# D(k,l)-索引:一种自适应的XML数据索引*)

丁道峰 吴红伟 王晓玲 周傲英
(复旦大学计算机科学与工程系 上海200433)

**摘 要** XML已成为网络上数据表示和交换的一种实际标准。为促进XML的数据和半结构化数据的查询,几种结构概要被提出。它们可以直接从数据中得出,并以索引的方式来估计在XML数据上的路径表达式。在本文中,综合几种索引提出新型数据结构D(k,l)索引。其参数k,l刻画了节点向上和向下的相似度。它考虑各个节点向上路径和向下路径的相似关系,因此它可以有效地支持路径表达式,尤其支持带分支路径表达式的查询,同时,它也可以根据查询情况的变化来动态地改变索引结构,使索引结构更适合当前的查询要求,实验表明我们的方法具有很好的效率和效果。

**关键词** 索引,XML,路径表达式,适应的

## D(k,l)-Index: An Adaptive Structural Summary for XML Data

DING Dao-Feng WU Hong-Wei WANG Xiao-Ling ZHOU Ao-Ying
(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

**Abstract** XML has become the de facto standard of data presentation and exchange on the Web and Internet. To facilitate queries over XML data or semistructrued data, various structural summaries have been proposed. And they are derived directly from the data and serve as indices for evaluating path expressions on XML data. In this paper, D(k,l)-index, a family of efficient approximate index structures is proposed in which data nodes are grouped according to their incoming paths of length up to k and outgoing paths of length up to l. D(k,l)-index fully exploit local similarity of XML data nodes on their upward and downward paths, so that it can be used to evaluate path expressions efficiently, especially for the branching path expressions. At the same time, D(k,l)-index is able to adjust its structure according to the current query load adaptively. In addition, we propose a method in order to adjust the index structure dynamically for a query workload. Preliminary experiments show that our method is very effective and efficient.

**Keywords** Index, XML, Path expressions, Adaptive

## 1 引言

随着XML在数据分发领域中的应用日益广泛和深入,针对XML数据的查询也得到了广泛的关注[1~4]。路径表达式的执行和处理是XML查询中的重要方面之一,最简单的路径表达式的执行方法是直接在整个XML源文档上进行查找,其缺点是效率较低。路径索引通过将查找限制在仅与查询有关的XML文档片段上以提高路径表达式的执行效率。因此,如何从半结构化数据中抽取路径索引结构也备受关注。这类索引结构有DataGuide[5],1-index[6],A(k)-索引[7],UD(k,l)-索引[8],D(k)-索引[9]等。DataGuide提出的方法是建立一个带标签的有向图作为索引结构,其主要思想是用尽可能少的节点和边来保存数据图中所有的路径信息。它对每一条路径都建立了一个索引节点,这样可以由一条以上的路径到达的节点就被重复记录。1-index引入了"相似"概念,两个节点相似,即所有可能到达它们的路径节点集是相同的。1-index将所有"相似"的节点看作一个等价类,然后为每个等价类建立一个索引节点。1-index保证了索引空间不会超过源数据空间的大小,但是由于"相似"的概念比较严格,数据压缩的效果不是很明显。针对这种情况,A(k)-索引提出了"局部相似"的概念,即对同一个等价类中的节点,只要求能到达它们的长度小于k(k是刻画局部相似度大小的一个参数,由用户指定)的路径集是相同的,随着相似概念的放宽,索引的大小也被大大压缩了,但是期望结果可能是由该索引得到的查询结果的子集,从而还需要到源数据上进行验证。基于A(k)-索引,文[8,9]提出了UD(k,l)-索引和D(k)-索引,UD-(k,l)索引不仅考虑节点向上路径的相似性,也考虑它们向下路径的相似性;D(k)-索引则可以根据具体的查询情况动态地构建索引。

在综合分析比较了上述几种索引的基础上,本文中,我们提出了D(k,l)新型索引结构,其中,参数k和l刻画节点的向上相似度和向下相似度。该索引结构不仅考虑各个节点的向上路径的相似关系,也考虑了向下路径的相似关系,因此很好地支持了带分支路径表达式的查询;同时它也可以根据查询情况的变化来动态改变索引结构,使索引结构更适合当前的查询要求。此外,我们还提出了根据查询情况来调整索引结构的具体方法。实验研究表明,D(k,l)-索引在查询带分支路径表达式的情况下具有显著的效果。本文以下部分组织如下:第2节描述本文中使用的基本概念和模型;第3节介绍D(k,l)-索引和它的构造算法,以及在该索引图上执行路径表达式的实现技术;第4节描述了如何根据查询情况来动态调整索引结构的策略;第5节给出了实验结果及分析;最后,我们总结全文,给出结论,并对未来的工作进行展望。

