第 34 卷 第 4 期
Vol.34 No.4
计 算 机 工 程
Computer Engineering
2008 年 2 月
February 2008
• 软件技术与数据库 • 文章编号：1000—3428(2008)04—0063—03 文献标识码：A 中图分类号：TP311.13

XQuery 在 XML 流上查询的结构化连接

吴晓勇，张 昱，孙东海
(中国科学技术大学计算机科学技术系，合肥 230027)

摘 要：通过分析 XQuery 查询与 XPath 查询的区别与联系，定义了扩展的基本 XSIEQ 机 E-XSIEQ，它是一种被索引化、基于栈的自动机。提出用变量表来收集 XPath 查询结果，并将这些中间结果组织成原子表集合，结果构造时能够根据原子表元组之间的上下文关系，对原子表集合快速地进行连接。描述了 XQuery 查询过程中的结构化连接算法，给出了结果构造的时间性能分析。

关键词：自动机；分层缓冲区；结构化连接

Structural Join in XQuery Processing on XML Stream

WU Xiao-yong, ZHANG Yu, SUN Dong-hai
(Department of Computer Science & Technology, University of Science & Technology of China, Hefei 230027)

【Abstract】 An extended XML Stream Query with Immediate Evaluation(XSIEQ) machine for XQuery is defined, which is a kind of indexed automata based on stack. Variable table is used to reserve the XPath query results, and these results form into a set of atom tables. The algorithm on structural join is introduced and the result-constructing performance is analyzed.

【Key words】 automata; layered buffer; structural join

一些基于 XPath 的 XML 流查询引擎能支持简单的 XQuery 查询，但是由于 XQuery 的嵌套和多关键字排序等特性增加了 XQuery 查询处理的复杂度。近年来出现了一些专门的 XQuery 查询工具，不过它们对 XQuery 查询的支持力度比较有限。XSIEQ(XML Stream Query with Immediate Evaluation)[1]是笔者所在实验室研发的一个 XML 流查询引擎，能支持多个 XPath 式的同时查询，记为 XSIEQ-XP。本文对 XSIEQ 进行了功能扩展，使其能支持较复杂的 XQuery 查询特性，记为 XSIEQ-XQ。本文在 XSIEQ 上扩展增加对 XQuery 查询的支持；提出用变量依赖树来保存 XQuery 脚本中各个变量之间的依赖关系，并把 XPath 查询收集的结果用变量表保存，便于构造结果；提出了变量表连接的 JIT(Just in Time)算法和原子表连接的 JAT(Join Atom Tables)算法，通过实验证明了变量表、原子表的结构化连接给查询性能带来较高提升。

1 预备知识

1.1 XSIEQ-XP

XSIEQ-XP 系统框架是基本 XSIEQ 机。它的输入是解析 XML 流产生的 SAX 事件，输出是匹配的 XML 片段。

定义 1 对于 1 个 XPath 式 P，P 的主 XPath 式是指去除 P 中所有谓词后剩余的部分。主 XPath 式匹配的 XML 结点称为候选结果。

基本 XSIEQ 机的构造过程为：先将各查询 XPath 式中的主 XPath 式和谓词中的 XPath 式提取出来，利用前缀共享的方法增量式地构造为一个 NFA；然后对 NFA 中以下特殊状态进行类型标记。

定义 2 结果状态是匹配主 XPath 式的 NFA 状态，即主 XPath 式的最后一个位置步到达的状态。叶子状态是匹配谓词中 XPath 式的 NFA 状态。分支状态是某 XPath 式对应的主路径上具有的、分支到谓词中 XPath 式的 NFA 状态。

文献[1]为每个 XPath 式引入一个分层缓冲区，XPath 式缓冲区的层次取决于其中含谓词的位置步个数，缓冲区第 0 层的结果就是 XPath 表达式的最终结果。

1.2 XQuery

XQuery 语言的核心是 XPath 式和 FLWOR 表达式。

定义 3 对任意作用在单文档上的 FLWOR 表达式，有且仅有一个根变量对应文档的根结点。在 where, order, return 子句中出现的变量称为结果变量。结果变量关联的 XPath 式称为结果 XPath 式。一次查询中，变量关联的 XPath 式匹配的值称为绑定序列。

