软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2019,30(4):1062−1077 [doi: 10.13328/j.cnki.jos.005390]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 基于关键字密度的 XML 关键字检索*

覃遵跃[1,2], 汤 庸[1,3], 徐洪智[2], 黄 云[2]

[1](中山大学 数据科学与计算机学院,广东 广州 510006)

[2](吉首大学 软件学院,湖南 张家界 427000)

[3](华南师范大学 计算机学院,广东 广州 510631)

通讯作者: 汤庸, E-mail: ytang4@qq.com

**摘 要**: 关键字检索具有友好的用户操作体验,该检索方式已在文本信息检索领域得到了广泛而深入的应用.对 XML 数据采用关键字检索是目前研究的热点.基于查询语义的 XML 关键字检索方法存在返回大量与用户查询意图无关的查询片段或者丢失符合用户查询意图的片段这两个问题.针对这些问题,在考虑 LCA 横向和纵向两个维度的基础上,提出了用户查询意图与 LCA 相关性的两个规则,根据两个规则定义了 LCA 的边密度和路径密度,建立了综合的 LCA 节点评分公式,最后设计 TopLCA-*K* 算法对 LCA 进行排名,并利用中心位置索引 CI 提高了 TopLCA-*K* 算法的效率.实验结果显示,利用所提出的方法返回的查询节点更加符合用户需求.

**关键词**: XML 关键字检索;边密度;路径密度;TopLCA-K 算法

**中图法分类号**: TP311

中文引用格式: 覃遵跃,汤庸,徐洪智,黄云.基于关键字密度的 XML 关键字检索.软件学报,2019,30(4):1062−1077. http://www.jos.org.cn/1000-9825/5390.htm

英文引用格式: Qin ZY, Tang Y, Xu HZ, Huang Y. Study on keyword retrieval based on keyword density for XML data. Ruan Jian Xue Bao/Journal of Software, 2019,30(4):1062−1077 (in Chinese). http://www.jos.org.cn/1000-9825/5390.htm

## Study on Keyword Retrieval Based on Keyword Density for XML Data

QIN Zun-Yue[1,2], TANG Yong[1,3], XU Hong-Zhi[2], HUANG Yun[2]

[1](School of Data and Computer Science, SunYat-Sen University, Guangzhou 510275, China)

[2](School of Software, JiShou University, Zhangjiajie 427000, China)

[3](School of Computer Science, South China Normal University, Guangzhou 510631, China)

**Abstract**: Keyword search has a friendly user experience; the method has been widely used in the field of text information retrieval. Keyword search on XML data is a hot research topic presently. The XML keyword search method based on query semantics have two problems: (1) a large number of query fragments which are not related to the user's query intention have been returned; (2) the fragments which are consistent with the user's query intention have been missed. Aiming at these problems, two rules of user query intention and LCA correlation are proposed on the basis of the two (horizontal and vertical) dimensions of LCA. The edge density and path density of LCA are defined according to the two rules, and a comprehensive scoring formula on LCA nodes is established, finally, the TopLCA-*K* algorithm is designed to rank LCA. To improve the efficiency of the algorithm, center location index is designed. Experimental results show that the nodes returned by this method are more in line with the needs of users.

**Key words**: XML keyword retrieval; edge density; path density; TopLCA-*K* algorithm

* 基金项目: 国家高技术研究发展计划(863)(2013AA01A212); 国家自然科学基金(61772211, 60970044, 61272067, 61363073); 广东省自然科学基金团队研究项目(2014B010116002, 2015B010109003, 2013B090800024, S2012030006242, 2015B010129009)

Foundation item: National High Technology R&D Program of China (863) (2013AA01A212); National Natural Science Foundation of China (61772211, 60970044, 61272067, 61363073); S&T Projects of Guangdong Province (2014B010116002, 2015B010109003, 2013B090800024, S2012030006242, 2015B010129009)

收稿时间: 2016-07-22; 修改时间: 2017-06-09; 采用时间: 2017-09-13

## 1 引 言

XML 是互联网中信息表示和交换的事实标准,在电子政务、电子商务、金融、出版、科学数据和各种资源数字化等方面扮演着极其重要的角色.利用关键字检索 XML 数据,用户不需要学习复杂的查询语言,并且也无需预先了解 XML 数据的结构知识,因此研究 XML 关键字检索是非常迫切和必要的.

