优化的XML查询匹配:基于B$^+$-Tree索引的包含段的结构化联接算法*)

樊小华 庞引明 张 谧 汪 卫 陈金海 施伯乐
(复旦大学计算机与信息技术系 上海200433)

摘 要 高效的结构化联接方法是XML查询的关键。本文提出一种新颖的结构化联接方法,使用了包含段结构化XML文档树,并且使用了B$^+$-Tree索引技术支持该新方法,从而在基于栈的结构化联接过程中得以忽略若干时空耗费,提高处理效率。

关键词 XML,结构化联接,包含段,B$^+$-Tree

Optimal XML Query Pattern Matching: A Structural Join Based on Containment Segment Indexed by B$^+$-Tree

FAN Xiao-Hua PANG Yin-Ming ZHANG Mi WANG Wei CHEN Jin-Hai SHI Bai-Le
(Dept. of Computing and Information Technology, Fudan University, Shanghai, 200433)

Abstract *Efficient structural join is thus the key to efficient implementations of XML queries. This paper proposes a novel method for structural joins, and uses Containment Segments structuralizing the tree of XML Documents and the B$^+$-Tree index served for this new method. Thereby, we can save some time and space cost in the stack-based structural joins, which gains better processing performance.*

Keywords XML, Structural join, Containment segment, B$^+$-Tree

1 引言

XML[1]已经成为当前数据库研究的热点。XML文档是一个带有标签的有序树,树中的节点表示元素或者属性,而树中的边表示元素与子元素(或属性)之间的关系。XML查询语言(比如:XQuery[2],Quilt,Xpath)都使用以树为中心的表示方式。如下,一个XQuery路径表达式:

book [tittle = 'XML']//author[fn='jane' AND ln ='doe']

该表达式查询的book元素满足如下条件:(1)author元素有一个子元素fn,fn的内容为'jane'字符串;并且有一个子元素ln,ln的内容为'doe'字符串。(2)author元素是book元素的子孙元素;并且book元素还有一个子元素title,title的内容为'XML'字符串。

这样一个复杂的运算通常是被分解成一组父子或前后代关系结点对,即:前后代关系(book,author)和父子关系(book,tittle),(tittle,XML),(author,fn),(fn,jane),(author,ln),(ln,doe)。

于是,该查询模式可以这样来完成:1)在XML数据库中匹配上述分解的二元结构关系;2)将查找到的二元结构匹配结果组装(stitching)成最终的符合上述查询表达式的完整树结构。

显然,问题的关键在于第一步,它是XML查询处理的核心操作。而完成这一步的方法是将匹配每一个二元组中两元素的XML树中的结点,根据其索引结构进行结构化联接(structural joins),得到所需要的二元组匹配结果。

文[3]中基于表示元素位置关系的三元组(第2节中会详细讨论)索引,提出一种多谓词归并联接算法MPMGJN(multi-predicate merge join),实现了关系库中存储的XML数据的包含查询(containment queries)。文[6]在此基础上,正式将结构化联接的概念引入XML数据库的查询操作中,实现了基于栈的结构化联接算法。文[7]针对文[6]中的算法提出,可以将匹配祖先结点的A-list队列加入B$^+$树索引,从而能够依据索引结构特点忽略掉该队列中的若干元素,使其不参加联接过程,以此提高联接操作的效率。

其实,通过对文[4,5]等文献的研究可以发现,所谓优化措施,均源于对A-list和D-list队列的索引结构特点的认识。认识上的每一次进步,都会带来联接优化效率的进一步提高。

本文正是基于这一点认识,通过对两个参与结构化联接的队列索引结构的认真分析,认清XML文档树的结构特点,得出其索引结构——包含段(Containment Segment,第3节中会着重阐述其概念),然后以此为基础提出了更为优化的同时忽略两个队列中不必联接的元素结点的结构化联接方法。

2 背景和相关工作

XML数据库是一个由有根、有序、带标记的树组成的森林。对于XML查询来说,快速的判断两个节点之间是否是父子关系或前后代关系非常重要。所以,下面首先讨论XML编

*)本文受国家自然科学基金重点攻关项目"电子图书馆的相关关键技术"(No. 69933010)和863攻关项目"基于Web服务的数据库新技术"(No. 2002AA4z3430)资助。樊小华 硕士研究生,研究方向为XML。庞引明 博士研究生,研究方向为移动计算、XML。张 谧 硕士研究生,研究方向为XML、遗传算法。汪 卫 博士,副教授,研究方向为数据挖掘、XML。陈金海 教授,研究方向为数据库与知识库。施伯乐 首席教授,博士生导师,研究方向为数据库与知识库。