XSIEQ-XP 查询的都是 XPath 式，查询中只要收集主 XPath 式最后一个位置步所匹配的 XML 结点，再由谓词计算结果，通过逐层缓冲提升确定 XPath 式匹配的最终结果。在 XSIEQ-XQ 中，每个变量都关联一个 XPath 式，但只要收集用于构造结果文档的结果 XPath 式的结果即可。

在 XSIEQ-XP 查询中，各 XPath 式之间是独立的；而在 XSIEQ-XQ 查询中，XPath 式绑定到的变量间具有依赖关系。

定义 4 对于一个 FLWOR 表达式中的某一变量 v_j，如果 v_j 的定义为 for v_j in v_i+p_j 或者 let $v_j := v_i$+p_j，其中，“+”表示串联；p_j 是用 XPath 式表示的路径，则称 v_j 直接依赖于 v_i，记为：$v_i \xrightarrow{p_j} v_j$，其依赖关系满足传递性，即：$v_i \xrightarrow{p_j} v_j$，$v_j \xrightarrow{p_k} v_k \Rightarrow v_i \xrightarrow{p_{jk}} v_k$，其中，$p_{jk}$ 是串联 p_j 和 p_k 得到的 XPath 式，称 v_k 传递依赖于 v_i。任意变量 v_k 传递依赖于根变量，从根变量到 v_k 的路径称为 v_k 的导航路径。一般地，将直接依赖

基金项目：国家自然科学基金资助项目(60673126)；中国科学院计算机科学重点实验室开放课题基金资助项目(SYSKF0502)
作者简介：吴晓勇(1977－)，男，硕士研究生，主研方向：XML 数据处理；张 昱，副教授、博士；孙东海，硕士研究生
收稿日期：2007-04-22 **E-mail**：yuzhang@ustc.edu.cn

和传递依赖统称为变量绑定依赖。

示例 XQuery 查询脚本示例 *xq* 定义如下：

```
for $b in doc(bib.XML)//book,
  $p in $b/publisher , $a in $b/author
let $t := $b/title
return <result>
        {$b,$t,$p,$a }
    </result>
```

在 *xq* 中，变量 *b* 与 *t*, *a*, *p* 之间具有直接依赖关系。

2 XSIEQ-XQ 系统实现

针对 XQuery 中 XPath 式的特点，找到实现 XSIEQ-XQ 的途径，如扩展基本 XSIEQ 机、改造缓冲区的设置与组织、用变量表保存 XPath 查询结果等。

2.1 扩展的 XSIEQ 机

在 XSIEQ-XQ 中，需要收集结果 XPath 式匹配的 XML 结点，因此，可以按文献[1]的方法为结果 XPath 式构造基本 XSIEQ 机进行查询。由于 FLWOR 表达式中的 for 变量不仅有导航功能，而且控制着查询结果的结构化输出，因此也要将 for 变量关联的 XPath 式构造到 XSIEQ 机中以标识它们匹配的 XML 结点的位置。这就需要扩展基本 XSIEQ 机来得到 E-XSIEQ 机。

定义 5 对于 E-XSIEQ 机中的任意 NFA 状态 *s*，如果 *s* 是匹配结果 XPath 式的主 XPath 式，则称为结果状态。如果 *s* 是匹配 for 变量关联的 XPath 式，则称为 for 变量绑定状态。结果状态和 for 变量绑定状态的交集可能不为空。

图 1 是为 XQuery 表达式 *xq* 建立的 E-XSIEQ 机，其中，状态用圆圈表示；双实线圈表示结果状态，如 S_1, S_2, S_3, S_4；带阴影的圆圈表示 for 变量绑定状态如 S_1, S_3, S_4；弧代表状态转换，虚线表示子孙轴类型的转换，实线表示孩子轴类型的转换，弧上的标记表示结点测试。

