一种基于 DTD 的 XPath 逻辑优化方法*

高 军+, 杨冬青, 唐世渭, 王腾蛟

(北京大学 信息科学技术学院,北京 100871)

XPath Logical Optimization Based on DTD

GAO Jun+, YANG Dong-Qing, TANG Shi-Wei, WANG Teng-Jiao

(School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62765825, E-mail: gaojun@db.pku.edu.cn, http://www.pku.edu.cn

Received 2003-09-12; Accepted 2004-03-02

Gao J, Yang DQ, Tang SW, Wang TJ. XPath logical optimization based on DTD. *Journal of Software*, 2004,15(12):1860~1868.

http://www.jos.org.cn/1000-9825/15/1860.htm

Abstract: XPath becomes the basic mechanism for XML query. The non-deterministic operators in XPath, such as '//' denoting ancestor-descendant relationship and '*' denoting wildcards in XPath, greatly enhance the flexibility of XPath, but at the same time, introduce the complexity in XPath evaluation. How to explore DTD to reduce non-deterministic operators in XPath in order to improve the efficiency of XPath processing becomes a fundamental problem. The existing work focus on the limited fragment of XPath or DTD. This paper employs tree automata to express XPath and DTD in a unified framework, proposes a novel production operation on tree automata for XPath and tree automata for DTD, proves that the result of production equals to the optimized form of XPath in the presence of DTD, and generates the optimized XPath in a polynomial time based on the generation cost. The experimental result demonstrate that logical optimization on XPath can lead to the increase of efficiency on the existing XPath evaluator.

Key words: XPath; DTD; tree automata; rewrite; optimization

摘 要: XPath 成为 XML 数据查询的基本机制.XPath 中表达节点之间的祖孙关系的'//'和任意匹配字符的'*'等非确定操作符,增强了 XPath 表达方式的灵活性,但同时引入了 XPath 处理的复杂性.如何利用 DTD 减少 XPath 中的不确定操作符,从而提高 XPath 的执行效率成为一个基本的研究问题.传统方法主要侧重于特定受限 XPath 的确定化重写.利用树自动机在一个框架中表达 XPath 和 DTD,提出了一种新的 XPath 树自动机和 DTD 树自动机的乘积运算,并证明了乘积的结果就是基于 DTD 的 XPath 优化形式,在多项式时间内基于代价获取了 XPath

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA4Z3440 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032705 (国家重点基础研究发展规划(973))

作者简介: 高军(1975－),男,山东临沂人,博士,讲师,主要研究领域为数据库与信息系统;杨冬青(1945－),女,教授,博士生导师,主要研究领域为数据库与信息系统;唐世渭(1939－),男,教授,博士生导师,主要研究领域为数据库与信息系统;王腾蛟(1973－),男,博士,讲师,主要研究领域为数据库与信息系统.

的优化结果.实验数据表明,基于提出的 XPath 的逻辑优化方法,能够有效地提高目前 XPath 执行器的执行效率.

关键词: XPath;DTD;树自动机;重写;优化

中图法分类号: TP393 **文献标识码**: A

随着 Internet 技术的发展,XML 目前成为 Internet 环境中数据表示和交换事实的标准.XML 本质上可以看作点标记的有向树.XPath 是目前 XML 树中的路径导航语言[1],支持表达路径中结点的父子关系、祖孙关系、任意匹配结点、路径的选择关系等.

由于树模型比关系模型复杂,成熟的传统关系查询优化理论不能直接用来优化 XPath 查询,而 XPath 本身的优化技术远未成熟,如何提高 XPath 的查询效率成为目前迫切需要解决的问题.我们初步分析了 XPath 的执行过程[2].如果我们把 XML 看作是点标记的有向图,则 XPath 的执行相当于 XPath 所对应路径在数据图中的匹配.以自上而下的匹配策略为例,查询处理从 XML 数据树根节点出发,找到 XPath 路径中下一个元素,这样递归执行,直到 XPath 路径处理完毕为止.如果在 XPath 定义中存在表达节点之间祖孙关系的“//”和任意匹配结点的“*”,则在自上而下的匹配执行过程中,可能出现庞大的候选搜索路径,导致 XPath 处理效率的下降.

这个问题实际上也出现在 XPath 处理的其他领域.例如,基于关系数据库所定义的 XML 视图之上的 XPath 查询处理,需要根据查询父节点和子节点的关系,将 XPath 查询转换为关系查询.如果在 XPath 中出现了“//”和“*”,则很难确定 XPath 路径所对应的关系表.

目前 DTD 已经成为 XML 结构约束事实的标准,本文所讨论的问题可以描述为:如何基于 DTD 重写 XPath,从而减少路径匹配过程中的不确定因素,完成逻辑优化,提高 XPath 查询处理的效率,这个问题是目前的一个研究热点[3,4].