XML 关键字检索只需要返回 XML 树片段,该树片段包含了匹配的所有关键字[1,2].文献[3−8]详细介绍了返回 XML 树片段的检索方式,通常,这种检索方式的结果包含了每个关键字实例的一棵最小连通树,该树的根节点是包含了所有关键字实例的最低公共祖先 LCA(lowest common ancestor).LCA 是针对 XML 关键字检索的所有可能情况,有些 LCA 可能不符合用户的查询意图.为了返回与用户意图密切的 LCA,文献[9−13]采用过滤方法抛弃了与用户意图不相关的 LCA,但这些方法经常过滤掉了用户想要的 LCA,具有低的准确率和召回率[14].针对过滤语义不足的问题,文献[6,15,16]设计了 Top-$k$ 算法,对查询结果 LCA 进行排名并优先选择 Top-$K$ 个结果.对于 XML 数据进行关键字检索也可以通过 TF*IDF 和 PageRank 进行排名[1,9,14,17]以提高用户的检索兴趣.

虽然对 XML 关键字检索研究取得了很多研究成果,但仍然存在一些问题.首先是检索质量问题,例如通过过滤语义 SLCA[11,18,19]和 ELCA[13,20]来产生 LCA 子集,然后对子集进行排名,这种方法存在丢失相关查询结果的问题.第二是查询效率问题,在检索关键字较多时将返回大量的 LCA 结果,处理 LCA 排名主要基于关键字倒排表的规模[8,14],因此,当数据集规模增大时,算法不能很好地满足应用要求.文献[21]提出了一种通用的自上而下(top-down)的处理策略来发现 LCA/SLCA/ELCA 语义的共同祖先重复(CAR)和访问无用节点(VUN)问题,但不能对 LCA 节点进行排名.

针对上面存在的两个问题,本文提出了对 LCA 直接进行排名的 TopLCA-$K$ 方法,该方法综合考虑了查询关键字数量、边的构成和路径等情况,然后计算出每个 LCA 的大小.

本文主要贡献包括:

(1) 提出了 LCA 与用户查询意图相关的两个规则.用户查询意图与查询关键字实例到 LCA 边的数量成反比,以及用户查询意图与 LCA 子树路径的数量成反比.

(2) 建立了计算 LCA 分数的公式.根据前面提出的两个规则,利用边密度和路径密度作为衡量 LCA 值的方法.

(3) 设计了计算 LCA 值的算法 TopLCA-$K$.为了避免丢失部分满足用户需要的查询结果,提出对所有 LCA 进行打分.为了提高查询效率,提出了中心位置索引 CI.根据 LCA 分数进行排名,把最可能满足用户查询意图的 LCA 首先展示给用户.

(4) 利用实际的不同规模数据集对提出的算法进行了测试,实验验证了在查询关键字个数不同的情况下,本文提出的算法具有更高的准确率和召回率,并且返回的 $K$ 个结果更符合用户需要.

本文第 2 节介绍研究背景和相关工作.第 3 节介绍度量 LCA 大小的概念和方法以及 TopLCA-$K$ 算法.第 4 节显示本文提出的算法的实验结果,以及与其他方法的比较情况.第 5 节总结本文的工作,并给出未来的研究方向.

## 2 研究背景与相关概念

### 2.1 研究背景

因为 XML 半结构化数据不像关系数据库那样具有严格的模式,因此对这种类型的数据进行关键字检索是一项复杂的任务.很多文献针对 XML 关键字检索提出了各种类型的查询语义,可以分为基于树模型和基于图模型的查询语义.在基于树模型的关键字查询语义中,LCA 是所有其他语义的基础,其中,SLCA[11]、ELCA[1,13]、MLCA[10]、XSearch[9]、VLCA[12]、TLCA[22]等是基于 LCA 的代表性语义.基于图模型的查询语义主要借鉴关系数据库的关键字查询思想[23],即对于给定的一组查询关键字,返回 CN(connected network).