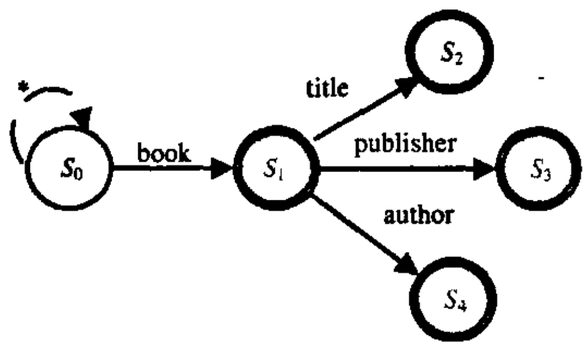


图 1 由 *xq* 提取 XPath 式构造的 E-XSIEQ 机

2.2 变量依赖树

XSIEQ-XQ 在解析 XQuery 查询脚本时获得并维护变量之间的依赖关系，根据这些依赖关系可以在 XPath 查询时组织各结果 XPath 式的查询结果，变量之间的依赖关系定义为变量依赖树。

定义 6 变量依赖树 $VarTree = (V, E, P)$。其中，V 是结点集合；E 是边集合；P 是边标记集合。$\forall v_i \in V$，v_i 表示变量，根结点表示根变量 r；$e=(v_i, v_j) \in E$ 表示直接依赖 $v_i \xrightarrow{p_j} v_j$，其中，$p_j \in P$ 表示 v_j 的关联 XPath 式的路径。

由示例的查询 *xq* 构造的 *VarTree* 如图 2 所示。

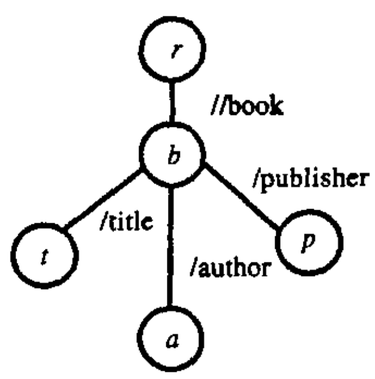


图 2 例子的变量依赖树

2.3 缓冲区的组织与设置

各变量对应的缓冲区根据变量之间的依赖关系，同样组织成树型结构，称为缓冲树。

定义 7 给定一棵变量依赖树 $VarTree=(V,E,P)$，其对应的缓冲树 $BufTree = (VP, EP, B)$，其中，VP 是结点集合；EP 是边集合；B 是结点标记集合。$\forall v_i \in V$，$\exists vp_i \in VP$，vp_i 表示变量 v_i 的关联 XPath 式的路径；$b_i \in B$ 表示 vp_i 的缓冲区；*BufTree* 中的根结点 p_r 表示 *VarTree* 中根变量 r 的路径，是一个虚结点，p_r 没有缓冲区。$\forall ep = (vp_i, vp_j) \in EP$，$\exists e = (v_i, v_j) \in E$ 且 $v_i \xrightarrow{p_j} v_j$。