## 2 背景知识和基本概念

我们首先介绍本文中将要使用的一些概念和定义。我们将XML文档模式化为一个带标签的有向数据图G,图G中的每个节点都对应一个标签和一个唯一的ID号,图G中的边表示元素的父子关系,元素-属性关系,以及元素间的引用

*)本文受中国国家自然基金(NO.60228006)资助。

的每个节点都对应一个标签和一个唯一的ID号，图G中的边表示元素的父子关系，元素-属性关系，以及元素间的引用关系。每个文档都有唯一的一个标签为Root的根节点。如果不考虑引用关系，那么XML文档的基本结构就是一个树状结构，边表示元素间的父子关系和元素-属性关系，这部分边在我们的数据图中用实线表示，而虚线边则表示元素间的引用关系。

图1显示了一个描述歌曲信息数据库的XML文档的部分信息。节点中的数字代表该节点的ID号，旁边的字符串代表该节点对应的标签。

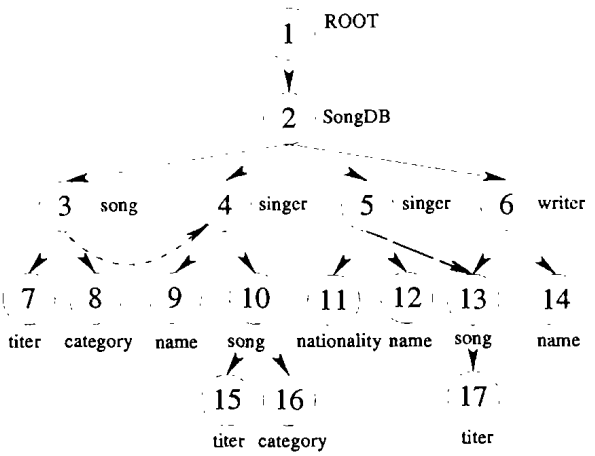


图1 XML数据图

我们用分割符(/)，二分符(|)，循环符(*)和替代符(?)来定义了一种正则路径表达式R，如下所示：

$R_{::}=\epsilon|l|\_|R/R|R|R|(R)|R?|R*$

用$\sum_G$表示图G中所有标签的集合，$l\in\sum_G$，'_'是一个能匹配$\sum_G$中任意一个标签的特殊符号。进一步，我们可以定义带分支的路径表达式，如下所示：

$BR_{::}=R[R]\mid R/BR\mid BR/BR\mid BR/R$

这里R是一个基本的正则表达式。对于带分支的路径表达式$R_1[R_2]$来说，主路径是指去掉带分支的路径表达式中的所有中括号括起来的部分(包括中括号本身)后得到的路径表达式，而一条分支路径则是指一个中括号括起来的那部分路径。分支路径出现在路径$R_1$的最后一个标签上，这个标签被称为分支节点。一个带分支的路径表达式由一条主路径和若干条分支路径这两种路径构成。不带任何分支节点的路径表达式称为简单路径表达式。

主路径表达式是从XML数据图的根结点开始的。给定一个路径表达式R，如果XML数据图中的某节点的标签路径和R匹配的话，就说R匹配这个节点。R的执行结果就是所有匹配R的节点的集合。例如在图1中，对于简单路径表达式ROOT/SongDB/singer来说，其执行结果包含两个节点{4,5}，因为节点路径1.2.4和1.2.5所对应的标签路径都匹配该路径表达式。下面是针对图1的另外两个例子：一个是简单路径表达式ROOT/_*/name，它检索XML文档中所有标签为name的节点，其执行结果为{9,12,14}；另一个是带分支的路径表达式ROOT/SongDB/singer [nationality]，它检索带国籍信息的歌手，只有节点5符合要求。