SLCA[11]过滤掉了后裔 LCA;在 ELCA[1,13]语义定义中,$v$ 是一个 LCA 节点,如果以 $v$ 为根的子树中过滤掉所有其他以 LCA 节点为根的子树后,$v$ 仍然包含所有关键字,则 $v$ 是一个 ELCA 节点;MLCA[10]语义去掉包含了关

键字标签相同但路径不同的 LCA;VLCA[12]和 XSearch[9]在 LCA 的基础上,将不同关键字到 LCA 路径上存在的同名节点的结果排除在外.基于树模型的查询语义除了 LCA 之外,还有 MCT(minimal cost tree)语义[15].

基于图模型的查询语义把 XML 数据建模为一个图,返回 CN(connected network)[24]作为关键字查询结果.每个 CN 都是文档树 $D$ 的一个无环子图 $T$,并且 $T$ 包含查询 $Q$ 中的所有关键字至少 1 次,而 $T$ 的任何真子图都不包含 $Q$ 中的所有关键字.基于 CN 的一种典型查询语义是 MCN(meaning connected network)[19],该查询语义找出给定关键字之间有意义的关系.XKeyword[24]和 BLINKs[25]也是基于图的查询语义.

已经提出的针对 XML 关键字查询语义,或者基于 LCA 进行过滤,例如 SLCA、ELCA 等,或者基于 CN 进行过滤,例如 MCN.虽然这些过滤语义可以返回更加贴近用户意图的查询结果,但用户查询往往很复杂,因此这些过滤的查询语义也显示出了低召回率问题[14].为了弥补过滤查询语义存在低召回率问题,对符合条件的查询结果进行排名是一种有效的解决办法.对查询结果进行排名,可以把与用户查询意图更相关的查询结果放在前面.已有排名方法分为两种,一种是根据 XML 结构和语义上的相关性进行排名[1,6,9,14,15,26,27],另一种是按照某种统计度量办法对 LCA 节点进行评分排名[1,6,9,14,15,28−30].例如 XRank[1]对 PageRank 算法进行了改进而适用于 XML 关键字查询结果排名;XSearch[9]对 TF*IDF 函数进行改进使之适用于 XML 文档树结构的关键字查询排名;XReal[4]的排序思想基于 TF*IDF,考虑的排序因素包括关键字频率、关键字出现顺序、关键字在查询结果中的位置以及结果的大小等.Termehchy 和 Winslett[14]通过采用交互信息来计算结果相干性以进行排名;Nguyen 和 Cao[31]使用交互信息来比较结果,然后在查询结果之间定义了一个支配关系来进行排名;SAIL[15]定义了最小代价树,并且通过使用链接来分析关键字对之间的相关性,最后返回 Top-$k$ 查询结果;XBridge[28]在使用打分函数时考虑了查询结果的结构,然后对查询结果进行分类;文献[25]基于图模型数据,提出了一种结合结构大小、PageRank 以及 TF*IDF 这 3 种排序思想的混合排名机制.文献[26]提出了一个新的对 XML 数据的关键字查询进行排名和过滤语义的方法 XReason 语义,该方法基于模式推理,模式记录了查询匹配的结构和语义特征.为了利用模式进行推理,提出了模式之间的同态关系.该方法的优点是从整体的角度考虑查询匹配,避免了以前语义利用局部 XML 子树比较查询匹配存在的缺陷.

### 2.2 相关概念

一般情况下,把 XML 文档建模为一棵有序标签树,树中的节点代表 XML 元素或者属性,每个节点有一个 id 号(节点唯一的编号)、一个标签(元素或者属性),有的节点是一个值(对应元素文本值或者属性值).

**定义 1(XML 有向树)**. 一个 XML 文档建模为一棵带标签的有向树 $T(V,E,r)$,其中,$V$ 表示节点的集合,$E$ 表示边的集合,$r$ 是树的根节点.

**定义 2(关键字匹配集合 KMS)**. 对给定的 XML 有向树 $T(V,E,r)$和查询关键字 $k$,用 $KMS(k)$表示 $T$ 中所有与关键字 $k$ 匹配的节点集合,$KMS(k)=\{v|v\in V,k=tag(v)$或者 $k=value(v)\}$.其中,$tag(v)$表示节点 $v$ 的标签,$value(v)$表示节点 $v$ 的值.

例如图 1 中,$KMS(A)=\{4,10,15\}$,$KMS(B)=\{6,11,16\}$,$KMS(C)=\{8,12,17\}$.

**定义 3(节点集 $N$ 的最低公共祖先 LCA(lowest common ancestor))**. 给定 XML 有向树 $T(V,E,r)$和 $m$ 个元素节点集 $N=\{v_i|v_i\in V,1\leqslant i\leqslant m\}$,节点 $v=LCA(N)$当且仅当满足以下条件:

(1) 任意的 $v_i\in N$,$v$ 是 $v_i$ 的祖先节点,$v\in V,1\leqslant i\leqslant m$.

(2) 不存在节点 $u(\in V)$,$u$ 是 $v$ 的后裔,并且 $u$ 是任意 $v_i(\in N)$的祖先节点.

**定义 4(LCA 集合 $LCASet(T,Q)$)**. 针对 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,LCA 集合 $LCASet(T,Q)=\{v|v\in V$ 且 $v=LCA(k_{[1,i]},k_{[2,i]},\ldots,k_{[m,i]})\}$,$k_{[1,i]}\in KMS(k_1)$,$k_{[2,i]}\in KMS(k_2)$,…,$k_{[m,i]}\in KMS(k_m)$.

例如图 1 中,查询 $Q=\{A,B,C\}$,$LCASet(T,Q)=\{1,2,9,14\}$,因为节点 $1=LCA(\{4,11,17\})$(其中的一种情况),节点 $2=LCA(\{4,6,8\})$,节点 $9=LCA(\{10,11,12\})$.

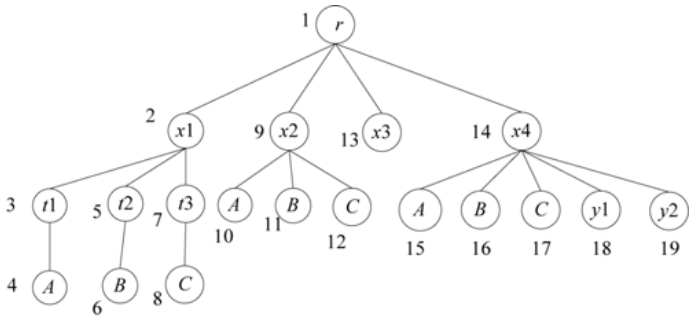



Fig.1 Ordered XML document tree

图 1 有序 XML 文档树

## 3 相关定义

### 3.1 研究动机

图 2 表示某图书馆的藏书情况,是一棵 XML 有序标签树 $T$,圆角矩形中的数字表示节点的 Dewey 编码,旁边是节点的标签,叶子节点下的字符串表示值,例如 $tag(1)=booklist$,$tag(1.2)=book$,$value(1.3.1.1)=BigData\ System$.

考虑对该树的关键字查询 $Q=\{BigData,Felix,James\}$,检索目标是查询书名包含 BigData,作者名字中包含 Felix 或者 James 的图书.这些关键字在树中出现多次,例如 1.1.3.1.1.1、1.2.1.1、1.2.4.1.1.1、1.3.1.1 和 1.4.1.1 这 5 个节点中出现关键字 BigData.按照 LCA 的定义,有 $LCASet(T,Q)=\{1,1.1,1.1.3,1.2,1.2.4.1,1.3,1.4\}$这 7 个节点满足查询 $Q$ 的要求,这些 LCA 中,只有 1.2、1.3、1.4 和 1.2.4.1 满足用户查询要求;1 和 1.1.3 是连接节点,因此不符合查询要求;1.1 标签包含 Felix 和 James,但与 title 对应的书是 XML,而不是 BigData,因此 1.1 虽然是书但也不符合用户查询意图.按照已经提出的过滤语义,例如根据 SLCA 语义则不能返回 1.2 而是返回 1.2.4.1,按照 MLCA 语义则不能返回 1.3,而根据 ELCA 语义返回 1.1.3 节点也有意义,这些过滤语义并不能准确地表达用户的查询意图.

上述查询语义失败的原因是因为那些过滤语义没有充分考虑现实应用的复杂性,直接删除了某些 LCA,导致了低召回率和准确率.

