XML 数据库结构连接算法之分析*)

门爱华[1] 周立柱[2] 张亚鹏[1]
(赤峰学院计算机科学与技术系 内蒙古 024000)[1] (清华大学计算机科学与技术系 北京 100084)[2]

摘 要 结构连接是 XML 查询处理的核心操作,受到了计算机研究界的高度关注。高效的算法是高效查询处理的关键,目前已经提出许多结构连接的算法。本文介绍了几种典型的算法,并分析了这几种算法的优缺点。

关键词 XML 查询处理,结构连接,编码,算法,索引

Analysis of Algorithms for XML Database Structural Join

MEN Ai-Hua[1] ZHOU Li-Zhu[2] ZHANG Ya-Peng[1]
(Department of Computer Science and Technology, Chifeng College, NeiMongolia 024000)[1]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)[2]

Abstract Structural join is the core operation in XML query processing, and catches the research community's attention. Efficient algorithm is the key of efficient query processing. There have been a number of algorithms proposed for structural join. The article introduces several algorithms proposed for XML structural join, providing careful description and analysis for each of them, and compares their functions and their strongpoint and shortcoming.

Keywords XML query processing, Structural join, Code, Algorithm, Index

1 引言

XML 文档采用了树型的数据模型,针对这种树状数据的查询,学者们已经提出了多种 XML 查询语言,例如 XPath, XQuery 等。这些查询语言,尽管各有特点,但它们都将路径表达式作为重要的组成部分。为了高效地计算路径表达式,人们提出了一种查询处理策略。其主要思想是将一个复杂的查询模式分解成为若干个二元基本结构关系的集合,首先计算二元基本结构关系,然后将基本的匹配结果组合起来。在这种处理策略下,基本结构关系(包括父子关系和祖先-后代关系)的计算成为查询处理的关键操作,这种操作被称为结构连接(或包含连接)。对结构连接算法的研究近年来备受重视,提出了一系列有效的结构连接算法。这些算法都是基于归并的思想,充分利用 XML 数据的结构特点来减少连接的扫描代价;有些算法在归并的基础上,利用索引来进一步减少 XML 查询的中间谓词的连接。

2 结构连接

目前已经提出的处理结构连接的算法主要包括:

(1)包含关系的结构连接;

(2)文档位置关系结构连接。

下面以 Zhang 编码为例,每个结点用四元组(docID, begin, end, level)来表示,Alist、Dlist 分别表示祖先(或双亲)元素列表、后裔(或孩子)元素列表,并将 Alist、Dlist 分别简记为 A、D;每个列表都按{docID, begin}有序存储或索引聚集存储。那么,包含关系结构连接的条件分别是:

A. docID=D. docID and A. begin<D. begin and D. begin <A. end (1)

A. docID=D. docID and A. begin<D. begin and D. begin <A. end and A. level=D. level-1 (2)

3 关系数据库的连接算法

在关系数据库系统中,比较高效的连接算法有:排序归并连接算法(Sort MergeJoin,简记为 SMJ)、索引嵌套循环连接算法(Index Nested Loop Join,简记为 INLJ)和 Hash 连接算法等。目前的商用关系数据库系统的 SMJ 算法一般都是先进行连接条件中的等值连接操作,然后对连接结构进行连接条件中的非等值连接操作。例如,对于连接条件(1),第一步是按条件"A. docID=D. docID"进行等值连接操作,结果是将关系表 Alist 和 Dlist 中的元组按 docID 列的取值分别划分为 m 个对应的元组子集 A1, A2, …, Am、D1, D2, …, Dm,其中元组子集 Ai 与 Di 中的元组有相同的 docID 取值,且关系表 Alist 或 Dlist 中没有对应 docID 取值的元组已经去掉;第二步就是对第一步等值连接的结果按条件"A. begin<D. begin and D. begin<a. end"进行非等值连接操作。因此,SMJ 算法对于包含连接的计算效率并不是很高[1]。INLJ 算法的基本思想是:对于外表中的每一个元组 ai,首先通过索引在内表中搜索第一条可能与元组 ai 进行连接的元组 d_{start},称为开始码。然后从元组 d_{start} 开始在内表中进行顺序扫描,对满足结束码条件的元组 dj 再判断是否满足连接条件。若满足,则产生连接结果元组 ai. dj,直到第一个不满足结束条件的元组 d_{stop} 为止。