## 3 D(k, l)-索引

1-index和DataGuide对XML数据图中的所有路径建立精确索引。因此，即使两个节点非常相似，也可能因为一条长路径将它们区分到不同的索引节点中存储。然而，像这样的长路径在查询中是极少用到的。基于这个观察，我们试图利用节点的局部相似度来构建索引图，以此来减小索引图的大小。很重要的一点是我们不仅支持节点的向上相似关系也支持节点的向下相似关系，而A(k)-索引和D(k)-索引都不支持节点的向下相似关系。通过对向下相似关系的支持，可以有效地提高执行带分支的路径表达式的效率。例如，在图1中，A(k)-索引和D(k)-索引都将节点4和节点5归为一类，由一个索引节点指代，但是执行带分支的路径表达式ROOT/SongDB/singer [nationality]却只有节点5这一个查询结果。在本文中，通过比较以节点为根的向下标签路径，我们使用D(k,l)-索引有效地区分了类似节点4和节点5的这类节点。

同时，我们知道，如果查询表达式以短路径为主，索引节点的相似度要求可以降低以较大程度地压缩索引的大小；如果查询表达式以长路径为主，索引节点的相似度要求应当提高以减少在源数据文档上进行验证的次数。那么，随着查询情况的改变，我们就需要动态地调整索引的结构来适应当前的查询情况。D(k,l)-索引具备了这种动态调整的能力，而A(k)-索引和UD(k,l)-索引都不具备这种能力。

### 3.1 D(k, l)-索引定义和性质

D(k,l)-索引是根据节点的长度为$k$的输入路径和长度为$l$的输出路径来对数据图中的节点进行分类的。我们使用了$k$-$l$相似度这样一个概念。如果两个节点$u$和$v$存在关系$u\approx_l^k v$，我们就说$u$和$v$的向上相似度为$k$，向下相似度为$l$。

$\approx_l^k$($k$-$l$-相似关系)须符合以下三个条件：a) 对于任意两个节点$u$和$v$，$u\approx_0^0 v$当且仅当$u$和$v$具有相同的标签；b) $u\approx_l^k v$当且仅当$u\approx_l^0 v$并且对于节点$u$的任一个父节点$u'$，存在节点$v$的一个父节点$v'$，使得$u'\approx_l^{k-1}v'$，反之亦然；c) $u\approx_l^k v$当且仅当$u\approx_0^k v$并且对于节点$u$的任一个子节点$u'$，存在节点$v$的一个子节点$v'$，使得$u'\approx_{l-1}^k v'$，反之亦然。

我们现在描述D(k,l)-索引的性质如下：

**性质1** 对于节点$u$和$v$，如果$u\approx_l^k v$，则它们所有长度为$m(m\leqslant k+1)$的输入标签路径都是相同的，所有长度为$n(n\leqslant l)$的输出路径也都是相同的。

**性质2** 如果D(k,l)-索引中某索引节点的向上相似度为$k_1$，向下相似度为$l_1$，则对于任意一条针对该节点的长度为$m(m\leqslant k_1+1)$的简单路径表达式和长度为$n(n\leqslant l_1)$的分支路径表达式的查询结果都是精确的。

**性质3** 如果D(k,l)-索引中某索引节点的向上相似度为$k_1$，向下相似度为$l_1$，则对于针对该节点的长度为$m(m>k_1+1)$的简单路径和长度为$n(n>l_1)$的分支路径，D(k,l)-索引的查询结果是近似的。它返回的结果集只是一个可能的结果集，并不一定是真正的结果集。我们需要到源数据图上对结果集进行验证。

### 3.2 D(k,l)-索引的构造算法

D(k,l)-索引具体的构造算法如图2中所示。对于每一个标签$L_i$都对应了一个向上相似度$k_i$和一个向下相似度$l_i$。执行了一定数量的查询后，$k_i$和$l_i$的大小根据具体的查询情况确定，而在此之前需要对每个标签提供默认的$k_i$,$l_i$值。首先，我们对所有的数据节点按其标签进行分类，这样我们得到了一个节点集列表，每一个节点集都有一个唯一的标签。显然，此时满足的索引节点的向上相似度和向下相似度均为0。然后我们将向上相似度需求值大于0的节点的向上相似度提高为

1，提高的方法是对于节点 $n_i$ 的每个父节点 $n_j$，将 $n_i$ 这个索引节点指代的数据图上的节点集，分裂为属于 $n_j$ 的子节点集的部分和不属于 $n_j$ 的子节点集的部分。依次类推，直到所有的节点要求的向上相似度均被满足。随后，我们计算每个节点集的向下相似度，同样反复分裂节点直到所有节点要求的向下相似度均被满足。对于图 G 中任意两个有边相连的顶点 $u$ 和顶点 $v$，如果索引节点 $A$（其对应的节点集包含 $u$）和索引节点 $B$（其对应的节点集包含 $v$）之间没有边的话，我们就添加上这条边。

**Algorithm** 1: The D($k$,$l$)-index Construction Algorithm

Input The data graph G, and local similarity requirements of label nodes specified by the query load.
Output The D($k$,$l$)-index graph $I_G$.