1. 相关工作

Dan Suciu 提出了如何在图模式中优化路径查询的方法[3],他利用传统自动机表达不包含“[]”的路径查询和图模式,通过路径自动机和图模式自动机的乘积,减少路径查询中的不确定因素,提高路径查询效率.但是,这种方法处理能力有限,不支持 DTD 或包含“[]”的路径查询.

Georg Gottlob 分析了目前 XPath 执行器的执行流程[2],提出了一种优化的自底向上的物理执行策略,提高了 XPath 的执行效率.本文将研究如何利用 DTD 等价变换 XPath 查询,实现 XPath 的逻辑优化.

S.Amer-Yahia 提出了简化 XPath 的方法[4],主要完成了将包含大量节点的查询转换为包含少量节点的等价查询.Peter Wood 利用 DTD 完成了 XPath 表达式的重写简化转换[5],他基于 DTD 产生了若干规则,利用规则等价重写 XPath,并且证明了算法在特定 DTD 和 XPath 之上是完备的.本文的主要目的是减少导致查询处理不确定的“//”和“*”因素,最终产生的优化 XPath 在形式上可能更加复杂.

Frank Neven[6]给出了支持不同特性的 XPath 在 DTD 下包含判定的复杂性.XPath 和 DTD 利用树自动机来表达,获取并解释树自动机之上基本运算的结果.本文算法基于同样的理论基础,但是本文主要目的是利用 DTD 获取等价的 XPath,而不是讨论 XPath 的查询包含问题.Michael Benedikt[7]分析了支持不同特性的 XPath 的查询表达能力,同时基于树模式和 XPath 的逻辑语义,给出了 XPath 在常规操作,如交、并、取反操作下的闭包性.本文主要讨论了本文自定义的树自动机乘积的性质.

2. 本文的工作

本文研究如何完备地利用 DTD 中的结构约束信息,来重写 XPath,减少 XPath 中的非确定操作符号.

1) 提出了 XPath 树自动机(ranked)和 DTD 树自动机(unranked)的乘积运算,证明了自动机乘积在本文环境中的正确性;

2) 在多项式时间范围内,基于 XPath 树自动机和 DTD 树自动机的乘积,获取 XPath 等价的重写形式;

3) 实现了本文的算法,通过实验,证明了本文 XPath 查询逻辑优化算法能够提高目前 XPath 处理器的效率.

本文第 1 节介绍背景知识,包括 XPath 特性定义、树自动机和 XPath 的树模式.第 2 节介绍本文所提出的基于 DTD 的 XPath 优化重写的方法,并且分析了算法的正确性.第 3 节给出了算法的实验数据,证明算法的有效性.第 4 节总结了全文,并对将来的工作进行了展望.

1 背 景

1.1 XPath

XML 可以看作是点标记的有向树.XPath 作为 XML 树中导航查询的基本机制,支持丰富的路径查询特性,但是,由于很少应用涉及到所有的特性,所以传统的讨论都是集中于部分常用的特性[6~8].本文讨论的 XPath 通过下列语法定义:

P ::=/E | //E

E ::=label | text() | * | @* | E/E | E//E |E [Q]

Q ::=E | E *Operl* Const | Q and Q

Oprel::= < | ≤ | > | ≥ | = | ≠

非形式化地,我们给出 XPath 中出现的操作符号的语义.给定 XML 数据树,"/"表示数据节点之间的父子关系,"//"表示节点之间的祖孙关系,"[]"表示路径之间的条件关系,"@"表示 XML 元素的属性,"*"表示任意的数据元素.另外,支持在路径表达式中定义逻辑表达式,包括"< | ≤ | > | ≥ | = | ≠".

1.2 树自动机

树自动机是传统字符自动机的扩展[8],两种自动机的根本区别在于状态转移规则的不同.在字符自动机中,状态转移前后最多只能存在一个状态,而树自动机中转换前或转换后支持状态集合,这种特性使得树自动机能够更加自然地处理树状数据结构.

树自动机支持两种执行策略,自上而下和自下而上.以自下而上树自动机为例,给出树自动机的定义.树自动机通过 4 元组 $T=(Q,\Sigma,\delta,F)$来定义,其中,Σ是字母集合,Q 是有限状态的集合,$F\subseteq Q$ 是终止状态集合,状态转移函数δ用 $2^{Q^*}\times\Sigma\to Q$ 来表示,也可以通过 $f(q_1,\ldots,q_n)-a\to q$ 来表示,其中 f 表示状态的正则表达式,$q\in Q,q_1\in Q,\ldots,q_n\in Q,a\in\Sigma$.

根据树自动机理论,DTD 结构约束可以利用树自动机来表达.DTD 对应的树自动机所产生的语言,实际上对应了满足 DTD 约束的 XML 文档.同样,XPath 也可以通过树自动机来表示,XPath 对应的树自动机所产生的语言,描述了所有包含 XPath 查询路径的 XML 文档.