4 直接归并连接算法

4.1 多谓词归并连接算法

文[1]对在关系数据库系统中如何有效地实现包含连接操

*)内蒙古自治区高等学校科学研究项目(编号:NJ05008);赤峰学院科学研究基金资助项目。门爱华 副教授,主要研究领域为数据库理论与技术;周立柱 博士生导师,主要研究领域为分布式数据库、WEB 服务;张亚鹏 讲师,硕士。

作进行了研究，提出了一种有效地计算包含关系结构连接的算法 MPMGJN(Multi-Predicate Merge Join)，并将 MPMGJN 算法与关系引擎中的 SMJ 算法、INLJ 算法的性能进行了比较分析。

算法基本思想是：设参加连接的两个关系表 listA(祖先列表)和 listD(后裔列表)，则对外表 listA 中的第一个元组 a1，首先在内表 list2 中顺序搜索到可能与元组 a1 连接的第一个元组(即 begin＞a1. begin 的第一个元组)，称为扫描始点，然后在内表 Dlist 中从扫描始点开始顺序扫描，对满足 begin＜a1. end 条件的所有元组 dj，再判断是否满足连接条件。若满足，则产生连接结果元组 a1. dj；继续对外表 Alist 中的第二个元组 a2 重复上面的步骤，直到外表 listA 或内表 listD 中元组连接完毕。具体算法如下：

```
procedure containment merge(listA,listD)
begin
1. set cursor1 at beginning of listA;
2. set cursor2 at beginning of listD;
3. while(cursor1≠end of listA and cursor2≠end of listD)
4.  if(cursor1. docID<cursor2. docID)
5.        cursor1 ++;
6.  else if(cursor2. docID<cursor1. docID)
7.        cursor2 ++;
8.  else
9.        mark=cursor2;
10.         while (cursor2. position<cursor1. position and cursor2≠
                end of listD)
11.           cursor2 ++;
12.           if(cursor2=end of listD)
13.             cursor1 ++;
14.             cursor2=mark;
15.           else if(cursor1. val(directly)contains cursor2. val)
16.                mark=cursor2;
17.                do
18.                  merge cursor1 and cursor2;
19.                  cursor2 ++;
20.                  while (cursor1. val(directly)contains cursor2. val
                          and cursor2≠end of listD)
21.                  cursor1 ++;
22.                  cursor2=mark;
23.           endif
24.         endwhile
25.     endif
26. endwhile
```

4.2 索引改进归并连接算法 IIMGJN

一个 Xpath 路径表达式的计算通常是在当前上下文中进行的，在这个算法里提出了改进。首先利用索引来进行 docID 定位，再在已定位的文档中进行包含连接时，利用索引来改进归并连接算法。因此称该算法为索引改进归并连接算法，记为 IIMGJN[2]。

IIMGJN 算法与 MPMGJN 算法相比，主要改进有：

(1)考虑了 Xpath 路径表达式计算的当前上下文，并利用索引来进行文档定位，再在已定位的文档中采用索引改进归并方法进行包含连接。这样，可以避免对不在查询范围内的文档中元组进行扫描。