图 3 是由示例中 *xq* 变量的缓冲区组成的缓冲树，由于 4 个变量关联的 XPath 式都没有谓词，因此缓冲区都只有 1 层。

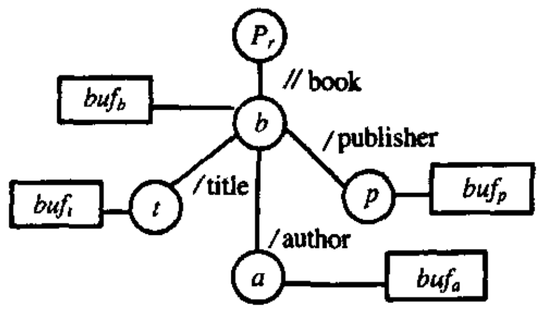


图 3 示例的缓冲树

在 XSIEQ-XQ 中，XQuery 脚本中的任一 XPath 表达式 *xp* 可以表示成：变量 *v* +路径 *p*，其中变量 *v* 绑定的 XPath 表达式 *pxp* 又可以是“变量 *v'*+路径 *p'*”的形式。如果候选结果缓存使用与 XSIEQ-XP 相同的策略，*xp* 缓冲区第 0 层的结果是 *xp* 的路径 *p* 中谓词过滤后的结果，但是没有经过 *pxp* 中谓词的过滤，不是 *xp* 的最终结果。因此，本文改进原有的分层缓冲区设置：对变量 *v* 的关联 XPath 式的路径 *p* 而言，如果 *p* 有 *m* 个带谓词的位置步，则 *p* 的缓冲区有 *m*+1 层，第 *m* 层保存未进行谓词计算的侯选结果。对 *v* 的一个候选结果而言，缓冲区的提升操作分为 2 种：一个缓冲区内部相邻层之间的提升；缓冲树中父子缓冲区之间的提升。

2.4 变量表及原子表

各个结果 XPath 式的缓冲区保存了对应的结果变量的绑定序列，为便于对这些数据进行后续处理和优化，将它们保存为变量表。

定义 8 以单个变量作为属性，变量的绑定序列作为元组序列的表结构为变量表。

定义 9 原子表的属性由变量依赖树中的 $n(n>0)$ 个变量组成，每个原子表有且仅有一个 for 变量属性，其他属性列由直接依赖于该 for 变量的 let 变量组成，表的元组是由各变量的变量表连接而成。如果原子表 AT_i, AT_j 之间满足 $AT_i \longrightarrow AT_j$，则对于 AT_i 的每个元组 *tup*, AT_j 中都有一组元组 S_{tup} 与之存在上下文关系，称 S_{tup} 为原子表 AT_j 的一个分组，*tup* 和 S_{tup} 中的每一个元组之间存在上下文索引。

对变量表连接的算法(操作)称为 JIT 算法(操作)。一个 JIT 操作的输入是 E-XSIEQ 机查询获得的 for 变量及所有依赖该变量的 let 变量的绑定序列。当元素结束事件(endElement)到达时，E-XSIEQ 机可能收集到一些变量的绑定值，此时执行 JIT 操作可由变量间的依赖关系对这些绑定值作即时的连接，形成原子表的一个元组。当 E-XSIEQ 机查询结束后，就得到填充后的原子表。

以示例的 *xq* 脚本和图 4 中的 XML 文档为例，对于变量 *b*，JIT 操作的第一次触发是由$</b_1>$引起的，此时 E-XSIEQ 机已收集到变量 *b*, *t* 绑定到的元素 b_1, t_1。JIT 操作将 2 个变量的绑定值根据依赖关系连接后，得到原子表 AT_b 的一个元

组(b_1, t_1), </b_2>, </b_3>各触发一次 JIT 操作。而对于变量 p, a 而言，由于没有依赖于 p, a 的变量，因此得到的变量表就是原子表，连接过程如图 5。查询的最后结果为图 6 中的原子表 AT_b, AT_p, AT_a。