码方案，然后介绍现有的结构化联接算法，最后讨论为加快算法而引入的一些常用索引技术。

2.1 XML 编码方案(Numbering Scheme)

XML 编码方案必须要能体现 XML 文档树的结构，并且能快速的判断其结构关系。一种方案是对每一个文档节点赋一个三元组的值(*preorder*，*postorder*，*level*)[3,4,6]，这三个值这分别是节点在文档树中的前序编号、后序编号和深度。按照树的结构特点，只要知道其前序遍历和后序遍历，就可以还原整棵文档树。

但是为了能支持文档树的更新，文[4]提出了可持久的(durable)编码方案，在这个方案里面每个文档节点赋予三元组值(*start*，*end*，*level*)，其中要求 *start* > *end*，但 *start* 和 *end* 还是按照前序遍历和后序遍历排列。这样，*start* 值和 *end* 值由元素节点在文档树中的位置决定。对于任意两个不同的元素节点 u、v，则它们之间的关系，(1)如果 $u.end<v.start$，则 u 完全在 v 的前面；(2)如果 $u.start<v.start<u.end$，则 u 是 v 的前代节点；如果再加上条件 $u.level=v.level-1$，则 u 是 v 的父节点。

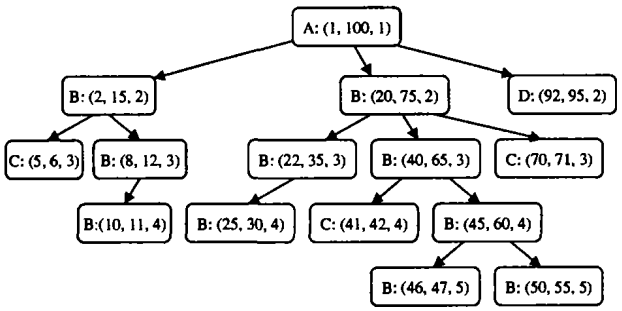

图1 XML 文档树编码示意图

图1举例了一棵经过编码后 XML 文档树，树的根节点 A (1，100)处在第1层，从位置1跨越到位置100，而它的第二个子元素 B 处在第2层，从位置20跨越到位置75。

2.2 结构化联接算法(Structural Join)

结构化联接算法[6,7]就是要找到两个元素集合之间的匹配对集合，这些匹配对满足前后代关系或父子关系。更准确地说：给定两个输入序列 *A-list*(作为候选前代的序列)和 *D-list*(作为候选后代的序列)，序列中的每个元素都格式化成(*start*，*end*，*level*)的形式，并且两个序列都按 *start* 从小到大排序，结构化联接算法的目的是找到所有这样的匹配对(ai，dj)，满足(1)$ai\in A\text{-}list$，$dj\in D\text{-}list$；(2)$ai.start < dj.start < ai.end$，也就是 dj 是 aj 的后代；(3)如果要求查询父子关系的节点对，则还要满足 $ai.level = dj.level - 1$。

如上所述的结构化联接算法，就是文[6]中提出的 Stack-Tree 联接算法。它虽然比文[3]中的多谓词归并联接算法 *MPMGJN* 更进了一步，但它却是一个很一般的基于栈的结构化联接算法。它不考虑 *A-List* 与 *D-List* 序列的索引结构特点，以致于不能忽略(或跳过)不必进栈或不必进行联接操作的结点。

文[7]中分析了文[6]中的这些不足，提出了 Anc-Des-B^+ 算法，以 B^+-Tree 作为索引，该算法可以快速跳过 AList 中部分不必参加联接运算的结点，从而达到提高效率之目的。但它只对整棵文档树建立索引，并没有分析 XML 文档结构特点从而对索引作相应的改进，工作完成得并不彻底。

至于文[8]的工作，是为了避免上述工作中产生大量无用的中间结果而提出的。它的核心算法：PathStack 算法与 TwigStack 算法采用多个栈一次生成查询的最终结果(或其从根到叶子的长枝)。它与本文的关系并不密切，它存在的问题是没有考虑联接次序的选择，效率可以进一步提高，作者会在将来工作中引入文[8]的方法加以改进。