(2)采用短路技术，来减少外表扫描代价和内表搜索代价。

(3)采用预侦技术，来减少内表的搜索代价。

(4)采用索引定位技术，来避免外表中不参与连接的元组的扫描以及内表中不参与连接的元组的搜索。

5 基于缓存的归并结构连接算法

直接归并结构连接算法在处理祖先列表 Alist 中存在许多同名嵌套元素的包含关系的结构连接时，会对连接的内表执行重复的扫描，因此最坏情况下的性能较差。为了避免对内表的重复扫描，研究者提出了各种基于缓存的归并连接算法，包括基于栈的和基于队列的归并结构算法。

5.1 Stack-Tree 算法

为了避免对内表的重复扫描，采用基于栈的归并连接算法—Stack-Tree[3]，该算法通过维护一个祖先元素结点栈，仅仅需要对祖先列表 Alist 和后裔列表 Dlist 分别扫描一次就可以包含关系的结构连接。算法可以分别按后裔有序和祖先有序方法来实现。这里主要介绍前一种算法(Stack-Tree-Desc)算法。该算法中栈用来存放一个嵌套祖先元素结点的序列。假设栈中自底到顶依次存放了 n 个结点{s1,s2,…,sn}，对于每一个结点 Si($i\in[2,n]$)，它是结点 si－1 后裔结点。

Stack-Tree-Desc 算法的基本思想是：在归并连接过程中，对于 Alist 列表中扫描得到的一个新结点 a，如果它是目前栈顶结点的后裔，那么它将被压入栈；对于 Dlist 列表中扫描得到的一个新结点 d，如果它是目前栈顶结点的后裔，那么它也是目前栈中所有结点的后裔，并且可以得出结论：它不可能是 Alist 列表中其它结点的后裔，因为结点 d 在 Alist 列表中的所有祖先结点都已经进栈了。因此，结点 d 与栈中所有结点之间的连接结果可以被输出了。对于 Dlist 列表中扫描得到的一个新结点 d，如果它不是目前栈顶结点的后裔，那么可以得出结论：目前的栈顶结点在 Dlist 列表中没有进一步的后裔结点。因此，可以将栈顶结点出栈了，并且判断结点 d 是否是新栈顶结点的后裔，直到栈为空。

Stack-Tree 算法对祖先和后裔列表分别顺序扫描一次。但是，对于两个列表中有些结点，能够预先判断出它们是不参与连接的。因此，文[4]提出了 Anc_Desc_B+算法，该算法在结构连接过程中利用 B+树索引来跳过不参与连接的结点。

5.2 XR-Stack 算法

对于 Anc_Desc_B+算法，可以利用 B+树索引来跳过不参与连接的结点。但是，对于祖先 Alist 中不匹配的结点就不能有效地跳过。为此，文[5]提出一种 XR-树索引。XR 树索引不仅能够有效地跳过后裔列表 Dlist 中所有不匹配的结点，而且能够有效地跳过祖先列表 Alist 中所有不匹配的结点。利用 XR-Stack 索引来处理包含关系的结构连接时，两个基本的操作是查找一个给定元素结点的所有后裔结点或所有祖先结点。

(1)FindDescendants：给定一个元素结点 $E_a=(b_a,e_a)$，在一个 XR-树索引中发现它的所有后裔元素结点。该操作就是查找所有满足 b_a＜Ei. begin＜e_a 条件的元素结点 Ei，它是一个关于元素结点的 begin 键的范围查询。

(2)FindAncestors：给定一个元素结点 $E_d=(b_d,e_d)$，在一个 XR-树索引中发现它的所有祖先元素结点。该操作就是查找所有满足 Ei. begin＜b_d＜Ei. end 条件的元素结点 Ei，也就是说，需要搜索所有被 b_d 刺中的元素结点。由于被 b_d 刺中的所有元素结点可能分散在 XR-树的 b_d 搜索路径所到达的叶结点及其左边所有叶结点中，显然顺序搜索这些叶结点的代价是昂贵的，这正是提出 XR-树的动机。对于 XR-树，在从根到叶的搜索过程中，同时搜索每一个经过的内部结点的刺中列表，从中发现被 b_d 所刺中元素结点；在到达叶结点之后，再从该叶结点中输出那些并不包含在刺中列表中的被 b_d 所刺中的元素结点。

基于 XR-树索引的结构连接算法 XR-Stack，其算法思想是假设 A 和 D 是两个元素结点列表，它们都建立了 XR 树索引。对于一个列表，由于它的 XR 树索引的叶结点中有序地包含了这个列表中的所有元素结点的区间编码，因此基于 XR 树可以采用归并的方法对两个列表进行结构连接。同

时，基于 XR 树索引还可以利用前面介绍的两个函数来有效地跳过并不匹配的元素结点。

5.3 Twig 模式的结构连接

一个 Xpath 路径表达式查询可以通过小枝模式建模，称为小枝模式查询。

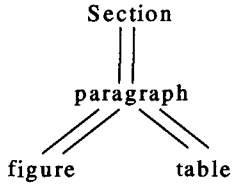

图 1 小技模式查询

查询：//section//paragraph [. //figure and. //table]

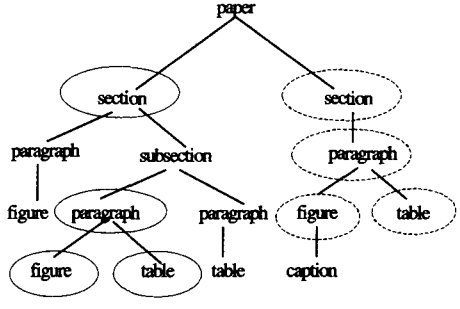

图 2 图 1 在一个 XML 文档树实例中的匹配

图 2 表示的是图 1 的小枝模式查询在一个 XML 文档树实例中的匹配情况。有两个小枝模式查询的结果被匹配出来。

模式匹配方法有两种：一是基于树遍历的模式匹配方法，二是基于集合的匹配方法。基于树遍历的匹配方法需要对 XML 文档树进行遍历，每一个 XML 文档树的结点都需要被访问一次，因此可能出现许多没有必要的遍历扫描。基于集合的匹配方法首先通过标记索引获得小枝模式中出现的每一个标记的接点集合，然后根据小枝模式的边对结点集合进行结构连接。Twig 方法是将小枝模式分解成一系列的结构连接，并且采用基于代价的优化方法来决定这些结构连接的执行顺序；Holistic Twig[6] 连接方法通过维护多个栈并将它们链接起来，按流水线方式实现小枝模式的所有结构连接。下面定义一些函数：

isLeaf：Node→Bool；isRoot：Node→Bool；parent：Node→Node；children：Node→{Node}；subtreeNodes：Node→{Node}。其中，subtreeNodes(q)返回查询 Q 中的结点 q 和它的所有后裔结点。

5.3.1 PathStack 算法

首先考虑不包含分支的小枝模式查询。例如 *section//paragraph//figure*，称它为路径模式查询。它的主要思想是：对于一个路径模式查询 q，在顺序扫描所有查询结点的流 T1，…，T*n* 的过程中，重复构造它的局部和整体匹配结果的栈链。

输入：路径模式查询 q，以及它的 *n* 个查询结点的流 T1，…，T*n*，这里 *n* 为查询 q 的查询结点数量
输出：路径模式查询 q 的匹配结果，它是按从叶到根有序的（即后裔有序）

```
PathStack(q,T1,…,Tn)
1. while(not end(q))
2.  qmin=getMinSource(q));
3.  for(every qi in subtreeNode(q))
4.    while(not empty(Si)and toEnd(Si)<nextBegin(Tqmin))
5.      pop(Si);
6.  moveStreamToStack(Tqmin, Sqmin,pointer to top (Sparent(qmin)));
7.  if(isLeaf(qmin))
8.      showSolutions(Sqmin,1);
9.      pop(Sqmin);
end(q)
  return ∀qi∈subtreeNode(q);isLeaf(qi)=>eof(Ti);
getMinSource(q)
  return qi∈subtreeNode(q)such that nextBegin(Ti)is minimal;
moveStreamToStack(Tq,Sq,p)
1. push(Sq,(next(Tq),p)
2. advance(Tq);
showSolutions(SN,SP)
1. index[SN]==SP;
2. if(SN==1)
3.    output(index[n],…,index[1]);
4. else
5.    for(i=1 to index[SN]. pointer)
6.       showSolutions(SN-1,i);
```