1. Build the label-split index graph $I_G$ from G;
2. Compute the local similarities of index nodes in $I_G$;
3. X is a copy of $I_G$;
4. For k=1 to kmax //kmax is the maximal up-local-similarity requirement in $I_G$
   For each index node $n_i$ in X
   If (its local similarity requirement≥k)
   For each parent $n_j$ of $n_i$ in X
   Replace the node $n_i$ in $I_G$ with $n_i \cap Succ(n_j)$ and $n_i$-$Succ(n_j)$;
   Update the edges in $I_G$;
   Set the local similarity requirements to newly created index nodes by inheritance;
   Set Xto be a copy of the updated $I_G$;
5. For l=1 to lmax // lmax is the maximal down-local-similarity requirement in $I_G$
   For each index node $n_j$ in X
   If (its local similarity requirement ≥ k)
   For each child $n_i$ of $n_j$ in X
   Replace the node $n_j$ in $I_G$ with $n_j \cap Prec(n_i)$ and $n_j$-$Prec(n_i)$;
   Update the edges in $I_G$;
   Set the local similarity requirements to newly created index nodes by inheritance;
   Set X to be a copy of the updated $I_G$;
6. Return the resulting $I_G$.

图2 D($k$,$l$)-索引构造方法

### 3.3 带分支的路径表达式查询

本节将介绍在 D($k$,$l$)-索引上进行查询的策略，这是一种基于验证的策略。

一个正则路径表达式能够被转换为一个确定有限自动机 DFA。我们可以把 XML 数据作为这个 DFA 的输入，运行这个 DFA 来找到那些可以驱动 DFA 到达最终状态的节点。也就是说，这些节点匹配了这个正则路径表达式。一个带分支的路径表达式可以转换一系列的 DFA。我们可以通过为数据图不断选择相应的 DFA 的方法来执行带分支的路径表达式。如果存在索引图的话，我们可以将索引图代替数据图作为 DFA 的输入进行查询，以减少查询代价。在一个索引图上执行带分支的路径表达式查询过程如下：

1. 构造一个 DFA A 对应带分支路径表达式的主路径，并为每个分支路径构造一个 DFA $A_i$。

2. 在索引图上运行 DFA A，也就是对索引进行深度优先遍历，自动机根据匹配的节点自动进行状态转换。

3. 为分支节点建立一张表来记录它们的中间结果。

4. 当 DFA A 到达一个分支节点时，我们暂停自动机 A 的执行，根据分支路径启动相应的 DFA $A_i$。当 $A_i$ 到达最终状态时，我们停止 $A_i$，验证 DFA A 停留的那个索引节点对应的节点集，然后在中间结果表中存储经验证证明是正确的节点，之后继续执行 DFA A。如果不能到达最终状态的话，DFA A 就不能继续往前执行了，而是要先返回它的前一个状态，再往下执行。

5. 建立一张表来避免重复在自动机的某一相同状态访问同一个索引节点。也就是说，当自动机接收一个节点时，我们先查找这张表，如果发现我们已经在同样的状态下接收过这个节点时，就跳过这个节点去接收下一个节点。这样就避免了对于有环路的索引的无限循环。

6. 当 DFA A 到达最终状态时，此时索引图到达某个节点，我们再对这个索引的节点集进行确认，确认正确的结果将添加到最终结果集中。

D($k$,$l$)-索引自身的性质决定了，不论是一个分支节点的中间结果还是索引图返回的“最终”结果都只是一个可能的结果集。所以对于较长的查询路径我们需要在数据图上验证结果集中的节点。

$$\underbrace{A_1/\cdots/A_{i_1}}_{n_1}[\underbrace{B_1/\cdots/B_{j_1}}_{m_1}]/\cdots/\underbrace{A_{i_x+1}\cdots A_{i_{x+1}}}_{n_{x+1}}$$