2.3 XML 索引(XML Indexing)

针对 XML 数据曾经提出多种不同的索引技术。早期提出的索引技术不适用于三元组式的 XML 编码方式，这些索引创建一个 XML 文档摘要，采用标签有序树的结构，这些索引本身就类似于 XML 文档的结构，但是使用了较少的节点和边。

文[4]首次采用了 B^+-Tree 对 XML 数据建立索引，而在文[7]中，作者提出了基于 B^+-Tree 索引的结构化连接算法 Anc_Des_B^+，该算法由于应用了 B^+-Tree 从而可以越过一些不必要的节点检查，减少读取 XML 节点的数量和 I/O 开销，加快结构化连接查询的速度。而其缺点上文已有所阐述。

文[10]提到了专门针对 XML 数据结构和查询特点的 XR-Tree 索引，它可以认为是 B^+-Tree 的一个扩展版本，该索引可以方便的获得某一个节点的所有前代节点或者所有后代节点。然而该索引不能被流行的数据库管理系统所支持，而 B^+-Tree 已经能为包含段(Containment Segment)建立良好的索引，所以本文作者并没有采用 XR-Tree 索引。

3 算法

本算法涉及到一个新的概念：包含段(Containment Segment 或 CS)。下面的段落首先介绍包含段的概念，然后编码包含段并且为包含段建立 B^+-Tree 索引，最后给出基于该索引的算法描述。

3.1 包含段的概念(Concept of Containment Segment)

定义1 给定一个结点输入序列 *AList*(或 *DList*)，$CI_{AList}=\{CI_{a1},CI_{a2},\cdots,CI_{an}\}$(或 $CI_{DList}=\{CI_{d1},CI_{d2},\cdots,CI_{dm}\}$)称为其包含区间序列。则 CI_{AList}(或 CI_{DList})中一组连续的存在包含关系的区间称为一个包含段 *CS*(*Containment Segment*)。如：$CI_{ai}\supset CI_{ai+1}\supset\cdots\supset CI_{ai+p}$为一包含段，其长度为 $p+1$。CI_{AList}(或 CI_{DList})中可能存在多个包含段，其中长度最大的称为最长包含段。显然，CI_{AList}(或 CI_{DList})的最短包含段长为1。

有了上述的定义，就可以将两个结点输入序列的 *AList* 和 *DList* 的完整索引结构形式化为如下定理：

定理1 结点输入序列(AList 或 DList)的形式化的索引结构为：

$$LS=(CS\,|\,\varepsilon)(\bigcup_{i=1}^{n}LS\,|\,\varepsilon)$$

这是一个递归描述。其中的 *LS* 代表序列结构(List Structure)，*CS* 代表一个包含段(Containment Segment)。

图2给出一个直观的索引结构。CI_1是序列结构 LS_1中的一个 XML 节点，同时又是包含段 *CS*1的首区间。LS_1包含 CS_1和左右两个序列结构 LS_2、LS_3。由于序列结构的递规定义，LS_2、LS_3可能还包含其他的包含段和序列结构。

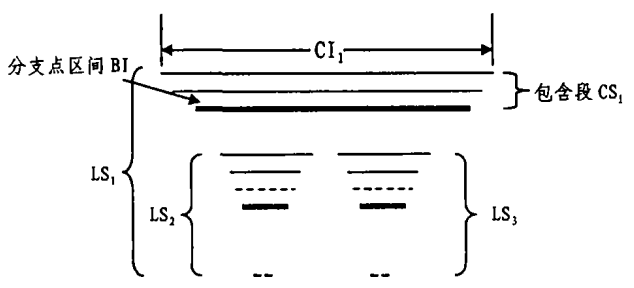

图2 基于 CI 的输入序列索引结构示意图

如图2所示，一个包含段中直接包含多个分支子序列的最小包含区间 CI 称为分支点区间 BI(Branch Interval)，它在图3中用粗线条表示。下面给出其形式化定义：

定义2 给定一个包含段 CS_i，如果$\exists\ CI_i \in CS_i$，使得 $\bigcup_{j=1}^{n} LS_j \subset CI_i$ 且$\forall\ CI_k \in CS_i$ 有 $CI_i \subset CI_k$(CI_i 最小)成立，则称 CI_i 为包含段 CS_i 的分支点区间 BI。