5.3.2 TwigStack 算法

小枝模式查询 Q 的匹配方法是：首先将小枝模式查询 Q 分解为多个从根结点 q 到每一个叶结点的路径模式查询 Q1，…，Qm。例如图 1 的小枝模式查询，可以把它分解为两个路径模式查询：section/paragraph/figure 和 section/paragraph/table。然后，用算法 PathStack 算法，单独得到每一个路径模式查询 Q*i* 的局部匹配结果。最后，将多个路径模式查询 Q*i* 的局部匹配结果粘和，得到最终的匹配结果。

Holistic Twig[6] 结构连接算法 TwigStack 算法。

输入：小枝模式查询 q，以及它的 *n* 个查询结点的流 T1，…，T*n*，*n* 为查询结点数量。
输出：小枝模式查询 q 的匹配结果，它是按从叶到根有序的（即后裔有序）。

```
TwigStack(q,T1,…,Tn)
1. while(not end(q))
2.  qact=getNext(q)
3.  If(not isRoot(qact)
4.       CleanStack(Sparent(qact),nextBegin(Tqact));
5.  If(isRoot(qact)or(not empty(Sparent(qact))));
6.       cleanStack(Sqact,nextBegin(Tqact));
7.            MoveStreamToStack (Tqact, Sqact, pointer to top
                (Sparent(qact)));
8.       If(isLeaf(qact))
9.            ShowSolutionsWithBlocking(Sqact,1);
10.            Pop(Sqact);
11.  Else advance(Tqact);
12.  MergeAllPathsolutions();
CleanStack(S,actBegin)
    1. While(not empty(S)and topEnd(S)<actBegin)
    2. pop (S);
getNext (q)
1. if(isLeaf(q))return q;
2. for(every qi in children(q))
3.     ni=getNext(qi);
4.     If(ni≠qi)return ni; Nmin=minargni {nextBegin(Tni)};
5. Nmin=minargni{nextBegin(Tni)};
6. Nmax=maxargmi{nextBegin(Tnmax)}
7. while(nextEnd(Tq);
8.     advance(Tq);
9. if(nextBegin(Tq)<nextBegin(Tnmin))return q;
10. else return nmin;
```

对于包含“/”边的小枝模式查询 q，TwigStack 算法不再保证它的 I/O 和 CPU 时间是最优的。TSGeneric+算法[7] 在 TwigStack 算法的基础上利用各种索引来尽可能多地跳过并不参加与连接的结点。

6 算法比较

MPMGIN 算法对于 ADR-Anc 算法的 CPU 时间和 I/O 复杂度均为 O(|Alist|+|Dlist|+|Output|)。如果祖先列表 Alist 中的元素不出现嵌套现象，则连接结果 Output 的大小是 O(|Alist|+|Dlist|)；如果祖先列表 Alist 中的元素出现嵌套现象，则对 Dlist 列表中相同后裔结点集合会出现多次扫描，因此 Output 的大小是 O(|Alist| * |Dlist|)；