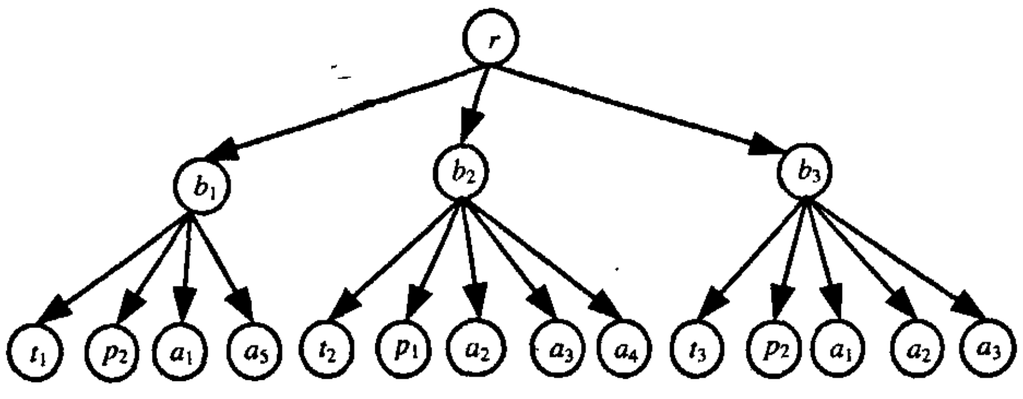


图 4　示例的 XML 文档示例

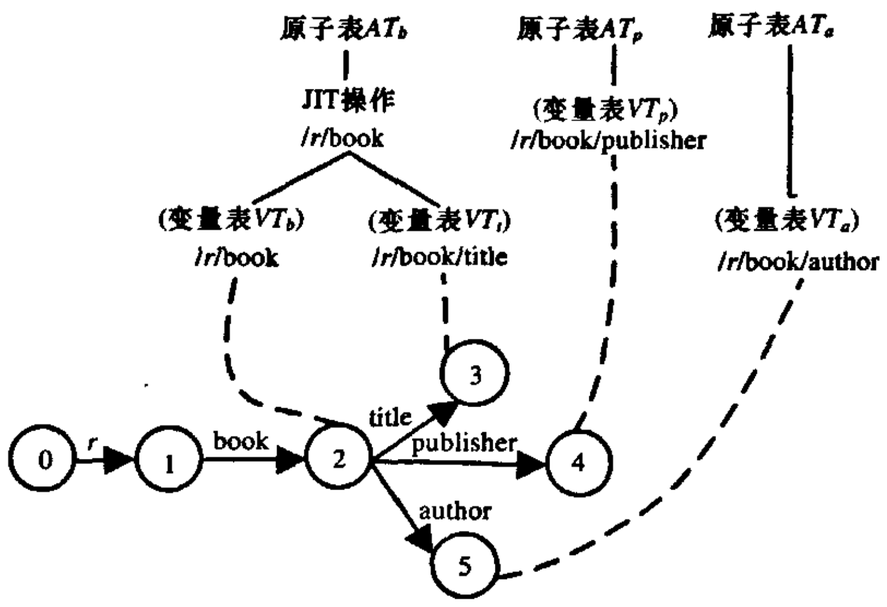


图 5　变量表连接与相应的自动机

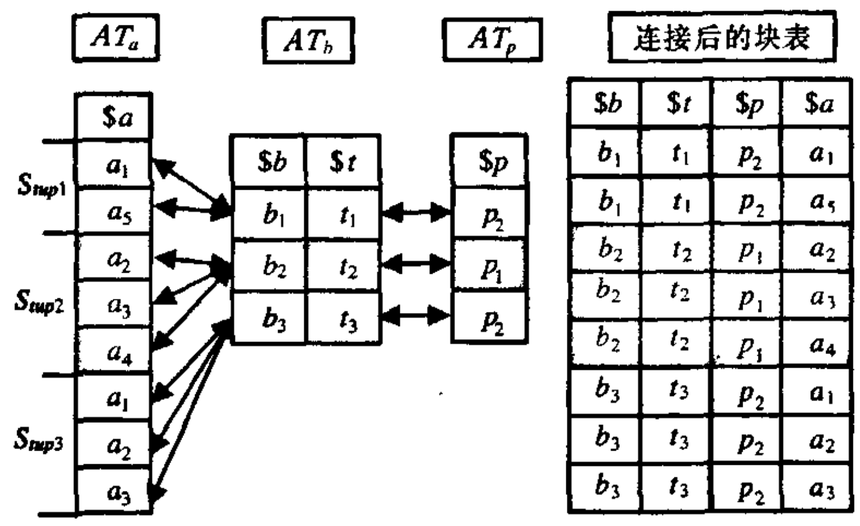


图 6　JAT 算法示例图

2.5　原子表的连接

定义 10 根据各个原子表中 for 变量属性间的依赖关系，连接一个 FLWOR 式中各结果变量所在的各个原子表得到用于构造结果的块表。

多个原子表的元组之间保存的双向索引(如图 6)为原子表间的连接形成块表提供了便利。下面以 2 个原子表的连接为例，描述原子表连接算法——JAT 算法。

算法 JAT 算法

输入：变量 a, b 对应的原子表 AT_a, AT_b

输出：原子表 AT_a, AT_b 连接后的块表 newBlockT

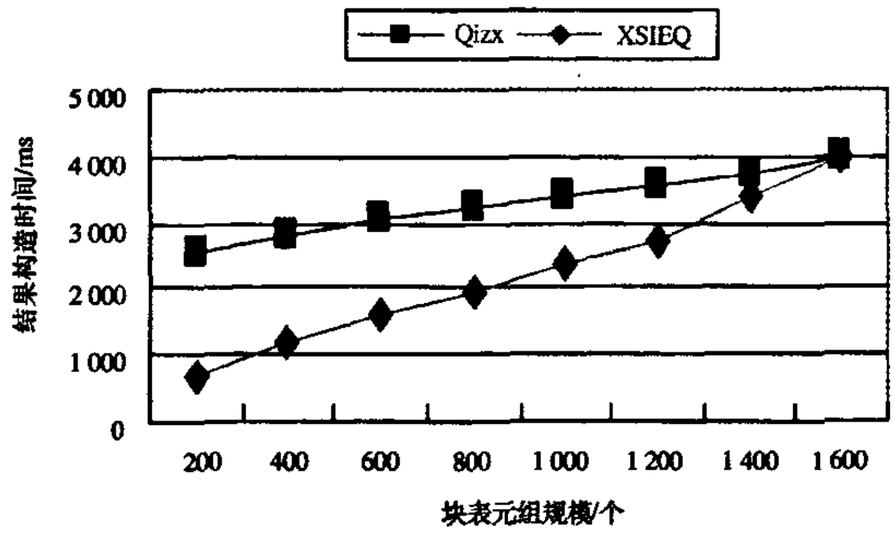
```
(1)  for each tupa in ATa {
(2)      if (a, b are not brothers)
(3)          Stup = getTups( tupa , ATb);
(4)      else{
(5)          ATc = getAncestor(a, b);
(6)          tupc =getTups(tupa, ATc );
(7)          Stup = getTups(tupc, ATb ); }
(8)      for each tupb in Stup {
(9)          blockTup = join( tupa , tupb );