定义分支点区间 BI，是为了便于找到各子序列 LS_i 的入口。

下面，本文给出一个很重要的推论。

推论1 $AList$(或 $DList$)$=\bigcup_{i=1}^{m} CS_i$，其中，要么$\bigcap_{i=1}^{m} CS_i = \phi$；要么$\exists\ CI_k, CI_{k+1}, \cdots, CI_{k+m-2} \in CS_1$，使得 $CS_2 \subset CI_k, CS_3 \subset CI_{k+1}, \cdots, CS_m \subset CI_{k+m-2}$。

从定义、定理和推论可以看出，XML 文档树经包含段压缩之后，同样呈现树状的结构：包含段之间存在父子关系。这棵经压缩后的包含段树中的节点比原本 XML 文档树中的要来的少，特别是对于那些带有深度嵌套甚至带有递规嵌套的 XML 文档树尤其如此。同时包含段树还带有更多的结构信息。这样经索引后的包含段树比简单的对 XML 文档树建索引有更好的效率、更少的空间消耗。利用包含段的这种特点，就可以进行结构化联接操作的优化，尽可能跳过那些不必要的 XML 节点检查。

3.2 包含段的编码方案(CS Numbering Scheme)

包含段的编码方案建筑在 XML 节点编码方案之上。从上面的介绍，知道了 XML 三元组式的可持久的编码方案。本文对其进行多文档扩展，同时加入一些必要的节点内部信息，则 XML 节点表示为(Doc, Tag, $startpos$, $endpos$, $level$, $value$)，其中 Doc 为该节点所在的文档名，Tag 为元素或属性的标签名，$value$ 存储属性的值。

依照如上扩展后的编码方式，同时知道包含段是连续的相互包含的 XML 节点序列，则可以把包含段编码为(Doc, Tag, $Sstart$, $Send$, $Pointer$)，其中 Doc, Tag 为序列中 XML 节点的 Doc 和 Tag，$Sstart$ 为包含段中首区间的 $startpos$ 值，$Send$ 为包含段中尾区间的 $startpos$ 值，$Pointer$ 指针指向首区间。

经编码后的包含段有如下特点：1)对于任意的 XML 节点 u 和包含段 c，如果 $c.Sstart \leqslant u.startpos \leqslant c.Send$，则节点 u 属于包含段 c。2)对于任意的两个包含段 c，d，不存在这样的情况，$c.Sstart \geqslant d.Sstart \geqslant c.Send$。对后一个命题的证明很显然，如果包含段相互包含，就丧失了包含段的意义。

3.3 包含段的 B$^+$-Tree 索引(CS Indexing)

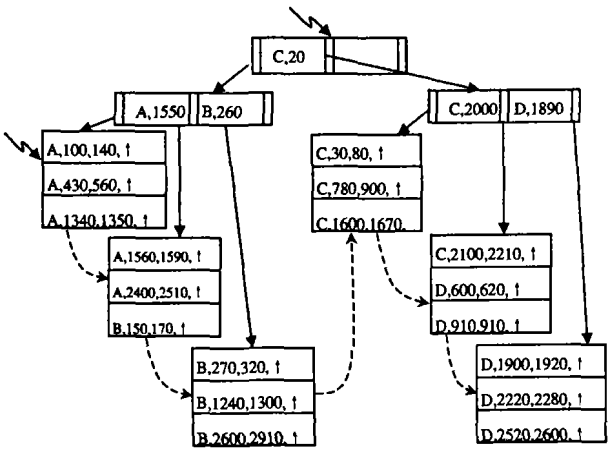

图3 包含段的 B$^+$-Tree 索引结构示例图

依照上面的包含段的编码方案对包含段建立 B$^+$-Tree 索引。B$^+$-Tree 索引的键值(key)必须能区别开不同一的 Doc 和 Tag，所以使用(Doc，Tag，Sstart)三元组作为 B^+-$Trees$ 索引的键值。在 B^+-$Tree$ 中，非叶节点是由 n 个键值和 n+1个指针(指向子节点)构成；而叶节点是由 m 个包含段和一个链表指针(指向下一个叶节点，从而把所有的叶节点串起来)构成。B^+-$Tree$ 还维持两个入口指针，一个指向树的根节点，一个指向首个叶节点。如图3为一个包含段 B^+-$Tree$ 索引结构的示例图，为了简单起见，把 Doc 值略去。

3.4 基于包含段 B$^+$-Tree 索引的结构化联接算法(CS-BP-SJ)