最坏情况下，Stack-Tree 算法的 CPU 时间和 I/O 复杂度如下：

（下转第 176 页）

行时间为 0.4821 秒，在 Mexican Hat 数据集上的平均运行时间为 0.5457 秒，其运行效率是比较高的。

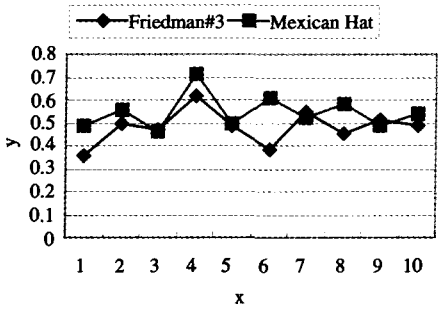

图 1 个体选择算法运行时间情况

结论 本文提出一种基于约束规划的选择性神经网络集成方法。该方法一般情况下能够得到最优解，且设计过程简单，效率较高，能以较小的代价提高神经网络集成的泛化能力。但该方法也存在一定的问题，可能会遇到求相关系数矩阵的逆阵的问题，而实际应用中，各网络的误差可能有较大的关联，这样个体网络的相关度矩阵是线性相关的，这时相关度矩阵的逆阵求解是相当困难的。为了避免这种情况，可以用一些方法[6,7]训练个体网络，使它们彼此之间相关度较小。

参 考 文 献

1 Krogh A, Vedelsby J. Neural network ensembles, cross validation, and active learning. In: Tesauro G, Touretzky D, Leen T, eds. Advances in Neural Information Processing Systems, Cambridge, MA: MIT Press, 1995,7:231～238
2 Partridge D, Yates W B. Engineering Mutiversion Neural-net Systems. Neural Computation, 1996, 8: 869～893
3 Wu Jianxin, Zhou Zhihua, Chen Zhaoqian. Ensemble of GA-based Selective Neural Network Ensembles. In: Proceedings of the 8th International Conference on Neural Information Processing (ICONP'01), Shanghai, China, 2001, 3:1447～1482
4 周志华. 神经计算中若干问题的研究：[博士学位论文]. 南京大学，2000
5 Friedman J. Multivariate adaptive regression splines. Annals of Statistics, 1991, 19(1): 1～141
6 Rosen B E. Ensemble learning using decorrelated neural networks. Connection Science, 1996,8(3-4):373～384
7 Tumer K, Ghosh J. Error Correlation and Error Reduction in Ensemble Classifiers. Connection Science, Special issue on combining artificial neural networks: ensemble approaches, 1996,8(3-4):385～404

（上接第 138 页）

(1)对于 Stack-Tree-Desc 算法，无论是祖先/后裔关系还是双亲/孩子关系的结构连接，它的 CPU 时间和 I/O 复杂度均为 O(|Alist|+|Dlist|+|Output|)，对 Alis 和 tDlist 列表只需要分别扫描一次，连接结果立即被输出；栈的大小是 XML 文档树的高度。

(2)Stack-Tree-Anc 算法，无论是祖先/后裔关系还是双亲/孩子关系的结构连接，它的 CPU 时间和 I/O 复杂度均为 O|Alist|+|Dlist|+|Output|)，对 Alist 和 Dlist 列表只需要分别扫描一次，连接结果需要进行缓存；栈的大小是 XML 文档树的高度。

模式匹配方法有两种：一是基于树遍历的模式匹配方法，二是基于集合的匹配方法。基于树遍历的匹配方法需要对 XML 文档树遍历，每一个 XML 文档树结点都需要被访问一次，因此可能出现许多没有必要的遍历扫描，以前的工作已经对模式树匹配问题做了大量研究。Twig 通过分解模式树为二元结构关系，分别连接，再把二元结果合并起来处理模式树匹配，但是会产生大量无用的中间结果。文[6]中提到的 TwigStack 算法通过维护多个栈并将它们链接起来，按流水线的方式实现小枝模式的所有结构连接。这样，可以将整棵模式树一起匹配处理，避免保存无用中间结果，它对于只有祖先后裔边的模式树匹配是最优的。但是它没有使用索引信息，必须对模式树中所有边都进行连接，而且它对于带有父子边的模式树不能保证最优。文[7]在 TwigStack 算法基础上使用了 XR-Tree 索引，可以跳过不参与连接的结点，但它也有 TwigStack 的缺点。

