 中的某一个祖先结点是否
       与 qNextNode 在 UDFTS_Q 中的祖先结点一致。*/
   /* 与 SCALER 相比，SCALER+不再强调结构一致性，而是强调结构相似性，从而
       解决了兄弟结点的顺序问题。*/
   /* 如果是祖先－后裔结点的情形，则需要将候选结点 L(N,P)中的 P 用当前结点
       l_i(n_i,//p_i)中的// p_i 代替。*/
14.           AddSequence（tempSequenceSet，CurrentSequence + appendItem）;
15.         end if
16.       end for
17.       RemoveSequence（resultSequenceSet，CurrentSequence）;
18.     end for
19.     for each Sequence in tempSequenceSet do
20.       AddSequence（resultSequenceSet，Sequence）;
21.     end for
22.   end if
23.   if resultSequenceSet==∅ then
24.     return ∅;
25.   else
26.     qNextNode=NextNodeOf（UDFTS_Q，qNextNode）;
27.   end if
```

28. end while
29. for each Sequence in resultSequenceSet
30. SortBySequenceNumber(Sequence);
/*按照位置编号排序，使得输出序列的结点顺序与 $UDFTS_{Doc}$ 一致。*/
31. end for
32. return resultSequenceSet；
End

从算法的执行步骤上来看，SCALER+与 SCALER 有两个主要的不同之处。一是第 13 行对候选结点的检查。SCALER 检查候选结点的局部有效性和结构一致性，而 SCALER+则检查候选结点的局部有效性和相似性。并且局部有效性的具体内容也是不同的。二是第 30 行对临时结果序列按照结点的位置编号进行排序，这在 SCALER 中是没有的。

下面利用图 4 的 XML 文档 Doc 和分支查询 Q 具体解释一下 SCALER+算法的执行过程。其中 $UDFTS_{Doc}$=P(1,0)S(2,1)W(3,2)R(4,1)U(5,4)T(6,4),$UDFTS_{Q}$=P(1,0)R(2,1)T(3,2)U(4,2)W(5,//1)。表 1 详细地描述了 SCALER+算法中序列的匹配过程以及每一步的执行结果。

表 1 基于图 4 的 SCALER+算法的执行过程

扩展过程	当前结点	Doc 的 resultSequenceSet
Step1	P(1,0)	{P(1,0)}
Step2	R(2,1)	{P(1,0)R(4,1)}
Step3	T(3,2)	{P(1,0)R(4,1)T(6,4)}
Step4	U(4,2)	{P(1,0)R(4,1)T(6,4)U(5,4)}
Step5	W(5,//1)	{P(1,0)R(4,1)T(6,4)U(5,4)W(3,//1)}
排序结果	NULL	{P(1,0)W(3,//1)R(4,1)U(5,4)T(6,4)}

将得到的结果序列 S=P(1,0)W(3,//1)R(4,1)U(5,4)T(6,4)按照 1 到 n 分配位置编号得到标准的 UDFTS=P(1,0)W(2,//1)R(3,1)U(4,3) T(5,3)。

4 SCALER+的效率分析

比较 SCALER 算法和 SCALER+算法的执行过程可以看到，SCALER+仅仅多了第 30 行的排序而已，其它语句的时间复杂度都基本一致。一般来讲，最后参与排序的结果序列长度不大（与模式树序列相同），个数也不会太多，而且它们是局部有序的，所以采用适当的排序方法可以获得相对于前面的步骤来说很小的时间复杂度。因此，SCALER+算法的性能与 SCALER 算法相差无几。而 SCALER 算法已经被实验证明具有比 PRIX 和 ViST 算法更高的效率（参见文献[1]），因此 SCALER+同样会具有比 PRIX 和 ViST 更高的效率。

5 结论及进一步的工作

随着 XML 的广泛应用，XML 查询求解成为目前一个非常重要的热点问题。为了高效且正确地处理 XML 分支查询求解的问题，本文在 SCALER 算法的基础上提出了 SCALER+算法。SCALER+是对 SCALER 的改进，是一种基于序列匹配的高效 XML 分支查询求解算法。在不牺牲算法性能的前提下，SCALER+明确地实现了对通配符*和后代轴//的支持，并且解决了模式树中兄弟结点的无序问题。这两个问题的解决，使得这个算法基本上能够高效处理所有的 XML 分支查询求解问题。

进一步的工作，将继续改进 SCALER+算法，使它具有更高的性能。并与其他优秀分支查询算法（如 Twig Join）进行比较，探讨效率更高处理能力更强的算法。

参考文献

［1］ 冯建华. 纯 XML 数据库的查询求解关键问题研究：[博士论文]. 北京：清华大学计算机科学与技术系，2006

[2] R. Goldman, J. Widom. Approximate DataGuides. In Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats. Jerusalem, Israel, 1999. 436~445

[3] T. Milo, D. Suciu. Index Structures for Path Expressions. Lecture Notes in Computer Science. 1999, 1540: 277~295

[4] R. Kaushik, P. Sheony, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Efficient Indexing of Paths in Graph Structured Data. In Proceedings of the 18th International Conference on Data Engineering. San Jose, California, USA, 2002. 129~140

[5] S. Abiteboul, P. Buneman, and D. Suciu. Data on the Web: From Relations to Semistructured Data and XML. San Francisco: Morgan Kaufmann Publishers Inc. , 1999

[6] S. AI-Khalifa, H.V. Jagadish, et al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In Proceedings of the 18th International Conference on Data Engineering. San Jose, California, USA, 2002. 141~152

[7] C. Zhang, J. Naaghton, D. DeWitt, et al. On Supporting Containment Queries in Relational Database Management Systems. ACM SIGMOD Record. 2001, 30(2): 425~436

[8] Praveen Rao, Bongki Moon. PRIX: Indexing and Querying XML Using Prüfer Sequences. In Proceedings of the 20th International Conference on Data Engineering. Boston, Massachusetts, USA, 2004. 288~300

[9] H. Wang, S. Park, W. Fan, P. S. Yu. ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. In: Alon Y. Halevy, Zachary G. Ives, AnHai Doan, eds. Proceedings of the ACM SIGMOD International Conference on Management of Data. San Diego, California, USA, 2003. 110~121

作者信息

论文题目：SCALER+：基于序列匹配的高效 XML 分支查询求解算法
作者姓名：刘乐
所属单位：清华大学计算机系
电子邮件：windeagle.liu@gmail.com
通信地址：北京清华大学计算机系软件所
电　　话：010－62789150
传　　真：010－62771138

作者姓名：冯建华
所属单位：清华大学计算机系
电子邮件：fengjh@mail.tsinghua.edu.cn
通信地址：北京清华大学计算机系软件所
电　　话：010－62789150
传　　真：010－62771138