(10)         newBlockT =appTup( blockTup ); }}
(11)   return   newBlockT; }
```

算法中用到的函数有：(1)getTups(*tup*, *X*)，取得元组 *tup* 在表 *X* 中的上下文分组；(2)getAncestor(*a*, *b*)，取得变量 *a* , *b* 的最近公共祖先变量所对应的原子表；(3)join(tup_a, tup_b)，将原子表元组 tup_b 连接到元组 tup_a 的后面，成为 *newBlockT* 的一个元组；(4)appTup(*blockTup*)，将连接后得到的块表元组 *blockTup* 插入到 *newBlockT* 中。

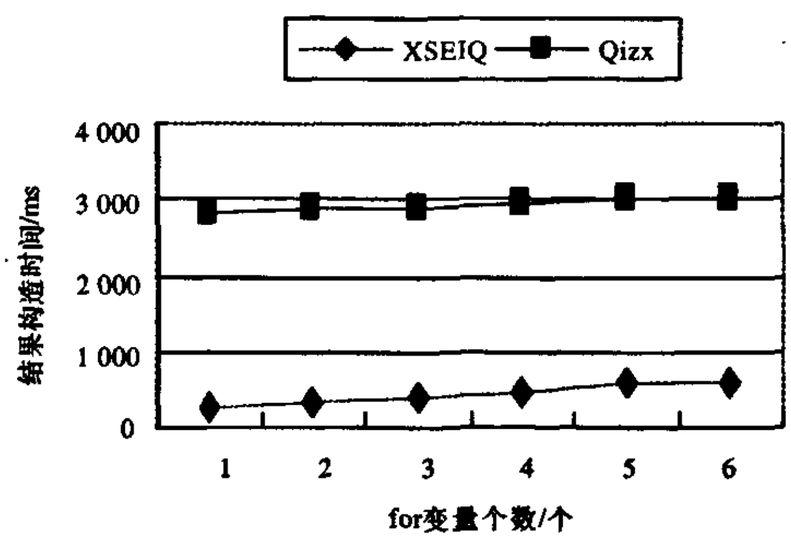
在变量依赖树中，当变量 *a*, *b* 的关系为祖孙关系(包括父子关系)或子孙与祖先的关系(包括孩子与父亲的关系)时，如算法第(2)步，对于原子表 AT_a 中的当前元组 tup_a，找到 tup_a 在表 AT_b 中的对应的分组 S_{tup}(当变量 *a*, *b* 是子孙与祖先关系时，分组 S_{tup} 内只有一个元组)；当变量 *a*, *b* 之间的关系为兄弟关系时，如算法第(4)步，对于原子表 AT_a 中的当前元组 tup_a，通过变量 *c* 绑定到的元组 tup_c 找到 tup_a 在表 AT_b 中的对应的分组 S_{tup}。最后将 tup_a 与 S_{tup} 分组内的每个元组进行连接，将连接后的元组作为 *newBlockT* 的一个元组。图 6 中原子表 AT_b, AT_p, AT_a 是示例的 *xq* 在图 4 文档上的查询结果，经连接 AT_b, AT_p, AT_a 后得到的块表见图 6。

3　实验结果与分析

笔者用 Java 实现了 XSIEQ-XQ 系统，并与 Qizx[2]在结果构造的时间性能上进行了对比测试。实验平台为 Windows XP 操作系统，CPU 为 P3，主频为 800 MHz，内存为 320 MB。实验用到的 XML 文件使用 Xmar[3]生成，XML 文档变化范围为 5 MB~7 MB，图 7 为部分实验结果。



(a)XSIEQ-XQ与Qizx结果构造性能的对比



(b)for变量个数对结果构造的影响

图 7　结果构造时间性能比较

实验 1 是在没有 where, order 子句且 return 子句返回的变量相同、XQuery 脚本中定义了 8 个 for 变量的情况下，块表元组规模不同时的测试结果。实验 2 是在块表元组规模为 200 个，且 return 子句返回的变量相同的情况下，XQuery 脚本中 for 变量个数不同时的测试结果。

从测试结果可以看出：

(1)XSIEQ-XQ 和 Qizx 的结果构造时间均随元组规模的增加而线性增长，当元组规模小于 1 600 个，XSIEQ-XQ 的结果构造时间比 Qizx 快。(下转第 70 页)

```
  FC ← {∅}; //置频繁封闭项集初值为空
for    (i = 1, F_n, ++)
   if    (h(F_i) = F_i)
      FC ← FC ∪ F_i; //若是封闭的，则并入
   endif
end
Call   GB;
Call   PB; }
```

4.2 计算复杂性

令最大频繁项集长度为 l，在一般的重述规则挖掘方法中，每个频繁项集的子集都可能作为规则的前件。于是，生成规则的数量为[8]

$$\sum_{i=0}^{l}\binom{l}{i}\cdot 2^{l-1} \leqslant \sum_{i=0}^{l}\binom{l}{i}\cdot 2^{l} = 2^{l}\sum_{i=0}^{l}\binom{l}{i} = 2^{l}\cdot 2^{l} = O(2^{2l})$$

由于运用了封闭项集，因此生成的重述规则是非冗余的。下面从两个极端情况讨论本文方法的计算复杂性。

(1)在最好的情况下，生成的重述规则全是精确的。仅有一个长度为 l 的封闭项，即所有 2^l 子集有与最大频繁项集相同的支持度。因此，所有项集之间的规则置信度是 100%。封闭项集方法对于单个项集产生最一般的规则，并且这些规则是在所有单个项之间产生。这种情况可能生成置信度是 100% 的规则数为 $l\cdot(l-1)=l^2-l$。因此，相应规则优化的效率是 $O(2^{2l}/l^2)$。

(2)最坏的情况是从项集的子集到它们的超集生成的规则可能全部无 100%的置信度，这时所有 2^l 频繁项集都是封闭的。对于长度为 k 的每个子集，从它的每个 $k-1$ 长度的子集到它所在的集合，可以生成 k 条规则。于是，在这种情况下生成规则的总数是

$$\sum_{i=0}^{l}\binom{l}{i}\cdot(l-i) \leqslant \sum_{i=0}^{l}\binom{l}{i}\cdot l = O(l\cdot 2^{l})$$