如上所示，带分支的路径表达式被分支节点分割成几部分。若索引节点 $N$ 在分支节点 $A_{i_1}$ 的中间结果中，那么 $N$ 中指代的数据节点只是有可能是中间结果而已，我们需要对它进行向上和向下方向的验证。如果 $N$ 的向下相似度大于等于 $m_1$，由性质2可知我们不需要进行向下验证。如果 $N$ 的向上相似度大于等于 $n_1-1$，并且这是第一个分支节点，我们不需要进行向上验证。否则，我们就需要到源数据图上进行验证。

## 4 动态调整 D($k$,$l$)-索引

随着查询流的改变，我们需要动态地调整索引的结构来适应当前的查询情况。当查询的路径表达式由以短路径为主转变为以长路径为主的时候，对于索引图中的节点的相似度的要求就会提高，我们需要提高这些节点的相似度要求来减少在源数据上进行验证的次数。例如图3中，索引图一中标签为 $F$ 的节点的向上相似度是1。如果查询标签 $F$ 的标签路径一般长度都为二的话，即查询的路径表达式为 $E/F$，则可以在索引图上得到精确的查询结果，也就是说该索引图对于 $F$ 要求向上相似度为1是合适的。但是，当查询情况改变时，如，查询的路径表达式一般都是 $B/E/F$ 或 $C/E/F$ 的形式时。在索引图上得到的查询结果就需要到源数据图上进行确认验证，而经常做这样的确认显然是很低效的。此时如果将 $F$ 的向上相似度提高为2，则索引图的结构变为索引图二的形式，此时在索引图上进行 $B/E/F$ 或 $C/E/F$ 形式的查询就是精确的了。

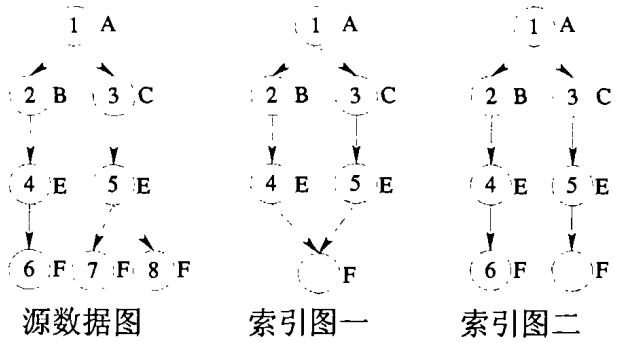


图3 提高索引节点的向上相似度

反之，当查询路径表达式由以长路径为主转变为以短路径为主时，则需要降低一些节点的相似度来更好地压缩索引空间，从而提高索引的效率。节点需要的相似度是这样确定的。当一个标签路径 $A_1/\cdots/A_{i_1}[B_1/\cdots/B_{j_1}]$ 执行时，我们认为主路径中标签所需的向上相似度为 $i-1$；分支节点 $A_{i_1}$ 所需的

向上相似度为 $i_1-1$，向下相似度为 $j_1$；分支路径中的标签 $B_i$ 所需的向下相似度为 $j_1-i$。对于源数据图中存在的每个标签，我们都记录它每次查询中所需的向上相似度 $k_i$ 和向下相似度 $l_i$，执行 $N$ 次查询后，我们对于每个标签取适合的 $k$ 和 $l$，使得 $k$ 和 $l$ 尽可能小，并且相关于该标签向上相似度的 $N_1$ 个查询中，只有 $\delta * N_1$ 个查询要求的向上相似度大于 $k$；对于相关于该标签向下相似度的 $N_2$ 个查询中，只有 $\delta * N_2$ 个查询要求的向下相似度大于 $l$。$\delta$ 是自定义的一个阈值。

执行一段时间的查询后，取得了各个节点需求的适应目前查询情况的相似度数值，我们就可以对原索引结构进行调整了。

### 4.1 调整节点的向上相似度

首先我们找出所有需要提高向上相似度的节点。按需求的向上相似度从大到小排列，依次进行调整。当节点 $A$ 的向上相似度需要由 $k_1$ 增加为 $k_2$ 时。我们首先试图将 $A$ 的所有的父节点的向上相似度调整为 $k_2-1$。如果 $A$ 的所有的父节点的向上相似度已经大于等于 $k_2-1$，那么对于 $A$ 的每个父节点 $B_i$，将 $A$ 中这个索引节点指代的数据图上的节点集，化分为属于 $B_i$ 的子节点集的部分和不属于 $B_i$ 的子节点集的部分。如果 $A$ 的某个父节点的向上相似度小于 $k_2-1$，则递归地调整该节点的相似度为 $k_2-1$。