限于篇幅，此处仅给出算法的描述。

```
/* 算法主函数 */
Function CS-BP-SJ(Alist,Dlist)
    指针 A = Alist 首区间
    指针 D = Dlist 首区间
    初始化 Stack,使栈为空
    while (Dlist is not empty)
        /* 清理栈内的节点 */
        Temp = nil;
        while(stack is not empty && stack's TOP before D)
            temp = pop();
        if(Alist is not empty && temp != nil)
            A = 大于 temp.endpos 首包含段的首区间 /* 有跳转 */
        Endif
        /* 把 D 的前代压入栈中 */
        While(Alist is not empty)
            if(A.endpos < D.startpos)      /* 也就是 A 在 D 的前面 */
              A = 大于 A.endpos 首包含段的首区间 /* 有跳转 */
            Else If(A.startpos < D.startpos)      /* 也就是 A 是 D 的前代 */
                Push(A);
                AdvanceEx(A);
            Else 跳出当前 while 循环      /* A.startpos > D.startpos */
        EndWhile
        /* 输出匹配结果 */
        For (every node S in stack )
            Output(S, D);
        EndFor
        /* D 指针往后移动 */
        If(stack is empty)
            D = 大于 A.end 首包含段的首区间      /* 有跳转 */
        Else
            AdvanceEx(D);
        EndIf
    EndWhile
EndFucntion
Fuction AdvanceEx(Pointer)
    Advance(Pointer)
    /* 判断指针是否越过了当前包含段 */
    If(Pointer.Startpos > CS.Send)
        Advance(CS)
    Endif
EndFunction
```

4 实验结果及分析

4.1 实验环境

我们用来完成实验的机器配置为：Pentium Ⅲ 1.0 GHz，256M 内存。操作系统是 Windows XP，所用编程工具为 Java SDK。

实验数据来源于在前人工作中被广泛用于实验的 IBM XML 数据生成器(IBM XML data generator[11])，我们同时编写了一个简易的存储管理器用于在文[12]的基础上将生成的 XML 数据解析(parser)后装入 DB2 8.0中。解析的目的是将 XML 数据变成三元组形式装入关系库。

4.2 实验结果及分析

我们将本文中的算法与文[6,7]进行了三项对比实验。在实验中，Stack-Tree-Desc[6]、Anc_Des_B$^+$[7]和本文算法 CS-

BP-SJ 分别被称为 algm1、algm2和 algm3。

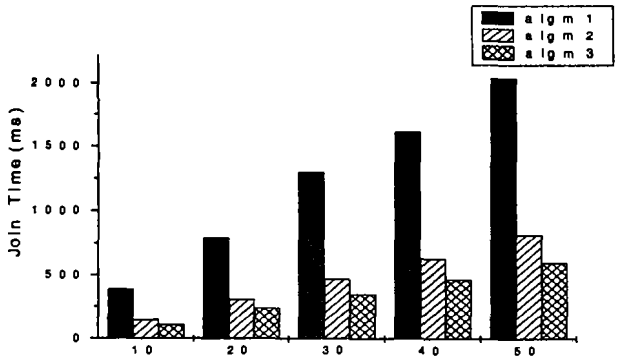

图4 Average Sibling Count

图4中所显示的实验结果是表明 Alist 和 Dlist 平均分支数的多少对结构化联接操作的影响。分支数越多，CS-BP-SJ 算法的优势更明显。

本实验中平均包含段的长度为10，横坐标的平均分支数分别取为10、20、30、40、50。纵坐标为算法的执行时间。不同分支数时所用的数据量如表1所示。

表1

原始分支数	10	20	30	40	50
数据量(K)	80.5	161	241	321	402

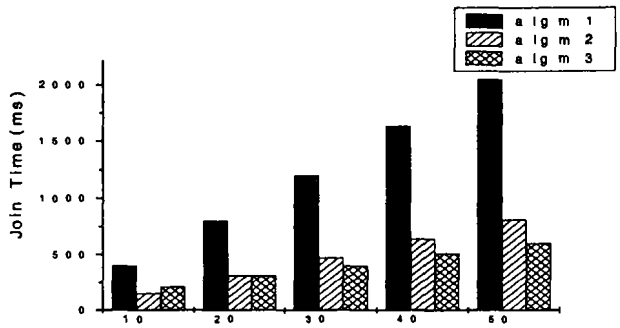

图5 Average C S Length

图5中所显示的实验结果是表明 Alist 和 Dlist 平均包含段的长度对结构化联接操作的影响。包含段的平均长度直接影响着本算法 CS-BP-SJ 是否优于算法 Anc_Des_B$^+$。实验表明包含段越长，包含段的索引结构才越优于 Anc_Des_B$^+$中的索引结构，所以算法 CS-BP-SJ 更适合于嵌套程度深甚至带递归嵌套的 XML 文档查询。