以一个简单的例子说明 XPath 对应的树自动机,给定 XPath 表达式//a[b=1][c>2],我们构造树自动机(Q,Σ,δ,F),其中 $Q=\{s_1,s_2,s_3,s_5\},\Sigma=\{a,b,c\},\delta=\{\{\}-b\to s_5,\{\}-c\to s_3,s_3s_5-a\to s_2,s_2-all\to s_1,s_1-all\to s_1\},F=\{s_1\}$.这个自动机接受所有包含 XPath 所描述路径的 XML 文档.

DTD 所对应的树自动机和 XPath 所对应的树自动机在表达能力上是有差别的.由于 DTD 中元素通过子元素正则表达式来定义,所以,DTD 树自动机转换前的状态数目可能是无限的,这种树自动机被称为 unranked 树自动机.而在 XPath 所对应的树自动机中,转换前状态个数是确定的,这种自动机称为 ranked 树自动机[9,10].

1.3 树模式

Dan Suciu 引入了 XPath 的树模式概念[7].给定一个支持{/,//,*,[]}的 XPath,可以构建一个等价的树模式.反之亦然.我们利用一个例子来简要说明 XPath 树模式.

例1:给定 XPath 的查询 $p_1=a/b//c[d][*/e]$,我们建立与之对应的树模式,如图 1 所示.

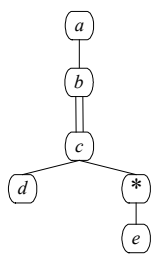


Fig.1 The sample of XPath tree pattern

图 1 XPath 树模式的示例

非形式化地,XPath 中的“[]”操作符号对应于树模式中的分支.树模式中边的标记“//”和“/”,分别描述节点之间的祖先关系和父子关系.

给定一个 XPath 查询,我们可以构造树模式和树自动机,两者从不同的角度等价描述了 XPath 的性质.树模式和树自动机之间也存在关联.根据树模式来构造树自动机是简单而直接的.

2 基于 DTD 的 XPath 重写优化

前面我们简要分析了 XPath 的执行过程,XPath 中出现的“*”(任意匹配字符)和“//”路径操作符号将导致路径匹配过程中的不确定性,在执行过程中产生大量回溯.由于目前 XML 大多遵从 DTD 约束,我们考虑如何利用 DTD 中的结构约束信息来减少 XPath 中的不确定因素,提高查询执行效率.

我们给出本文所要解决问题的描述:给定支持{//,*,/,[]}特性的 XPath 查询 p_1 和 XML 文档的 DTD,利用 DTD 重写 p_1 得到 XPath 查询 p_2,其中 p_2 包含的{//,*}尽可能少,同时,p_2 的执行代价要低于 p_1 的执行代价,p_1 和 p_2 在满足 DTD 的 XML 文档之上的查询结果等价.

2.1 XPath树自动机和DTD树自动机的乘积

DTD 所对应的树自动机是 unranked 树自动机,XPath 所对应的树自动机是 ranked 自动机.由于 DTD 所对应的树自动机规则中包含正则表达式,而 XPath 所对应的树自动机规则中转换前的状态集合是固定的,在给出 DTD 树自动机和 XPath 树自动机的乘积之前,我们首先给出字符包含定义.

定义 1(字符包含). 给定字符串 S_1 和字符串正则表达式 E,如果对于 E 表达式所产生的任何语言 S_2,都能建立从 S_1 到 S_2 的等价映射,使得对于 S_1 中的每个字符,都能够映射到 S_2 中相同的字符,则我们称 S_1 字符包含于字符正则表达式 E.

定义 2(XPath 树自动机和 DTD 树自动机的乘积). 给定 XPath 树自动机 $PT=(Q_1,\Sigma_1,\delta_1,F_1)$和 DTD 树自动机 $DT=(Q_2,\Sigma_2,\delta_2,F_2)$,则我们构造自动机 T_3 如下:$Q_3=Q_1\times Q_2$,$F_3=F_1\times F_2$,$\delta_3=\delta_1\times\delta_2$,其中,$\delta_1\times\delta_2=\{f((q_{11},q_{21})\ldots(q_{1n},q_{2n}))\to r_a\to(q_1,q_2)|(f(q_{11},\ldots,q_{1n})\to q_a\to q_1)\in\delta_1$,存在$(f(q_{21'},\ldots,q_{2m'})\to d_a\to q_2)\in\delta_2, f(q_{21},\ldots,q_{2n})$字符包含于$f(q_{21'},\ldots,q_{2m'})$中,$r_a=q_a\cap d_a,\{(q_{11},q_{21})\ldots(q_{1n},q_{2n})\}\subseteq Q_3\}$.我们利用 $PT\times DT$ 来表明 PT 和 DT 的乘积操作.

直观上,我们可以枚举所有的状态组合,构成新的状态集合,检查新的状态之间是否存在状态转换.这种算法是可行的,但是将产生大量的无效状态.根据规则,只有能够响应同一输入的规则才能进行乘积操作,下面我们给出自下而上的计算 XPath 自动机和 DTD 树自动机乘积的算法.

算法. XPath 树自动机和 DTD 树自动机的乘积.

输入:支持{/,//,*,[]}XPath 查询,DTD.

输出:XPath 树自动机和 DTD 树自动机的乘积.

根据 XPath 定义,构造 XPath 所对应的树自动机 $PT=(Q_p,\Sigma_p,\delta_p,F_p)$;

根据 DTD 定义,构造 DTD 对应的树自动机 $DT=(Q_d,\Sigma_d,\delta_d,F_d)$;

初始化 $T=(Q,\Sigma,\delta,F)$为空树自动机;

//开始自下而上处理 PT,获取 PT 和 DT 的乘积

构造初始的状态组合 $Q=\{(q_p,\varepsilon)|q_p\in Q_p\}\cup\{(\varepsilon,q_d)|q_d\in Q_d\}$;

$F=\{(f_p,f_d)|f_p\in F_p,f_d\in F_d\}$

Repeat

检查是否存在如下转换规则,$\{f((q_{11},q_{21})\ldots(q_{1n},q_{2n}))\to r_a\to(q_1,q_2)|(f(q_{11},\ldots,q_{1n})\to q_a\to q_1)\in\delta_1$,存在$(f(q_{21'},\ldots,q_{2m})\to d_a\to q_2)\in\delta_2, f(q_{21},\ldots,q_{2n})$字符包含于$f(q_{21'},\ldots,q_{2m'})$中,$r_a=q_a\cap d_a$,并且,$\{(q_{11},q_{21})\ldots(q_{1n},q_{2n})\}\subseteq Q\}$;

如果存在,则$\delta_d=\delta_d\cup\{(f((q_{11},q_{21})\ldots(q_{1n},q_{2n}))\to a\to(q_1,q_2)),Q=Q\cup\{(q_1,q_2)\},\Sigma=\Sigma\cup\{a\}$;

Until no such rules can be found

我们分析 XPath 树自动机和 DTD 树自动机乘积的性质.首先给出树自动机特定状态的定义.

定义 3(树自动机中关键状态). 给定树自动机 $T=(Q,\Sigma,\delta,F)$,如果存在状态转换规则 $f(q_1,\ldots,q_n)\to a\to q$,其中

$q_1\in Q,\ldots,q_n\in Q,a\in\Sigma,|f(q_1,\ldots,q_n)|$定义了转换前状态集合中的状态个数,$|f(q_1,\ldots,q_n)|>1$,或者$q\in F$,或者$\{\}-a\rightarrow q$,则状态$q$被称为关键状态.

根据 XPath 树自动机和 DTD 树自动机乘积运算,则乘积的性质分析如下:

(1) XPath 树自动机和 DTD 树自动机乘积中的关键状态(p,q),必定对应 Xpath 所对应的树自动机中的关键状态(p).

(2) 由于树自动机乘积中转换规则的转换前状态数目等于 XPath 树自动机的规则转换前状态数目,所以,XPath 树自动机和 DTD 树自动机的乘积一定是 ranked 自动机.

(3) 由于在 DTD 树自动机转换规则定义中,所有的输入字母都是确定的,即在 DTD 所生成的转换规则中d_a是确定的,并且由于$r_a=q_a\cap d_a$.所以,乘积中转换规则的输入字母r_a必定是确定的.

2.2 根据自动机乘积获取优化XPath查询

XPath 树自动机和 DTD 树自动机的乘积等价于基于 DTD 的 XPath 查询的优化形式(见第 2.3 节中的证明).但是,由于 DTD 和 XPath 可能存在递归,最终产生的 XPath 的优化形式的长度可能是指数级别的.我们需要根据重写结果,选择查询最终的输出形式.

定义 4(树模式中的关键点). 给定树模式$T=(N,E)$,如果存在$n\in N$,并且n的出度大于 1,或者n的入度=0,或者n的出度=0,则节点n被称为树模式中的关键点.

非形式化地,树模式中的关键点实际上是根节点、叶节点和分支节点.树模式中的关键点和 XPath 树自动机的关键状态是一一对应的.

定义 5(树模式中的路径段). 给定树模式$T=(N,E)$,$p=(n_1,\ldots,n_k)$,其中p是树模式中的路径,$n_i(1\le i\le k)\in N$,n_1是关键点,n_k是关键点,并且$n_i(2\le i\le k-1)\in N$不是关键点,则p被称为树模式中的路径段.

我们以路径段为优化的粒度,完成整个树模式的优化.

根据树模式路径段的两个端点所对应的 XPath 树自动机的关键状态k_1和k_2,获取 XPath 树自动机和 DTD 树自动机乘积中的关键状态集合 K_1 和 K_2,其中,K_1 集合中状态的形式是$(k_1,d),d\in Q_2$;K_2 集合中状态的形式是$(k_2,d),d\in Q_2$,Q_2是 DTD 树自动机的状态集合.XPath 树自动机中k_1和k_2关键状态之间的路径表达式等价于乘积自动机中所有s_1关键状态到s_2关键状态的路径表达式的 union,其中,$s_1\in K_1,s_2\in K_1$.由于路径段中不存在其他的关键点,s_1关键状态到s_2关键状态之间不存在其他的关键状态,s_1到s_2之间的表达式就是正则表达式.然而,如果关键状态之间存在状态转换圈,获取正则表达式的复杂性可能是呈指数级的.在这种情况下,原有树模式的路径段直接输出.这样,最终的查询优化结果的执行效率至少等价于原有查询效率.

算法. 基于 DTD 的 XPath 确定化重写和优化.

输入:XPath,DTD.

输出:优化的 XPath 查询.

根据 XPath 构造树自动机 PT,根据 DTD 构造树自动机 DT

构造乘积 $PT\times DT$;

对于 XPath 树模式路径段的端点所对应的两个关键状态k_1和k_2,在 XPath 树自动机和 DTD 树自动机的乘积中获取关键状态集合 K_1 和 K_2,其中,K_1 集合中状态的形式是$(k_1,d),d\in Q_2$;K_2 集合中状态的形式是$(k_2,d),d\in Q_2$,Q_2是树自动机 DT 的状态集合;

获取任意$p_1\in K_1$和$p_2\in K_2$之间的正则路径表达式,如果状态之间存在转换圈,则直接返回 XPath 树模式的路径段;否则,将所有的路径表达式 Union 起来,作为优化后的路径段表达式;

反复处理 XPath 的所有路径段.

下面我们给出一个例子,说明基于 DTD 的 XPath 逻辑查询重写.

例 2:给定 DTD 的定义 D:$\{a\rightarrow bc|p,c\rightarrow kp,k\rightarrow eac\}$,XPath 查询 $p_1=//p[k]$,$p_2=//p$,如何根据 DTD 来重写优化 p_1 和 p_2.

首先,分别构造 D, p_1 和 p_2 所对应的树自动机.

D 对应的树自动机 $T_1=(Q_1,\Sigma_1,\delta_1,F_1)$,其中,$Q_1=\{s_1,s_2,s_3,s_4,s_5,s_6\}$,$\Sigma_1=\{a,b,c,k,p,e\}$,$\delta_1=\{\{\}-e\rightarrow s_1,\{\}-a\rightarrow s_2,\{\}-c\rightarrow s_3,s_1s_2s_3-k\rightarrow s_4,\{\}\rightarrow k\rightarrow s_4,\{\}\rightarrow p\rightarrow s_5,s_4s_5-c\rightarrow s_3,\{\}-b\rightarrow s_6,s_6s_3-a\rightarrow s_2,s_5-a\rightarrow s_2\}$,$F_1=\{s_2\}$;

XPath 查询 p_1 对应的树自动机 $T_2=(Q_2,\Sigma_2,\delta_2,F_2)$,其中,$Q_2=\{m_1,m_2,m_3\}$,$\Sigma_2=\{p,k\}$,$\delta_2=\{\{\}-p\rightarrow m_1,\{\}-k\rightarrow m_2,m_1m_2-all\rightarrow m_3,m_3-all\rightarrow m_3\}$,$F_2=\{m_3\}$;

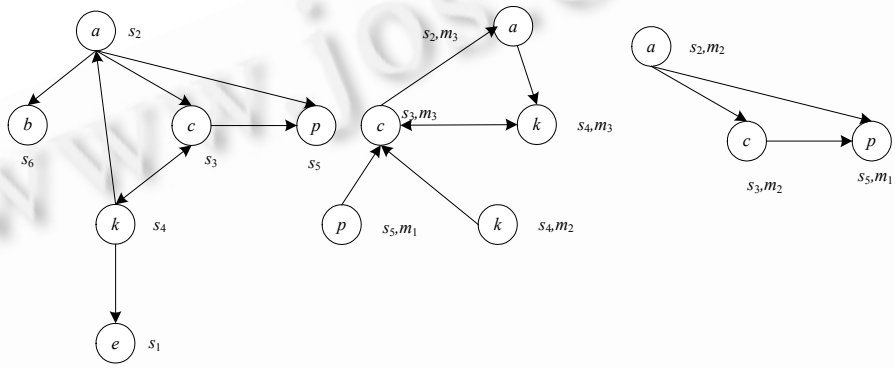
XPath 查询 p_2 对应的树自动机 $T_3=(Q_3,\Sigma_3,\delta_3,F_3)$,其中,$Q_3=\{m_1,m_2,m_3\}$,$\Sigma_2=\{p\}$,$\delta_2=\{\{\}-p\rightarrow m_1,m_1-all\rightarrow m_2,m_2-all\rightarrow m_2\}$,$F_3=\{m_2\}$;

其次,构造 DTD 对应树自动机和 XPath 对应树自动机的乘积:

构造 XPath p_1 和 DTD 的乘积 $T_4=(Q_4,\Sigma_4,\delta_4,F_4)$,其中,$Q_4=\{(s_5,m_1),(s_4,m_2),(s_3,m_3),(s_4,m_3),(s_2,m_3)\}$,$\Sigma_4=\{a,c,k,p\}$,$\delta_4=\{\{\}-p\rightarrow(s_5,m_1),\{\}-k\rightarrow(s_4,m_2),(s_5,m_1)(s_4,m_2)-c\rightarrow(s_3,m_3),(s_3,m_3)-k\rightarrow(s_4,m_3),(s_3,m_3)-a\rightarrow(s_2,m_3),(s_4,m_3)-c\rightarrow(s_3,m_3),(s_2,m_3)-k\rightarrow(s_4,m_3)\}$,$F_4=\{(s_2,m_3)\}$;

构造 XPath p_2 和 DTD 的乘积 $T_5=(Q_5,\Sigma_5,\delta_5,F_5)$,其中,$Q_5=\{(s_5,m_1),(s_3,m_2),(s_2,m_2)\}$,$\Sigma_5=\{p,c,a\}$,$\delta_5=\{\{\}-p\rightarrow(s_5,m_1),(s_5,m_1)-c\rightarrow(s_3,m_2),(s_3,m_2)-a\rightarrow(s_2,m_2),(s_5,m_1)-a\rightarrow(s_2,m_2)\}$,$F_5=\{(s_2,m_3)\}$.

图 2 说明乘积的过程.



(a) DTD tree automata (b) The production of tree automata for DTD and tree automata for XPath p_1 (c) The production of tree automata for DTD and tree automata for Xpath p_2

(a) DTD 树自动机 (b) DTD 所对应树自动机和 XPath p_1 对应树自动机的乘积 (c) DTD 对应树自动机和 XPath p_2 对应树自动机的乘积

Fig.2

图 2

最后,XPath 查询 p_1 和 DTD 乘积的关键状态之间存在状态圈,根据规则,原有的 XPath 树模式路径段直接输出,最终的优化形式为 p'_1=//p[k];而 XPath 查询 p_2 和 DTD 乘积的关键状态之间不存在状态圈,获得最终的优化查询 p'_2=/a/c/p | /a/p,其中,"|"表示路径的"或"操作.

2.3 算法分析

在上一节中,我们根据 XPath 树自动机和 DTD 树自动机乘积,获取了 XPath 的优化表达式.下面,我们分析算法的性质,讨论在哪些情况下,XPath 树自动机和 DTD 树自动机的乘积中关键状态之间不包括状态的转换圈.同时,分析算法的正确性和完备性.

定理 1. 如果 DTD 是非递归的,则 $PT\times DT$ 中不包括状态的转换圈.

证明:如果 DTD 是非递归的,则 DTD 对应的树自动机内部状态转变中不存在圈.假定 DTD 树自动机和 XPath 乘积的树自动机的关键状态之间存在圈,不妨假设为$(p_1,d_1)\ldots(p_1,d_1)$,则根据自动机乘积的规则,在 DTD 树自动机中,存在状态之间的转换圈 $d_1\ldots d_1$,所以假设错误. □

定理 2. 如果 XPath 支持{/,*,[]},而 DTD 支持递归,则 $PT\times DT$ 中不包含状态的转换圈.

XPath 中不支持"//"意味着 XPath 树自动机中不存在状态转换圈.证明过程类似定理 1.

定理 3. 自动机乘积正确性证明,给定 XPath 表达式 P,DTD 约束 D,P 对应的自动机为 PT,DTD 对应的自动机为 DT,对于 $PT\times DT$ 所接受的任意树结构 K,PT 接受 K.

证明:假定 $PT\times DT$ 接受树结构 K,则必然存在一个转换规则序列 T,使得在 K 的输入下自动机 $PT\times DT$ 能够到达终止状态.我们根据转换规则序列 T,获取新的转换规则序列 T',其中,对于 T 中出现的每一个状态转换序列 r,将 r 中的状态形式(p,d)改变为(p),则根据树自动机的乘积构造规则,在 PT 自动机中按照 T'的规则转换序列,能够到达终止状态,所以,PT 同样接受 K. □

定理 4. 自动机乘积完备性证明.给定 XPath 表达式 P,DTD 约束 D,P 对应的自动机为 PT,DTD 对应的自动机为 DT,对于 P 所接受的任意树结构 K,$PT\times DT$ 接受 K.

证明思路同定理 3 的证明,分别构造 DTD 自动机机中的状态转换序列和 XPath 自动机中的状态转化序列,通过自动机的乘积操作完成状态转换序列的组合,本文不再给出完整的证明.

3 算法数据分析

3.1 复杂性分析

算法首先获取了 XPath 树自动机和 DTD 树自动机的乘积:根据 XPath 和 DTD 构造树自动机,这个过程需要扫描一遍 XPath 定义和 DTD 定义;完成 DTD 树自动机和 XPath 树自动机的乘积,需要耗费多项式的时间代价.

其次,算法根据树自动机的乘积产生优化的 XPath 查询:算法的核心步骤是根据自动机的状态转换获取路径表达式.一般意义上,获取表达式的时间复杂性为指数级别,但是,本文算法检查了状态之间是否存在转换圈,在不存在转换圈的情况下,多项式时间产生了优化的 XPath 查询,否则直接输出原有的查询形式.所以,算法的第 2 个步骤也是多项式时间能够完成的.

3.2 实验数据

我们建立了基于 DTD 的 XPath 查询逻辑优化的测试环境,主要基于目前最流行的 XPath 执行器 Xalan[12],讨论了包含不同特性的 XPath 在重写优化前和重写优化后的执行效率.

我们利用 Java 1.31 完成 XPath 的重写优化,本实验中硬件环境为 Dell Optiplex 商用计算机,操作系统为 Windows 2000,CPU 为 P4,主频为 1.4G,主存为 512M,利用 Xmark[13]分别产生了 5M,10M,15M,20M,25M 的数据.由于支持不同特性的 XPath 中的执行性能不同,在本文 XPath 测试集合产生过程中,指定操作符{*}的概率 w 和{//}的概率 d 获取不同的 XPath 查询集合,每组集合包括 1 000 条 XPath.我们利用本文的算法重写优化这些查询,统计查询效率提高的比例.如果优化前的执行时间是 t_1,优化后的执行时间是 t_2,则本文查询效率提高比例定义为 t_1/t_2.我们基于的 DTD 分为两种,一种是支持递归的 DTD,另一种是不支持递归的 DTD.由于递归 DTD 对支持“*”的 XPath 作用类似于非递归 DTD 对支持“*”的 XPath 的作用(见定理 2),所以,本文主要讨论了基于递归 DTD 的支持“//”的 XPath 的效率提高比率.实验数据如图 3 所示.

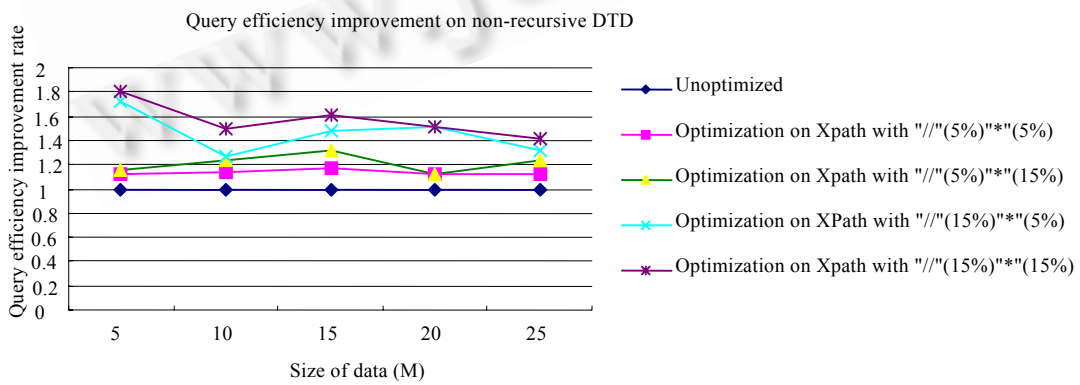


Fig.3 The effectiveness of optimization on XPaths with different {//,*} rates in the presence of non-recursive DTD

图 3 非递归 DTD 作用下不同{//,*}比率的 XPath 查询优化效果

从上述 4 组测试(*w* 和 *d* 分别设置为 15%和 5%)结果可以看出,基于 DTD 的 XPath 重写优化步骤能够有效地提高已有 XPath 执行器的执行效率.随着 XPath 中“*”和“//”的增加,优化后 XPath 效率提高更加明显.在 XPath 中,重写优化算法对于“//”的作用相对于“*”更加明显.这是由于“//”操作符号将导致目前查询处理器中更多的回溯.在实验过程中,我们获得最大的效率提高比率是 1.74.

根据定理 2,递归 DTD 主要影响 XPath 的“//”操作符号.图 4 主要讨论递归 DTD 之上包含“//”的 XPath 的优化后查询效率的提高.由于递归 DTD 树自动机和包含“//”的 XPath 树自动机的乘积中可能包含状态转换圈,所以在这些情况下,获取不包含非确定操作符的 XPath 表达式的复杂性可能是指数的,本文的方法是直接输出原有查询.所以,递归 DTD 之上查询优化的效率提升没有非递归 DTD 之上查询优化的效率提升得明显.

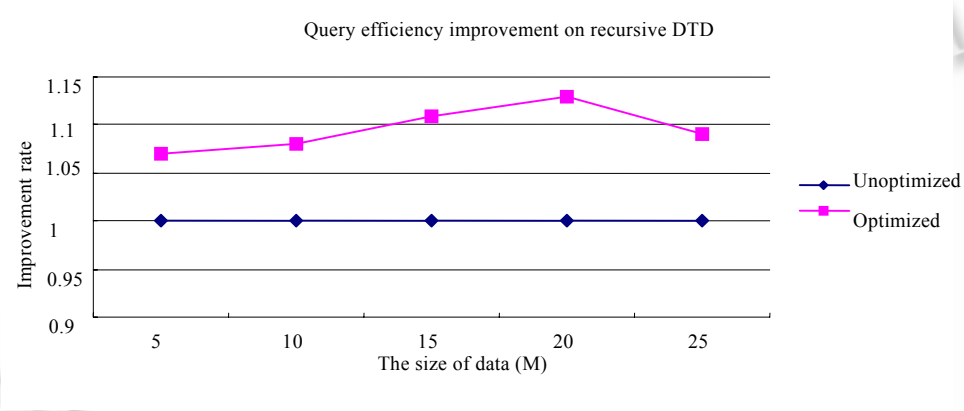


Fig.4 The effectiveness of optimization on XPath in the presence of recursive DTD

图 4 递归 DTD 作用下 XPath 查询优化效果

4 结束语

本文针对支持{//,*,/,[]}的复杂 XPath 查询执行的效率问题,将 DTD 引入逻辑优化过程,利用树自动机理论,提出了如何利用 DTD 来减少 XPath 查询中不确定因素,完成 XPath 逻辑优化的方法.未来的研究将侧重于递归 DTD 之上的逻辑查询优化.

References:

[1] Clark J. XML path language (XPath). 1999. http://www.w3.org/TR/XPath

[2] Georg G, Christoph K, Reinhard P. Efficient algorithms for processing XPath queries. In: Stéphane B, Akmal B. eds. Proc. of the VLDB 2002. Heidelberg: Springer-Verlag, 2002. 95~106.

[3] Mary F, Dan S. Optimizing regular path expressions using graph schemas. In: Proc. of the ICDE'98. Florida: IEEE Computer Society, 1998. 14~23. http://www.cs.washington.edu/homes/suciu/file33_paper.ps

[4] Sihem A, SungRan C, Laks VS. Minimization of tree pattern queries. In: Walid GA, eds. Proc. of the SIGMOD. Santa Barbara, 2001. http://www.research.att.com/~sihem/publications/SIGMOD01.pdf

[5] Peter W. Minimising simple XPath expressions. In: Giansalvatore M, Jérôme S, eds. Proc. of the WebDB. ACM, 2001. 13~18.

[6] Frank N, Thomas S. XPath containment in the presence of disjunction, DTDs, and variables. In: Diego C, Maurizio L, eds. Proc. of the ICDE. Heidelberg: Springer-Verlag, 2003. 315~329.

[7] Gerome M, Dan S. Containment and equivalence for an XPath fragment. In: Lucian P, eds. Proc. of the PODS. ACM, 2002. 65~76.

[8] Michael B, Wenfei F, Gabriel M. Structural properties of XPath fragment. In: Diego C, Maurizio L, eds. Proc. of the ICDE. Heidelberg: Springer, 2003. 79~95.

[9] Hubert C, Max D, Remi G, Florent J, Denis L, Sophie T, Marc T. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata/tata.pdf

[10] Frank N. Automata, logic, and XML. In: Julian C, ed. Proc. of the Int'l Workshop on Computer Science Logic. LNCS 2471,Heidelberg: Springer-Verlag, 2002. 2~26.

[11] Ashish K, Dan S. Stream processing of XPath queries with predicates. In: Alon YH, Zachary G, eds. Proc. of the SIGMOD. ACM, 2003. 419~430.

[12] Apache org. 2004. http://xml.apache.org/xalan-j

[13] Florian Waas. 2002. http://www.xml-benchmark.org