当调整完所有需要增加向上相似度的节点后，我们可以调整需要减少向上相似度的节点，在描述调整方法之前，我们先介绍一些相关知识。

数据图的细分：我们说图 G' 是图 G 的一个细分就意味着 G' 中任意标签 A 的向上相似度 k' 大于等于图 G 中标签 $A$ 的向上相似度 $k$，向下相似度 $l'$ 大于等于图 G 中标签 $A$ 的向下相似度 $l$。显然，任何源数据图都是它的 $D(k,l)$-索引图的一个细分。

**性质4** G 是一个 XML 数据图，$I_G$ 是 G 的 $D(k,l)$-索引图，如果 $I'_G$ 是 $I_G$ 的一个细分，则 $I'_G$ 的 $D(k,l)$-索引和源数据的 $D(k,l)$-索引都是 $I_G$。

例如，图3中索引图二是索引图一的一个细分。如果对于给定的各节点的相似度要求，源数据图的 $D(k,l)$-索引图应该是索引图一的话，对于同样的相似度要求，将索引图二看成是源数据图的话，其 $D(k,l)$-索引图也应该是索引图一。

所以，调整需要减少向上相似度的节点的方法，可以是将目前的索引图看成是源数据图，然后，在此基础上直接用 $D(k,l)$-索引构造算法建立所要求的 $D(k,l)$-索引。

### 4.2 调整节点的向下相似度

同样，经过一段时间的查询，我们也需要根据查询情况调整节点的向下相似度。和调整节点的向上相似度一样，首先我们找出所有需要提高向下相似度的节点。按需求的向下相似度从大到小排列，依次进行调整。当节点 A 的向下相似度由 $l_1$ 增加为 $l_2$ 时，我们首先试图将 $A$ 的所有的子节点的向上相似度调整为 $l_2-1$。如果 $A$ 的所有的子节点的向下相似度已经大于等于 $l_2-1$，那么对于 $A$ 的每个子节点 $B_i$，将 $A$ 中这个索引节点指代的数据图上的节点集，化分为属于 $B_i$ 的父节点集的部分和不属于 $B_i$ 的父节点集的部分。如果 $A$ 的某个子节点的向下相似度小于 $l_2-1$，则递归地调整该节点的相似度为 $l_2-1$。

当调整完所有需要增加向下相似度的节点后，我们可以调整需要减少向下相似度的节点，方法和调整需要减少向上相似度的节点的方法一样，都是直接在已有的索引图上构建 $D(k,l)$-索引图。所以为了提高效率，我们可以在调整索引的向上相似度时暂缓调整向上相似度降低的节点，可以在最后将所有需要降低向上相似度或向下相似度的节点统一调整。

## 5 实验分析

实验环境为：P Ⅳ 2400MHZ 处理器，512M 内存，Window 2000 Server 操作系统。试验数据采用 XMark XML benchmark[10]数据集，大小由1k 到4M 不等。Xmark 数据模拟了一个拍卖站点的数据情况。试验中所用的查询是根据 Xmark 数据的 DTD 随机生成的。

我们使用查询中访问的节点总数来评估查询代价。节点总数是指在索引图上访问的节点数量和在源数据图上进行验证时访问的节点数量之和。

图4显示了 $D(k,l)$-索引和 $D(k)$-索引的比较结果，其中 $\delta$ 取0.1(即只允许至多10%的查询需要到源数据上进行验证)，产生的查询路径的主路径长度不超过7，分支路径长度不超过4。由于路径查询表达式是随机生成的，因此查询代价和文档大小的对应关系不是很规则。从图中可看出，由于考虑了向下相似度，对于带分支的路径查询表达式，$D(k,l)$-索引有效地降低了查询代价。

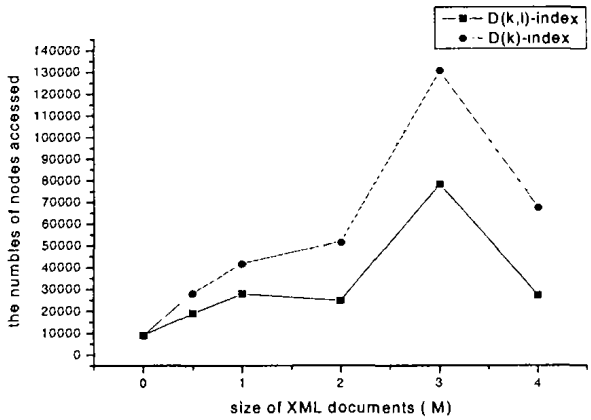


图4 $D(k,l)$-索引和 $D(k)$-索引比较

图5显示了 $D(k,l)$-索引和 $UD(k,l)$-索引的比较结果，其中 $\delta$ 取0.1，产生的查询路径的主路径长度不超过10，分支路径长度不超过5。$UD(k,l)$-索引尝试了参数 $k$ 和参数 $l$ 所有可能的取值，然后取其最好的一种实验结果和 $D(k,l)$-索引的实验结果进行比较。从图中可看出，通过根据查询来动态的调整索引结构，$D(k,l)$-索引的查询效率比 $UD(k,l)$-索引在取得最佳的参数值后得到的查询效率还要略高一些。

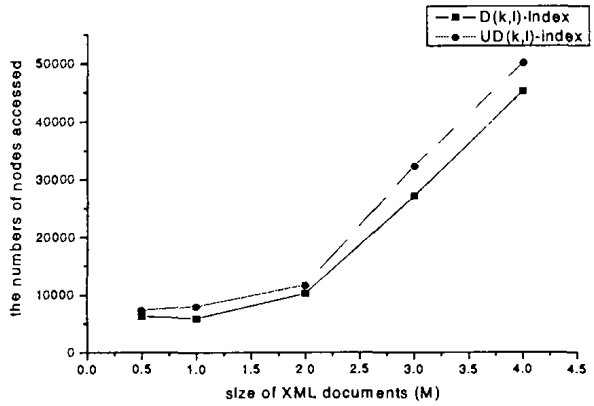


图5 $D(k,l)$-索引和 $UD(k,l)$-索引比较

**总结和展望** 本文提出了一种叫做 $D(k,l)$-索引的新型索引结构来高效地执行针对 XML 数据的路径表达式查询，

它不仅考虑各个节点的向上路径的相似关系而且也考虑它们向下路径的相似关系，这样很好地支持了带分支的路径表达式查询；同时它也可以根据查询情况的变化来动态地改变索引结构，使索引适合当前的查询要求，从而获得较高的查询效率。把基于路径的索引方法和基于元素的索引方法结合起来去处理更复杂条件的查询，这是我们正在进行的研究工作。

## 参考文献


1 Abiteboul S. Querying semi-structured data. ICDT, 1997. 1～18

2 Buneman P, Kaushik R, UnQL D S. A query language and algebra for semistructured data based on structural recursion. VLDB Journal, 2000, 9(1): 76～110

3 Clark J, Derose S. XML Path Language (Xpath). W3C Recommendation, 1999. http://www.w3.org/TR/Xpath

4 Chamberlin D, Florescu D, Robie J, Simeon J, Stefanescu M. XQuery: A query language for XML. W3C Working Draft, 2002. http://www.w3.org/TR/xquery

5 Goldman R, Widom J. DataGuide. Enable query formulation and optimization in semistructured databases. VLDB, 1997. 436～445

6 Milo D, Suciu D. Index structure for path expression. ICDT, 1999. 277～295

7 Kaushik R, Shenoy P, Bohannon P, Gudes E. Exploiting local similarity for efficient indexing of paths in graph structured data. ICDE, 2002. 129～140

8 Wu H, Wang Q, Xu J, Zhou A, Zhou S. UD(k,l)-index: An Efficient Approximate Index for XML Data. WAIM, 2003

9 Qun C, Lim A, Win K. D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data. SIGMOD, 2003. 134～144

10 Xmark. The xml benchmark project. http://monetb.cwi.nl/xml/index.html


(上接第113页)

模型检验器。

建模元素选择栏是一个独立的矩形窗口，内含一个树形控件，其中包含 CWM 的所有建模元素。系统针对用户所选择的建模元素，可以在图形编辑界面中画相应的对象和关联，同时在后台记录用于模型检验和元数据转换的信息。

由于 CWM 只定义了静态属性，因此图形编辑环境只需定制 Rose 的类图。类图及其中的模型元素在 CWM 建模环境下赋予了新的含义：类图成为了对象图，包维持原来的含义，类变成了对象，原来属性的 Initial Value 现在代表对象的属性值，关联在此表示关联的实例。用例图和序列图在新的建模环境仍然可以重用，其它的图在 CWM 建模环境中暂时不需要使用。Rose 各种视图的界面大家很熟悉，这里就不再附图说明。

模型检验器是建模工具中最重要的部分。它必须具备对操作的合法性进行实时检查的能力，并立刻将信息反馈给建模者，同时最大程度保证模型能客观地描述实际信息系统并且符合 CWM 的语义范畴。

由于对象都是从建模元素选择栏中选定的，因此不存在对象超出 CWM 描述范畴的情况。而且在对模型维护的时候，Rose 提供了修改对象的级联作用，即当建模者删除一个对象时，系统会自动将与该对象连接的所有关联实例删除，当然我们需要在后台数据结构中记录下这一信息用于后面的模型检验和元数据转换。然而由于 Rose 具有广义的建模能力，因此对关联的创建和修改不作任何的检查，但是这对于我们针对数据仓库领域的建模需求是远远不够的。因此，模型检验器主要还是针对关联的创建和修改进行检查，根据 CWM 的特点，关联主要有以下三方面的检查：

(1)合法性检查：判断两个对象之间的关联在 CWM 中是否有描述；

(2)导航性检查：针对有向关联(包括有向聚合)，用于判断箭头的方向与 CWM 中的导航性约束是否一致；

(3)多样性检查：判断对象之间的关联数目是否满足 CWM 中的多样性约束。

在介绍核心用例图的时候就已经提到，为了方便实现模型检验，每个会产生元数据的类都对应了一个建模元素类，新建一个对象的同时也产生建模元素类的一个实例，其中记录所有用于模型检验的信息。在实现模型检验的时候，我们为这些信息定义了一个统一的数据结构，同时还设计了几个高效的检验算法(由于篇幅的原因在此就不一一介绍了)。

**结束语** 本文介绍了一个基于 CWM 的元数据管理系统 WMMS，该系统具有强大的元数据管理能力，并且更适用于分布式环境下的元数据管理。目前本课题组已基本完成对 WMMS 中建模工具和元数据转换工具的设计和实现。接下来将进一步完善和扩展建模工具的功能，其中包括在建模工具中实现把各种数据源的元数据如各类厂商的关系模型的元数据、分布式环境下搜索 XMI 格式的元数据等转换成建模工具可识别的模型信息，以及对建模工具产生的元数据进行格式转换以实现在 WMMS 中基于 MOF 的 CORBA 对象的形式存储。同时展开对专门针对基于 CWM 的、以 XMI 形式的元数据智能搜索引擎的研究。至此我们就可以实现一个在分布式环境下的应用开发的闭环。对于本地环境下的元数据操作接口技术中，需要实现将基于 MOF 的元模型到相应 IDL 接口的映射。所以要实现 WMMS 系统强大的元数据管理功能还需要开展很多研究工作。

## 参考文献


1 OMG. Common Wrehouse Metamodel Specification Version 1.1. March 2003

2 Poole J, Chang D, Tolbert D, Mellor D. Common Warehouse Metamodel. John Wiley & Sons, Inc. 2002

3 Rational the e-development company™. Using the Rose Extensibility Interface Version: 2001A.04.00

4 Inmon W H. Building the Data Warehouse Second Edition. 机械工业出版社，2000

5 Poole J 等著. 公共仓库元模型—数据仓库集成标准导论. 彭蓉，何璐璐等译. 机械工业出版社，2004,3

6 李姗姗，宁洪，陈波，等. 通用数据仓库元数据模型(CWM)的研究. 计算机科学与工程

7 李姗姗，宁洪. 基于 CWM 的元数据管理系统中数据转换工具的设计与实现. 计算机工程与应用

8 数据仓库与元数据管理. http://www.dwway.com/file/warehouse&metadata.doc

9 李姗姗，宁洪. CWM 扩展机制的研究. 计算机科学，2003，30(10)：169～171