因此，相应规则优化的效率是 $O(2^l/l)$。

5 结束语

本文以 FCA 和 Galois 联络为理论基础，通过 Galois 联络的闭包运算，由封闭描述符集提出了重述数据集规则挖掘的生成子概念，并利用生成子构造无条件重述规则的生成基 GB 和条件重述规则的本征基 PB 算法，以及重述规则挖掘的 NRRM 算法。NRRM 算法具有如下特点：它可以提供最大信息量、最小非冗余的重述规则；NRRM 算法挖掘的是数据集全部蕴藏规则的子集，但它却不丢失任何信息，所有被有效重述规则覆盖的信息，NRRM 算法也都覆盖；特别是当数据集稠密的情况下，与其他算法比较，NRRM 算法则可以不生成大量的冗余规则。

如何将 FCA 应用于近似数据库、多值数据库和时态数据库的知识发现，使得 FCA 的应用从定常信息系统扩展到非定常信息系统的数据挖掘是今后值得进一步研究的问题之一。

参考文献

[1] 叶依群. 阅读理解中的重述现象探析[J]. 浙江科技学院学报, 2003, 15(1): 48-50.

[2] Ramakrishnan N, Kumar D, Mishra B, et al. Turning CARTwheels: An Alternating Algorithm for Mining Redescriptions[C]//Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: [s. n.], 2004: 266-275.

[3] Zaki M J, Ramakrishnan N. Reasoning About Sets Using Redescription Mining[C]//Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. New York: [s. n.], 2005: 364-373.

[4] Ganter B A, Wille R. Formal Concept Analysis: Mathematical Foundations[M]. [S. l.]: Springer-Verlag, 1999: 200-225.

[5] Jaoua A. Galois Connection, Formal Concepts and Galois Lattice in Real Relations: Application in a Real Classifier[J]. Journal of System and Software, 2002, 60(2): 149-163.

[6] Zaki M J. Generating Non-redundant Association Rules[C]//Proc. of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: [s. n.], 2000: 34-43.

[7] Zaki M J. Mining Non-redundant Association Rules[C]//Proc. of the 6th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: [s. n.], 2004: 223-248.

[8] Pasquier N, Bastide Y, Taouil R, et al. Efficient Mining of Association Rules Using Closed Itemset Lattices[J]. Journal of Information Systems, 1999, 24(1): 25-46.

(上接第 65 页)

(2)在 return 子句返回的变量相同时，结果构造时间随 for 变量个数的增加而增加。这是因为 for 变量与系统中的原子表是一一对应关系，for 变量个数增加，所需要构造的原子表个数增加，在原子表连接时所需要的时间则会增加。

(3)从图 7 中也可以看出，XSIEQ-XQ 的结果构造时间对元组规模、for 变量的个数的变化更加敏感，这是因为 for 变量的个数增多，原子表个数增多，连接的代价相应增大。而 Qizx 在内存中创建 DOM 树后进行查询，以空间的代价来获得稳定的性能。

4 结束语

XSIEQ-XQ 扩展了基本 XSIEQ 机，提出了有效的结构化连接算法，保证了中间结果的收集和结果构造较稳定的性能。另外，随着元组规模和 for 变量的增加，XSIEQ-XQ 的结果构造时间比 Qizx 增加得更快。如何获得更稳定的结果构造性能、减少元组规模以及 for 变量个数变化的影响将是下一步工作的重点。

参考文献

[1] 张　昱, 吴　年. 一种逐层提升缓冲的 XML 流查询自动机[J]. 小型微型计算机系统, 2007, 28(3): 456-461.

[2] Qizx 1.1p2 Released[Z]. (2006-09-03). http://www.axyana.com/qizxopen/.

[3] Schmidt A, Waas F, Kersten M, et al. XMark: A Benchmark for XML Data Management[C]//Proc. of VLDB'02. Hong Kong, China: [s. n.], 2002.