本实验中平均分支数为10时，通过变化包含段长度从10到50所得到的实验结果。其中的数据量分别为：

包含段长度	10	20	30	40	50
数据量(K)	21.5	72.7	153	263	402

图6表示的是有效数据量的变化对三个算法的影响。有效数据量是指在 Alist 和 DList 中，最终有多少数据是实际参加联接的。这个参数实际上反映了算法的适用范围。本实验中，同时固定分支数为10和包含段长度为10。实验的数据量固定在31.7K，通过变化有效数据量来观察三个算法的表现。从实验结果可以看出，随着有效数据量的上升，以跳转为基础的 Anc_Des_B$^+$和 CS-BP-SJ 逐渐逊色于没有多少跳转语句的 Stack-Tree-Desc。这是因为，随着有效数据量的上升，Anc_Des_B$^+$和 CS-BP-SJ 实际上已经没有多少跳转余地。大量的跳转语句只会降低效率。Stack-Tree-Desc 则因为不含大量的跳转语句而性能稳定。这说明本文的算法 CS-BP-SJ 和 Anc_Des_B$^+$一样适用的范围为 Alist 和 DList 中的有效数据量较小的时候。

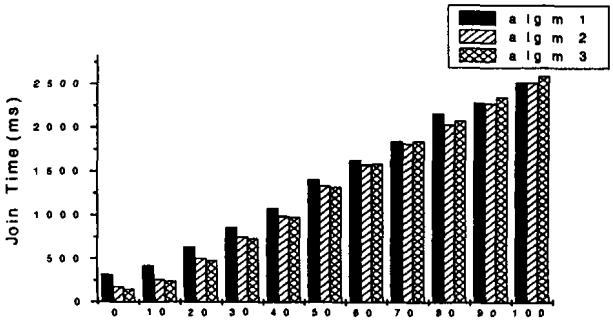

图6 Effective Nodes/Whole Nodes(%)

结束语 本文描述 XML 查询匹配的特点，分析了现有的几种结构化联接算法，依据 XML 文档的特点，提出了包含段(Containment Segment)的概念，并且按照包含段的新颖特征设计了其编码方案(CS Numbering Scheme)和 B$^+$-Tree 索引方法(CS Indexing)。最后在此基础之上，提出了基于包含段 B$^+$-Tree 索引的结构化联接算法(CS-BP-SJ)并且给出算法实验，从理论上和实验上证实了该算法优越的联接效率。

参 考 文 献

1 Paoli R J, Sperberg-McQueen C M. Extensible markup language (XML) 1.0 X3C recommendation. http://www.w3.org/tr/1998/rec-xml-19980210, Feb. 1998

2 Boag S, Chamberlin D, Fernandez M F, et al. XQuery 1.0: An XML query language. In W3C Working Draft, Aug. 2002. http://www.w3.org/TR/xquery/, 2002

3 Zhang C, et al. On supporting containment queries in relational database management systems. In SIGMOD, 2001. 425～436

4 Li Q, Moon B. Indexing and querying XML data for regular path expressions. In VLDB, 2001. 361～370

5 Dietz P F. Maintaining order in a linked list. In ACM Symposium on Theory of Computing, 1982. 122～127

6 Al-Khalifa S, et al. Structural joins: A primitive for efficient XML query pattern matching. In ICDE, 2002

7 Chien S-Y, et al. Efficient structural joins on indexed XML documents. In VLDB, 2002

8 Bruno N, Koudas N, Srivastava D. Holistic Twig Joins: Optimal XML Pattern Matching. ACM SIGMOD 2002

9 Jiang H, Wang W, Lu H. Holistic twig joins on indexed XML documents. VLDB 2003

10 Jiang H, Lu H, Wang W, Ooi B C. XR-Tree: Indexing XML data for efficient structural join. In ICDE, 2003

11 SAX (Simple API for XML). http://sax.sourceforge.net

12 IBM XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator