基于两阶段查询重写的 XML 近似查询算法

衡星辰, 覃 征, 邵利平, 曹玉辉, 高洪江
(西安交通大学电子与信息工程学院, 陕西西安 710049)

摘 要: 提出了基于两阶段查询重写的 XML 近似查询算法. 该算法不仅能够返回精确查询结果, 而且能够返回带有相似度分值的近似结果序列. 首先, 通过模式重写策略, 将原始查询树改写为多种 XML DTD(文档类型定义) 下的重写查询树, 从而解决了 XML 数据的多样性带来的查询语义缺失问题, 接着, 利用基本变异操作得到的变异查询树对 XML 数据树完成精确嵌入, 可将 XML 近似查询的问题转变为多棵变异查询树的精确查询问题, 并给出了基于 XML 数据统计的相似度计算模型和 Top-K 问题求解的优化算法. 最后, 在汽车外形智能化设计的实验中表明该算法优于 SSO 算法.

关键词: XML 近似查询; 基本变异操作; 变异查询树; 模式重写; 异质 XML 文档
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2007) 07-1271-08

Two-Phase Query Rewriting Based Approximate XML Query Algorithm

HENG Xing-chen, QIN Zheng, SHAO Li-ping, CAO Yu-hui, GAO Hong-jiang
(*School of Electronics and Information Engineering of Xi' an Jiaotong University, Xi' an, Shaanxi 710049, China*)

Abstract: A two-phase query rewriting based approximate XML (extensible markup language) query algorithm is proposed. The algorithm can not only return the exact answers, but also return the sequence of approximated answers with similarity degree. Firstly, through the strategy of scheme rewriting, an original query tree is rewritten into query trees with different XML DTDs (document type definition) so as to solve the problem of semantic loss due to the heterogeneous XML data. Secondly, the transformed query trees derived by the sequences of the basic transformation operations are used to perform exact embeddings into XML data tree, so as to transform the problem of approximate XML query into the problem of exact XML query for transformed query trees. Thirdly, XML data distribution statistics based similarity degree computing model and optimization algorithm for Top-K problem are given. Finally, the experiments of intelligent design of automobile shape show our algorithm outperforms the SSO algorithm.

Key words: approximate XML query; basic transformation operation; transformed query tree; schema rewriting; heterogeneous XML documents

1 引言

近年来, 随着 XML (extensible markup language) 的出现, 针对 XML 文档半结构化数据的查询算法的研究逐渐引起了国内外信息检索领域人们的关注[1~4]. XML 为 Internet 上的数据的描述和交换提供了一种事实上的标准, 作为一种通用格式来存储和交换来自于多种数据源的数据信息. XML 文档具有灵活的表达能力, 而这种灵活的表达能力使得不同组织和个人建立的 XML 文档, 难以遵照一个统一的数据模式, 即使内容完全相同的文档, 不同的组织和个人也很难按照统一的标准来建立结构和标识内容完全相同的 XML 文档, 从而造成了 XML 文档数据的多样性. 如何针对 WEB 上的大量的多样性的 XML 数据进行有效的信息提取是目前迫切需要解决的问题. 目前不同的组织和个人已经展开了不少的研究工作. 已有的 XML 近似查询算法的研究主要包括三个部分: 查询变异策略, 打分策略, 查询评估策略. 其中, 按照采用的查询评估策略的不同, XML 近似查询算法大体上可被分为三类: 基于查询重写策略的[5~8], 基于计划松弛策略的[9~11]和基于数据松弛策略的[12]等. 基于这些查询评估策略, 近年来提出的算法主要包括: 动态有序罚分算法(DPO)[8], 静态有序选择算法(SSO)[11], 加权树模式剪枝算法[9]等. 它们的基本思想是采用多种弱化方法对用户的原始查询树进行变异, 得到多棵和用户原始查询条件相似的变异查询树, 再利用这些变异查询树对 XML 数据库进行精确匹配, 将多次精确匹配后得到的带有变异代价值的查询结果作为近似查询的答案, 从而将 XML 近似查询的问题转变为

收稿日期: 2006-06-15; 修回日期: 2007-02-20
基金项目: 国家 973 重点基础研究发展规划(No. 2004CB719401)

多棵变异查询树的精确查询问题.但是,目前 XML 文档的查询方法大多关注对单一文档或遵循同一 DTD (document type definition)的多个 XML 文档的查询,随着 XML 数据库数据多样性的增加,查询服从多类 DTD 的 XML 文档逐渐成为查询需求的主流,文献[3]提出了一种 XML 查询中的 DTD 排序技术试图解决这个问题,但文中没有进一步给出排序后的 DTD 对用户的查询结构进行重构的方法以及结合该 DTD 排序技术的 XML 近似查询算法框架.其他现有的算法也未能对此给出较好的解决方案.另外,随着 XML 查询系统应用领域的扩展和后台 XML 数据库的扩大,XML 近似查询算法中基本变异操作代价的计算不能简单的依赖于领域专家的经验或者对 XML 样本数据的统计模型.因此,本文在 SSO 算法的基础上提出了基于两阶段查询重写的 XML 近似查询算法-TRXQ (two-phase rewriting based approximate XML query)来试图解决以上这些问题.

2 XML 查询形式化

定义 **1** 由 XML 文档按照一定的映射关系转化成的无向多叉树结构称为 XML 数据树,简称为数据树,它可用一个四元组 $D=(N, E, R, root(D))$ 来表示,其中 N 代表树中的结点集,E 代表边集,R 代表结构关系集,$root(D)$ 代表数据树的根结点.

本文考虑了 XML 文档集中元素和属性名的语义多样性的问题.假定文档集中的文档来源于不同的组织和机构,它们采用了不同的 DTD 进行了定义,这样对同样一个现实领域实体的描述可能在不同的 XML 文档中用了不同的标签或者结构关系.

定义 **2** 由 XML 查询语句转化成的无向多叉树结构称为查询模式树,简称为查询树,它可用一个四元组 $Q=(V, E, R, root(Q))$ 来表示,其中 V 代表查询树中的结点集,E 代表边集,R 代表结构关系集,$root(Q)$ 代表查询树的根结点.

查询树中的根结点指明了当前所要查询的范围,叶子结点描述了用户所要查询的具体信息,内部结点指明了具体信息所在的上下文.查询树中结点之间的边有两种类型:父子关系和子孙后代关系.

定义 **3** 给定一棵 XML 数据树 D,那么由 D 中的2个结点 v_i 和 v_j 以及存在于这2个结点之间的结构关系集 R 构成的一个二元谓词 $R(v_i, v_j)$ 被称为结构查询谓词,简称结构谓词或查询谓词.

一棵查询树可分解为多个查询谓词构成的集合.

根据已经引入的数据树和查询树的概念,关于 XML 数据查询的问题就可以被转化为一个查询树到数据树的嵌入问题,该嵌入又可分为精确嵌入和近似嵌入.精确嵌入[8]的定义如下:

定义 **4** 给定一棵查询树 Q,若在查询过程中,Q 中的所有结点 $v_i(v_i \in V)$ 满足如下四个条件,则一个从查询树 Q 到数据树 D 的映射函数 f 被称为一个精确嵌入.

(1) $v_i = v_j \Rightarrow f(v_i) = f(v_j)$;

(2) 结点 v_i 的标签和 $f(v_i)$ 的标签相同;

(3) 结点 v_i 的类型和 $f(v_i)$ 的类型相同;

(4) 查询树 Q 中所潜在的所有查询谓词 QP 在数据树 D 中都存在.

3 XML 近似查询

为了发现和用户设定的查询树近似匹配的答案,首先要对原始查询树进行查询变异.

定义 **5** 利用插入一个结点,删除一个结点或重命名一个结点标签等对一棵原始查询树 Q 作出的每种修改称为一个查询变异 QT.

下面,本文简要介绍查询变异 QT 过程中对原始查询树所常用到的几种基本查询变异操作.

3.1 基本查询变异操作

本文引入了文献[8]定义的几种常用的基本查询变异操作:插入结点操作,删除内部结点操作,删除叶子结点操作和重命名结点操作.本文对基本变异操作的使用制定了规则,不允许对原始查询树任意使用由基本变异操作构成的序列,目的是为了避免生成毫无用户语义要求的大量的"盲目"变异查询树而增加了查询算法的复杂性.本文从两个方面定义了基本变异操作的使用规则:

(1) 基于查询标记:当用户在设定查询树时,可以根据自己的查询需求对树中的查询结点设置特殊标记的方法来限制变异操作对这些结点的影响,例如不得对用户最感兴趣的结点进行重命名操作,就在该结点旁边加上"!"号.这种方法灵活性高,但用户需要具备一定的查询语法知识.

(2) 基于限制规则:这是对基本变异操作使用的通用规则,主要包括以下四种:

①不能为根结点插入父结点,或者对叶子结点插入子结点;

②在删除内部结点时不包括根结点,在删除叶子结点时,如果该叶子结点的父亲结点只有它一个儿子,那么该结点不允许被删除;

③在重命名结点时,新的标签必须是该结点的父亲结点或兄弟结点的标签;

④查询树中不能出现没有叶子结点的分枝,若有将该分枝自动删除.

3.2 TRXQ 近似查询算法

定义 **6** 由一组任意的基本变异操作构成的序列称为变异操作序 T.

定义 **7**　通过应用一个指定的变异操作序到原始查询树 Q 后得到的查询树称为变异查询树 Q'.

定义 **8**　对原始查询树执行基本变异操作后所带来的和用户原始查询需求之间的语义损失度称为基本变异操作代价, 它是大于 0 的实数.

定义 **9**　给定一棵原始查询树 Q 和一棵变异查询树 Q', Q' 由应用一个变异操作序 $(t_1, t_2, \cdots, t_n)$ 到 Q 后得到. 若用 $cost(t_i)$ 代表某个基本变异操作 t_i 的代价, 那么一棵变异查询树 Q' 的嵌入代价可被定义如下:

$$\text{MappingCost}(Q') = \sum_{i=1}^{n} \text{cost}(t_i) \qquad (1)$$

嵌入代价是衡量查询结果近似程度的标准, 嵌入代价越大则最终查询结果的近似程度越低.

定义 **10**　给定一棵原始查询树 Q, 若用 $Q'(\boldsymbol{T}_i)$ 代表应用一个变异操作序 $\boldsymbol{T}_i = (t_1, t_2, \cdots, t_n)$ 到 Q 后得到一个变异查询树, 那么查询闭包 Q^* 可被定义如下:

$$Q^* = \bigcup_{i=0}^{\infty} Q'(\boldsymbol{T}_i) \qquad (2)$$

其中, $Q'(\boldsymbol{T}_0) = Q$, $\text{MappingCost}(Q'(\boldsymbol{T}_{i+1})) > \text{MappingCost}(Q'(\boldsymbol{T}_i))$.

定义 **11**　给定一个数据树集合和一个查询闭包 Q^*, 若用一个二元组 $(D_{\min}, c)$ 来代表近似查询闭包中的一棵变异查询树的查询结果, 而 $D_{\min}$ 代表与该变异查询树满足精确嵌入的所有数据树中所需嵌入代价最小的数据树, c 代表 $D_{\min}$ 对应的嵌入代价, 那么对所有的二元组 $(D_{\min}, c)$ 的求解被称为一个近似嵌入.

从定义 11 可以看出一个近似查询的问题可以被转化为针对近似查询闭包的多次精确查询的问题来求解, 解决近似查询问题的算法必须能够发现近似查询闭包中所有变异查询树的精确嵌入结果. 又因为用户往往感兴趣的是最近似的 k 个查询结果, Top-K 问题被定义如下:

定义 **12**　给定一棵原始查询树 Q 和一个整数 K, 那么求解按嵌入代价从小到大排列的 K 个近似查询结果列 $(D, \text{MappingCost})$ 的问题被称为 Top-K 问题.

定义 **13**　若一个给定的 XML 文档集中的所有 XML 文档都按照同一种 DTD 来定义, 不同的 XML 文档只存在结构上的不同 (XML DTD 允许部分结构的缺失), 而在元素及其属性的命名上不存在异名同义的问题, 则称为异构同质 XML 文档集.

定义 **14**　若一个给定的 XML 文档集中的所有 XML 文档可按照不同的 DTD 来定义, 不同的 XML 文档不仅存在结构上的不同, 而在元素及其属性的命名上存在异名同义的问题, 则称为异构异质 XML 文档集.

定义 **15**　给定两种 XML DTD: S_1、S_2, 再给定一棵 S_1 定义下的原始查询树 Q, 若 Q 中的所有结点 $v_i (v_i \in V)$ 和 S_2 中的元素结点 $u_j (u_j \in U)$ 满足如下条件, 那么原始查询树 Q 中的结点集 V 到 S_2 定义下的查询树 Q' 的结点集 $V' (V' \subset U)$ 的一个映射 m 称为模式重写:

$$\text{Max}(\text{Sem\_Distance}(v_i, u_j)) = \text{Sem\_Distance}(v_i, m(v_i)) \quad (1 < i < N, 1 < j < M) \qquad (3)$$

其中, $m(v_i) \in V'$, $\text{Sem\_Distance}(v_i, u_j)$ 代表 S_1 和 S_2 中元素结点之间的语义距离.

定理 **1**　给定一个查询闭包 Q^*, 则按照 Q^* 所包含的变异查询树顺序查询得到的结果集在质量上是单调递减的, 在数量上是单调递增的.

证明　用 $\boldsymbol{T}$ 来代表变异查询树序列, $\boldsymbol{T} = (t_1, t_2, \cdots, t_n)$, 其中 t_i 代表变异查询树. 用 $\boldsymbol{R}$ 来代表查询结果序列, 用 $r^*(t_1)$ 代表 t_1 对应的查询结果集, 则 $\boldsymbol{R} = (r^*(t_1), r^*(t_2), \cdots, r^*(t_n))$. 由查询闭包的定义可知, 该序列是按照嵌入代价 c 由低到高来排列的, 即 $c(t_1) <= c(t_2) <= \cdots <= c(t_n)$. 又由嵌入代价的定义可知, 查询结果的相似度 (质量) 和其对应的近似查询树的嵌入代价成反比, 若用 $\text{Sim}()$ 代表近似查询树 t_i 最终对应的查询结果的相似度函数, 则有 $\text{Sim}(r^*(t_1)) >= \text{Sim}(r^*(t_2)) >= \cdots >= \text{Sim}(r^*(t_n))$, 固结果集在质量上是单调递减的.

由变异查询树的定义可知, 一个变异查询树对应着一组基本变异操作序, 若用 $A(t_i)$ 来代表变异查询树 t_i 对应的基本变异操作序, 那么有以下两种情况:

(1)　$A(t_i) = A(t_{i-1}) + \{a, b, c\}$, 其中 $\{a, b, c\}$ 代表三种基本变异操作可能构成的任意序列. 由于 t_i 在 t_{i-1} 基础上增加了新的基本变异操作, 显然因为这些新的基本变异操作扩大了 t_i 的查询的范围, 导致 t_i 的查询结果集更大, 固按照查询闭包中的变异操作序查询出的结果序列在数量上是单调递增的.

(2)　$A(t_i) \neq A(t_{i-1}) + \{a, b, c\}$, 即 t_i 没有对 t_{i-1} 的基本变异操作进行闭包. 由已证明的按照查询闭包中的变异操作序查询出的结果序列在质量上是单调递减可以推出 t_i 对应的变异代价大于 t_{i-1} 对应的变异代价. 又由于本文提出的是基于样本数据分布统计的方法来计算变异操作的代价, 所以 t_i 的变异代价越大, 它所应用的基本变异操作所涉及的查询谓词在后台 XML 数据库中出现的频率越高, t_i 中查询谓词的频倍越高, 从而导致 t_i 的查询结果集更大, 固按照查询闭包中的变异操作序查询出的结果序列在数量上是单调递增的.

故定理 1 由此得证.

至此, 本文提出的针对异构异质 XML 文档集的 TRXQ 算法的主要步骤如下:

(1) 求出原始查询树 Q 对应的 DTD 和 XML 数据库所涉及的其他 DTD 之间的语义距离 r 以及结点最佳映

射关系表;

(2) 利用(1) 步中得到的结点最佳映射关系表对原始查询树 Q 进行重写, 得到不同 DTD 定义下的重写查询树 Q_s;

(3) 将(1) 步中得到的语义距离 r 作为重写查询树 Q_s 的重写代价, 利用该重写代价 r 对不同 DTD 定义下的重写查询树 Q_s 从低到高进行排序, 形成 Q_s 队列;

(4) 若 Q_s 队列为空, 则终止查询, 否则从当前 Q_s 陷列中弹出重写代价 r 最低的 Q_s.

(5) 求出当前 Q_s 对应的查询闭包 Q_s^*, 并将 Q_s^* 中的变异查询树 Q_s^t 对应的嵌入代价 c 和 Q_s 对应的重写代价 r 的乘积 $c*r$ 作为变异查询树 Q_s^t 最终的查询代价 e;

(6) 对查询闭包 Q_s^* 中的所有的变异查询树 Q_s^t 按查询代价 e 从低到高进行排序;

(7) 如果查询闭包为空, 则返回(4), 否则从查询闭包中取出当前分值最低的 Q_s^t, 并将该 Q_s^t 从查询闭包中删除. 采用结构化连接算法[13] 对该 Q_s^t 和数据树集执行精确嵌入, 如果有满足精确嵌入的四个条件的查询结果, 就将其放入查询结果集中, 并将该变异查询树的查询代价 e 作为当前查询结果和原始查询树 Q 的匹配近似度;

(8) 如果查询结果集中的结果个数达到用户查询时指定返回结果数 K, 则终止查询, 否则返回(7) 继续执行.

4 XML 查询代价评估

XML 查询代价的正确评估, 是有效衡量查询结果近似度的标准, 其主要包括两个部分: 基于 XML DTD 的重写代价 r 和基于变异操作的嵌入代价 c.

4.1 重写代价

由于 XML 数据库的异质性, 增添到文档库中的 XML 文档并不会遵循同一种 DTD 来定义, 那么很有可能出现这种情况: 虽然新到来的 XML 文档所对应的 DTD 和查询默认的 DTD 在结构和对元素的描述词汇方面并不一致, 但是在语义上, 它们的确描述了非常接近的对象信息, 所以这些 XML 文档对于用户的查询也是很有用的. 为了查出所有的这些异质同义的文档, 需要利用这些 XML DTD 间隐含的相似信息来重写查询树. 基于文献[14] 中的部分思想, 本文设计的基于 XML DTD 的查询树重写算法的主要步骤如下:

(1) 用 RDF 规范, 将被计算的两个 XML DTD 转化成一个有向标识图[15], 接着, 利用该有向标识图构造出一个双连接图[16], 该图中包含了由这两个 DTD 树中的结点构成的结点对联结.

(2) 计算双连接图中结点对的初始相似度. 本文采用 in-depth 语义的方法来处理这个问题. 这里要利用一个中文同义词库, 通过比较两个结点的标签名在中文同义词库中的同义词分别所在的类层次的深度和连接两个同义词的最短路径的长度来量化两个结点间的相似度, 比如针对结点对 $(s1, s2)$ 的相似度计算公式如下:

$$Sim\_Distance(s1, s2) = 1 + (Min\_Path(s1, s2) / |Depth(s1) + Depth(s2)|) \quad (4)$$

(3) 初始相似度仅仅反映了单个结点对的语义距离, 它是片面的, 没有考虑结点在 DTD 树中的位置对它们产生的影响. 本文采用相似度传播算法[16] 对初始相似度进行了修正, 该方法在计算过程中考虑了 DTD 的结构信息, 其核心思想是属于不同 DTD 中的两个结点越相似, 那么它们的相邻结点也越相似, 也就是说 DTD 中两个元素的相似情况也影响到了它们各自的相邻结点.

(4) 统计针对每个结点的相似度最高的结点对作为该结点的最佳匹配, 以确保对于每一对结点 $(s1, s2)$ 来说, 不存在另外的结点对 $(x1, x2)$, 使得 $s1$ 更相似于 $x1$ 比 $s2$, 而 $x2$ 更相似于 $s1$ 比 $x1$.

(5) 利用得到的最佳匹配结点对对原始的查询树进行重写. 重写的过程是对原始查询树中从根结点出发的每条路径的重写构成, 并将基于路径的重写过程中所涉及的替换结点的相似度累加求和, 将其平均值作为该次重写所需的代价, 即重写代价 r.

(6) 统计源 DTD 和目标 DTD 中的所有最佳匹配结点对的相似度之和, 并除以目标 DTD 中所包含的结点数所得到的值作为这两个 DTD 间的相似度.

4.2 嵌入代价

根据定义 5 嵌入代价是应用到原始查询树上的所有变异操作的代价总和, 那么计算嵌入代价的问题就转化为计算基本变异操作代价的问题. 下面介绍基于样本数据分布统计和专家经验的基本变异操作代价的计算方法.

4.2.1 重命名代价

重命名代价指的是将一个结点的标记更新为该结点的父结点或者兄弟结点的标记所需要的代价. 例如要将结点 n_1 更替为它的祖先结点 n_2, 它的计算步骤如下:

(1) 根据 XML DTD 求出 $Min\_path(n_1, n_2)$: 结点 n_1 和 n_2 间的最短路径;

(2) 统计该条路径所经过的结点.

(3) 累加这些结点的权重作为重命名所需的代价. (路过"零"问题: 路径上没有其他结点, 则将结点 n_1 和 n_2 的权重差作为代价; 再次路过问题: 路径上出现同名的结点, 出现同名结点说明结点 n_1 和 n_2 分别处于同名的两个结点所代表的不同的语义上下文下, 此时代价定义为无穷大.)

如何定义 XML 文档中每个结点的权重是接下来需要解决的问题, 一个结点的权重和两个因素有关: (1)

所处的位置; (2) 出现的频率. 在 XML 文档树中所处的位置越高, 出现的频率越大, 则该结点的权重也越大, 因此可以给出如下的计算公式:

$$W(n_i) = \alpha * \beta_{naming} * \sum_{t=1}^{N} (G_t(n_i)/N) * (N/N_{normal}) \quad (5)$$

式(5)中 N 代表结点 n_i 在 XML 数据库中出现的次数, N_{normal} 是标准频数, 即预先统计得出的所有结点出现的平均次数. (N/N_{normal}) 表示结点 n_i 出现的相对频倍, 它的值越高, 说明结点 n_i 出现的频率越大, 从而导致它的权重也越大. α 为平衡因子, 可以用来调整结点和该结点构成的结构谓词之间的频度差异, β_{naming} 为重命名因子, 可以用来调整因为环境上下文引起的权重和重命名代价之间转换的等价度. $G_t(n_i)$ 是一个度量结点 n_i 在第 t 次出现时所在位置重要性的函数, 它的计算公式如下:

$$G_t(n_i) = L_{n_i}/L_{tree} \quad (6)$$

其中 L_{n_i} 是结点 n_i 在当前所在的 XML 数据树中所在的层数, 根结点位于最高层, L_{tree} 代表当前这棵数据树的总层数.

4.2.2 删除代价

由于一个结点的权重和该结点在文档库中出现的频率成正比, 那么该结点的权重越大, 那么删除该结点所需要的代价就越小. 设结点权重的上限值为 W_{max}, 那么删除一个结点 n_i 的代价为:

$$Cost_{del}(n_i) = \alpha * \beta_{del} * (W_{max} - w_i) \quad (7)$$

其中, w_i 为结点 n_i 的权重, α 为平衡因子, 可以用来调整结点和该结点构成的结构谓词之间的频度差异, β_{del} 为删除因子, 可以用来调整因为环境上下文引起的权重和删除代价之间转换的等价度. 特殊地, 当结点取上限值时, 它的删除代价就为 0.

4.2.3 插入代价

插入结点就相当于在查询树中增加新的结点. 和删除一个结点的代价的计算方法相反, 插入结点的代价和新增加的该结点权重的大小成正比, 也就是说该结点在文档集中出现的频率越大则插入它的代价也越大, 那么插入一个结点 n_i 的代价为:

$$Cost_{inse}(n_i) = \alpha * \beta_{inse} * w_i \quad (8)$$

其中, w_i 为结点 n_i 的权重, α 为平衡因子, 可以用来调整结点和该结点构成的结构谓词之间的频度差异, β_{inse} 为插入因子, 可以用来调整因为环境上下文引起的权重和插入代价之间转换的等价度.

5 Top-K 问题

5.1 查询闭包的 Top-K 顺序求解

根据定义 10 可知查询闭包是由原始查询树得到的所有变异查询树的集合, 在 TRXQ 近似查询算法步骤(6)中需要依据查询代价对查询闭包中所有的变异查询树从低到高进行排序. 但是要能进行正确有效的排序, 有两个问题需要解决: (1) 求出所有变异查询树; (2) 预先计算出每棵变异查询树的查询代价. 本文通过结合以下三个方法来解决这两个问题:

(a) 利用 3.1 小节给出的基本变异操作的限制规则, 来精简出切实有效的查询闭包;

(b) 对 XML 数据库中的 XML 数据树进行编码[8], 生成带有如下索引结构的数据树来解决盲目的结点插入操作带来复杂性问题:

index= (pre, bound, pathcost, inscost)

其中 pre 代表对数据树进行前序遍历时结点的序号, bound 是该结点的所包含的最右边的叶子结点的序号, pathcost 是该结点所有祖先结点的插入代价的总和, inscost 是该结点的插入代价. 利用该索引结构, 可以计算将任意查询谓词嵌入到数据树中所需的插入代价, 其计算公式如下:

$$\begin{aligned} InsertCost &= distance(n_1, n_2) \\ &= pathcost(n_2) - pathcost(n_1) - inscost(n_1) \end{aligned} \quad (9)$$

其中, n_1 和 n_2 是该查询谓词包含的两个结点, 而 n_1 是结点 n_2 的祖先. 该插入代价 insCost 是前文中提出的嵌入代价 c 的一部分.

(c) 对精确嵌入所需满足的第四个条件进行补充: 在查询树中指定为父子关系的查询谓词, 其在数据树中可以扩展为子孙后代关系.

至此, 查询闭包的 Top-K 顺序求解的步骤如下:

(1) 依据方法 a) 对原始查询树依次应用重命名、删除内部结点和删除叶子结点这三种变异操作(不包括插入操作)构成的变异操作序, 生成按变异操作代价由低到高排列的变异查询树构成的查询闭包, 此步得到的变异操作代价称为基本嵌入代价 $c_{naming,del}$.

(2) 如果查询闭包为空, 则终止查询, 否则从查询闭包中选出当前分值最低的变异查询树对数据树集进行精确嵌入. 按照方法(c), 在精确嵌入过程中, 在数据树中满足子孙后代关系的查询谓词(在查询树中被指定为父子关系)也将被保留, 并应用方法 b) 中定义的函数 distance( ) 来实时计算所有潜在的插入结点操作的代价 c_{insert}.

(3) 将(2)步中得到得插入代价和(1)步中得到该变异查询树的基本嵌入代价之和作为当前查询结果的近似度, 并插入到查询结果集的适当位置中去.

(4) 如果查询结果集中的结果个数达到用户查询时指定返回结果数 K, 则终止查询, 同时返回 K 个按近似度由高到低排列结果序列给查询用户, 否则返回步(2)继续执行.

5.2 最优变异查询树

定义 **16** 给定一棵原始查询树 Q 和其查询闭包 Q^*, 如果 Q^* 中的一棵变异查询树 Q' 满足如下条件, 那么该 Q' 称为最优变异查询树.

$$\mathrm{Min}(|\mathrm{Answer}(Q_i')-K|)=|\mathrm{Answer}(Q')-K| \quad (1<i<n) \quad (10)$$

其中 $\mathrm{Answer}(Q_i')$ 代表 Q_i' 对 XML 数据库执行一次精确嵌入后得到的查询结果个数, n 代表 Q^* 中变异查询树的总个数.

通过最优变异查询树, 本文对查询闭包的 Top-K 顺序求解过程的(2)步进行了优化, 同时, 按近似度值由高到低顺序返回查询结果给用户, 其步骤如下:

(1)利用查询谓词在 XML 数据库中出现的概率分布[17]来从查询闭包中选取变异查询树的方法来求解最优变异查询树, 这里的最优是指通过该最优变异查询树对 XML 数据库扫描一次就可以获取满足用户指定的 K 个查询结果的要求.

(2)利用(1)步求出的最优变异查询树对 XML 数据库进行精确嵌入, 执行结束后, 如果得到的查询结果个数仍然小于 K, 则从查询闭包中顺序选取最优变异查询树的下一棵变异查询树继续对 XML 数据库进行精确嵌入, 如此循环, 直到查询结果个数达到 K 个为止.

(3)从查询闭包中顺序提取最优变异查询树之前的变异查询树, 对已存在于结果集中但没获取相应的近似度值的查询结果进行精确嵌入. 若嵌入成功则将该变异查询树对应的嵌入代价作为它的近似度值, 否则继续从查询闭包中顺序提取变异查询树进行精确嵌入, 直到得到它相应的近似度值. 显然, 由定理 1 可推知(2)步中得到的查询结果一定能在步(3)中找到其相应的近似度值.

(4)返回 K 个按近似度由高到低排列结果序列给查询用户.

5.3 时间复杂度分析

设 XML 数据库的规模为 n, 查询闭包 Q^* 中的变异查询树的个数为 s, 若 5.2 小节中求解得到的最优变异查询树在查询闭包 Q^* 中的序列号为 m, 那么在最好的情况下, 算法只需要对数据库扫描一遍, 就可得到全部的 Top-K 结果, 此时算法的复杂性是 $O(n)$; 在最坏的情况下, 要依次应用查询闭包中的所有元素, 才能得到全部的 Top-K 结果, 此时算法的复杂性是 $O(n^{s-m+1})$, 因此算法的复杂性是多项式时间的.

6 实验

AFXDQ(automobile figuration XML database query)原型系统是基于国家 973 关于智能化汽车外型设计的检索要求实现的, 主要承担对后台基于 XML 描述的汽车外型部件进行查询. 目前原型系统的结构图如图 1 所示:

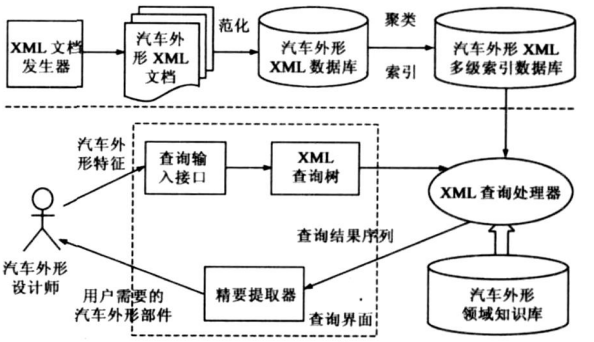

图 1 AFXDQ 原型系统结构图

如图 1 所示, AFXDQ 原型系统由四个部分构成. 中间虚线上方的是 XML 数据存储部分, 通过对范化后的带有索引结构的 XML 数据树进行基于语义树间距离的聚类, 从而建立带有多级索引的 XML 数据库. 虚线下方是 XML 数据查询部分, 其中长方形虚线框部分是 XML 近似查询界面, 它需要一个精要提取器从查询出的描述汽车外型的 XML 文档中提取用户想要的部件. XML 查询处理器是系统的核心部分, 主要由 TRXQ 算法构成. 汽车外形领域知识库, 主要是为变异操作代价的计算提供支持.

实验中, 利用自行开发的 XML 数据发生器为五个不同的汽车外型 XML DTD 分别生成了 200 个 XML 文档, 总共包含 200,000 个文档元素(结构结点), 20,000 个查询词(值结点), 200,000 个文档元素共享了 150 个不同的元素名(结点标签). 20,000 个查询词共出现了 1,000,000 次, 并呈现均匀分布. 所有的实验均在 530MHz 的奔腾 3 处理器, 256M 内存, Windows 2000 的操作系统平台下进行, 并利用 VC++ 6.0 进行了编码实现.

实验中, 根据汽车外型 XML DTD 设计了 10 条具有代表性的类 XPath 查询路径表达式来反映用户多种查询需求, 如表 1 所示:

表 1 类 XPath 查询路径

路径标识	类 XPath 查询路径
Q1	Car/door/leftfront
Q2	Car/frontglass/FGKeypoint
Q3	Car/door/@coverbord/EHHlength
Q4	Car/carID/@frontview/wholewidth("1666")
Q5	Car_body[brand("sampas"), frontform("round"), rear("round")]
Q6	Car[@brand("golf"), overwidth("1825"), bootlength]
Q7	Car(door, coverbord[@EHHlength("1044"), @EHHwide("1200"), canopy])
Q8	Car[sideglass, frontglass(FGKeypoint("0.626 1245"))]
Q9	Car/door[@leftfront("750"), @rightfront("850")]//glass[FGKeypoint("0.626 1245")]
Q10	Car//carID("passat")/frontview[wholewidth("1666"), backview[wheelbase("2731"), bootwidth("1558")]]

其中, Q1~Q4 属于 XPath 绝对路径("/"代表父子

关系标识符), Q5~ Q8 属于包含拥有关系的 XPath 绝对路径("[]"代表拥有关系标识符), Q9 和 Q10 属于包含拥有关系的 XPath 相对路径("//"代表子孙后代关系标识符). 基于以上 10 条类 XPath 查询路径, 本文从查询速度, 查准率和查全率三个方面对三种调整模式下的 TPQR 算法和 SSO 算法[11]的性能进行了比较, 其测试结果如下:

(1) TRXQ 算法; (2) TRXQ 算法(不包含模式重写阶段); (3) TRXQ 算法(不包含模式重写阶段和重命名变异操作); (4) SSO 算法;

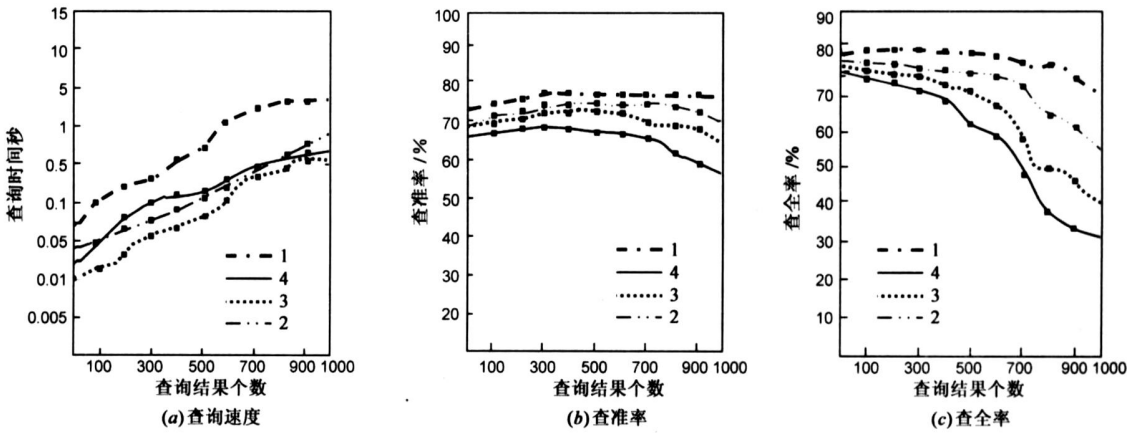

图 2　算法性能比较

图 2(a)(b)(c)分别显示了 TRXQ 和 SSO 查询算法在包含和不包含模式重写阶段的四种情况下关于查询速度, 查准率和查全率三个方面的性能测试结果. 图中的性能曲线中的每个点都代表对给定的 10 条类 XPath 查询路径的平均评估结果. 图 2(*a*) 表明 TRXQ 算法在返回同样数量查询结果的情况下所用的时间最长, 这是因为模式重写阶段对原始查询树的重写花费了大量的时间, 而在调整为 2 和 3 两种模式时, TRXQ 算法所用的查询时间和 SSO 算法差别不大. 图 2(*b*) 表明 TRXQ 算法在三种模式下和 SSO 算法查询的准确度基本上能稳定在 70~ 60% 之间, 而随着样本数量的增大, 由于 TRXQ 算法中采用了本文提出的基于样本数据分布统计的基本变异代价计算方法, 在选择大于 600 个样本以后, 查准率基本上在 70% 左右逐渐稳固下来. 图 2(*c*) 表明包含模式重写阶段和重命名变异操作的 TRXQ 算法的查全率远高于 SSO 算法, 尤其在样本的数量逐渐增多的情况下更是如此, 这是由于随着样本总量的增加, 来源于不同 DTD 的 XML 样本的数量也随之增多, 这些不同 DTD 的 XML 样本之间的差异恰恰可以靠模式重写阶段来弥补, 而这正是 SSO 算法所缺失的. 实验结果也表明 α, β_{inse}, β_{naming} 和 β_{del} 这四个代价模型中参数的取值会直接影响查询结果的准确率, 这些值的选取和当前所采用的 XML 文档样本集有一定的关系, 均取 0.5 是一种折衷的方法, 可以使不同样本集带来的差异对查询算法性能的影响程度尽可能的小.

以上实验表明 TRXQ 算法能够根据样本库的组成特性和用户需求的不同而进行相应的调整, 与 SSO 算法相比, TRXQ 算法在查准率和查全率两方面的性能分别提高了 5%~ 10% 和 30~ 38%.

7　结论

本文提出了基于两阶段查询重写的 XML 近似查询算法 TPQR. 该算法通过模式重写策略, 将原始查询树改写为多种 XML 模式下的重写查询树, 从而解决了多源 XML 数据库带来的查询遗漏问题. 通过利用基本变异操作序列得到的近似查询树对 XML 数据树完成精确嵌入, 从而将 XML 近似查询的问题转变为多棵近似查询树的精确查询问题. 最后, 在国家 973 关于汽车外形智能化设计子项目的实验中表明, TPQR 算法能够根据样本库的组成特性和用户需求进行相应的调整, 从而在衡量算法性能优劣的查准率、查全率、查询速度 3 个方面均优于或接近 SSO 算法. 下一步工作将考虑通过引入量化本体的方法使查询代价的计算更加的精确, 从而使查询结果更接近用户的查询语义. 另外, 对 XML 索引技术的研究也是接下来研究工作的重点, 尤其是能够设计出高效、实用的基于 XML 索引技术的近似查询算法.

参考文献:

[1] A Theobald, G Weikum. Newblock adding relevance to XML [A]. Newblock 3rd International Workshop on the Web and Databases[C]. Dallas, Texas, 2000.

[2] J M Bremer, M Gertz. XQuery/IR: integrating XML document and data retrieval[A]. Proceedings of the 5th International Workshop on the Web and Databases[C]. Madison, Wisconsin, USA, 2002, 1- 6.

[3] 路燕, 张亮, 等. XML 查询中 DTD 的排序技术[J]. 计算机

研究与发展, 2003, 40(11): 1579- 1585.
Lu Yan, Zhang Liang, et al. DTD ranking in the smart XML query[J]. Journal of Computer Research and Development, 2003, 40(11): 1579- 1585. (in Chinese)

[4] 胡勤友, 胡运发. 基于扩展路径表达式的 XML 查询[J]. 计算机研究与发展, 2003, 40(5): 721- 727.
Hu Qinyou, Hu Yunfa, et. al. XML querying based on extended path expressions[J]. Journal of Computer Research and Development, 2003, 40(5): 721- 727. (in Chinese)

[5] T Schlieder. Similarity search in XML data using cost based query transformations[A]. ACM SIGMOD 2001 Web and Databases Workshop[C]. Santa Barbara, California, May, 2001. 19- 24.

[6] C Delobel, M C Rousset. A uniform approach for querying large tree-structured data through a mediated schema[A]. International Workshop on Foundations of Models for Information Integration[C]. Viterbo, Italy, 2001.

[7] N Fuhr, K Grossjohann. XIRQL: an extension of XQL for information retrieval[A]. ACM SIGIR Workshop on XML and Information Retrieval[C]. Athens, Greece: ACM Press, 2000. 172- 180.

[8] Torsten Schlieder. Schema-driven evaluation of approximate tree-pattern queries[J]. EDBT' 02[C]. Prague, Czech Republic, 2002, March 25- 27, 514- 532.

[9] S Amer-Yahia, S Cho, D Srivastava. Tree pattern relaxation[A]. EDBT' 02[C]. Prague, Czech Republic, March 25- 27: 496- 513.

[10] P Kilpelainen. Tree matching problems with applications to structured text databases[D]. PhD thesis, University of Helsinki, Finland, November 1992.

[11] Sihem AmerYahia, L V Lakshmanan, S Pandit. FleXPath: flexible structure and full-text querying for XML[A]. In Proceedings of the ACM SIGMOD Conference on Management of Data[C]. Paris, 2004. 83- 94.

[12] E Damiani et al. The APPROXML tool demonstration[A]. In EDBT 2002[C]. Berlin: Springer-Verlag, 2002, 753- 755.

[13] S Al-Khalifa, H V Jagadish, N Koudas, J M Patel, D Srivastava, Y Wu. Structural joins: A primitive for efficient XML query pattern matching[A]. Proc of the 18th Int'l Conf on Data Engineering[C]. Los Alamitos: IEEE Press, 2002, 141- 152.

[14] Federica Mandreoli, Riccardo Martoglia, Paolo Tiberio. Approximate query answering for a heterogeneous XML document base[A]. In Proc. of the The 5th International Conference on Web Information Systems Engineering[C]. LNCS 3306, 2004, 337- 351.

[15] Lassila O, Swick R. Resource Description Framework (RDF) model and syntax specification[D]. W3C Working Draft WD-rdf-syntax-19981008, 1998.

[16] Melnik, S, Garcia-Molina, H, Rahm, E. Similarity flooding: a versatile graph matching algorithm and its application to schema matching[A]. Proc of the 18th ICARE[C]. USA: Morgan Kaufmann Publishers, 2002, 117- 128.

[17] R Goldman, J Widom. DataGuides: enabling query formulation and optimization in semistructured data[A]. In Proceedings of the 23rd International Conference on Very Large Databases (VLDB)[C]. Athens, Greece, August 1997, 436- 445.

作者简介:

衡星辰 男, 1980 年 2 月生于陕西省汉中市, 西安交通大学电子商务研究所博士生. 研究方向: XML 信息检索, 计算智能和贝叶斯网等.
E-mail: boyhxc@163.com

覃 征 男, 1956 年 9 月生于湖南石门, 西安交通大学电子与信息工程学院教授、博士生导师; 清华大学信息学院、软件学院教授、博士生导师; 西安交通大学电子商务研究所所长, 赴美高级访问学者. 主研方向: 数据融合、软件项目管理、电子政务和电子商务等.

邵利平 男, 1978 年生, 博士生、研究方向为 XML 查询、信息隐藏等.

曹玉辉 男, 1966 年生, 博士生, 讲师. 研究方向为 XML 查询、知识发现等.

高洪江 男, 1969 年 3 月生于新疆昌吉, 西安交通大学电子商务研究所博士生, 研究方向为图像处理和软件工程.