结论 目前，国内外科研人员在这方面做了大量的研究工作。中国人民大学孟小峰等提出了基于区域划分的 XML 结构连接[8]，该算法基于任务分解的思想，利用区域编码的特点对输入集合进行划分。该算法在输入数据无序或没有索引的情况下优于现有的排序合并算法，可以为查询计划提供更多的选择。江西财经学院万常选等提出的基于区间编码的 XML 索引结构有效实现结构连接[10]，给出了一个 XML 树数据模型的形式化定义。文[9]将编码方案、逆序列表和路径索引的思想相结合，提出了一种改进的 XML 数据的索引结构，给出了两个实现双亲/ 孩子关系和拥有关系的结构连接算法，它们最多只需要对参与连接的两个列表分别进行一次扫描，并且能够根据双亲结构信息等利用 B^+ 树索引尽可能多地跳过不需要参与连接的元素结点。通过国内外科研人员对 XML 数据库结构连接研究的不断深入，会有更多、更有效的连接算法出现，为 XML 数据库的查询优化奠定良好的基础。

参 考 文 献

1 Zhang C, Naughton J, DeWitt D, et al. On Supporting Containment Queries in Relational Database Management Systems. In: Mehrotra S, et al. eds. Proceedings of the 20th ACM SIGMOD Intdrnational Conference on Management of Data. Santa Barbara, California, USA. MAY, 2001. New York: ACM Press, 2001. 426～437
2 刘云生，万常选，徐升华. 基于关系数据库有效地实现 PRE 查询. 小型微型计算机系统，2003,24(10):1764～1771
3 Al-Khalifa S, Jagadish H V, Koudas N, et al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In: Hiong Ngu A H, et al. eds. Proceedings of the 18th IEEE ICDE International Conference on Data Engineering. San Jose, California, USA. February 26-March 1, 2002. Los Alamitos: IEEE Computer Socety, 2002. 141～152
4 Chen Shu-Yao, Vagena Z, Zhang Donghui. Efficient Structural Joins on Indexed XML Documents. In: Proceedings of the 28th VLDB International Conference on Very Large Database, Hong Kong, China. August 2002. San Francisco: Morgan KaufmannPublishers, 2002. 263～274
5 Jiang Haifeng, Lu Hongjun, Wang Wei, et al. XR-Tree: Indexing XML Data for Effcient Structural Joins. In: Casati F, et al. eds. Proceedings of the 19th IEEE ICDE International Conference on Data Engineering. Bangalore, India. March 2003. Los Alamitos: IEEE Computer Society, 2003. 253～264
6 Bruno N, Koudas N, Srivastava D. Holistic Twig Joins: Optimal XML Pattern Matching. In: Franklin M J, et al. eds. Proceedings of the 21th ACM SIGMOD International Conference on Management of Data. Madison, Wisconsin, USA. June 2002. New York: ACM Press, 2002. 310～321
7 Jiang Haifeng, Wang Wei, Lu Hongjun, et al. Holistic Twig Joins on Indexed XML Documents. In: Heuer A, et al. eds. Proceedings of the 29th VLDB International Conference on Very Lage Database. Berlin, Germany. Sept. 2003. San Francisco: Morgan Kaufmann Publishers, 2003. 273～284
8 王静，孟小峰，王珊. 基于区域划分的 XML 结构连接. 软件学报，2004,15(5):720～729
9 Wang Wei, Jiang Haifeng, Lu Hongjun, et al. PbiTree Coding and Efficient Processing of Containment Joins. In: Casati F, et al. eds. Proceeding of the 19th IEEE ICDE International Conference on Data Engineering. Bangalore, India. March, 2003. Los Alamitos: IEEE Computer Society, 2003. 391～402
10 万常选，刘云生，徐升华，等. 基于区间编码的 XML 索引结构有效实现结构连接. 计算机学报，2005,28(1)