针对已经出现的问题,下面提出了按照查询关键字数量、关键字实例路径、LCA 类型以及关键字实例离散度等综合因素对 LCA 进行度量的新方法,然后按照 LCA 测量值的大小返回 Top-$K$ 个结果.

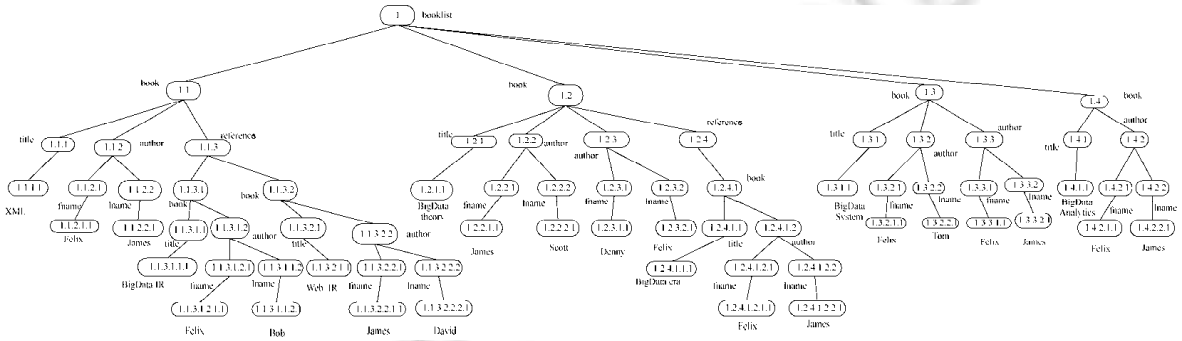



Fig.2 XML for library

图 2 图书馆藏书 XML 文档

### 3.2 LCA与用户意图相关性规则

**规则 1**. 用户查询意图与查询关键字实例到 LCA 节点边的数量成反比.

在 LCA 子树中,查询关键字实例离 LCA 节点越近,表示 LCA 越接近用户的查询意图,查询关键字实例到 LCA 节点边的数量表示了它们之间的距离.

如图 1 所示,关键字查询 $Q=\{A,B,C\}$,$x1$ 和 $x2$ 节点都是查询 $Q$ 的 LCA,显然,查询关键字的实例距离 $x2$ 比 $x1$ 更近,因此,$x2$ 比 $x1$ 更符合用户的查询意图.

该规则确定了查询关键字实例与 LCA 之间的纵向关系.但在实际情况中,除了纵向关系之外还存在横向关系,即查询关键字实例在整个 LCA 子树中的占比情况.如图 1 所示,$x2$ 和 $x4$ 都是查询 $Q$ 的 LCA,根据规则 1,它们的度量值是相同的,但 $x2$ 仅仅包含了 3 个查询关键字实例,而 $x4$ 包含了 $y1$ 和 $y2$ 两个冗余节点,因此,$x2$ 比 $x4$ 更符合用户查询需求.针对这个问题,提出了规则 2.

**规则 2**. 用户查询意图与 LCA 子树路径的数量成反比.

规则 2 反映了 LCA 中查询关键字实例的离散度,LCA 包含的子树相对于查询关键字越多,表示查询关键字之间的离散度越大,说明该 LCA 与用户查询的相关度越小.

### 3.3 基于关键字密度的LCA评分方法

根据规则 1 和规则 2,给出了计算 LCA 排名的有关定义,并利用这些定义建立了计算 LCA 分数的公式.

**定义 5(最小最优 LCA 路径数 *MOPSize*(*Q*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,与查询 $Q$ 对应的 LCA 集合 $LCASet(T,Q)$,则最小最优 LCA 的路径数是关键字的数量,即 $MOPSize(Q)=|Q|$.

对于关键字查询 $Q$,LCA 包含了所有查询关键字实例,最小最优情况是 LCA 节点到叶子节点的每条路径上包含一个查询关键字实例.这种情况在横向轴上没有冗余信息,也不缺少信息.如果 LCA 节点到叶子节点的路径数多于关键字数量,则说明 LCA 有多余信息,这种多余信息可能会对用户产生干扰.例如图 2 中,节点 1.4 是关键字查询 $Q=\{$BigData,Felix,James$\}$的 LCA,$MOPSize(T,Q)=3$,因为 3 个叶子节点到 LCA 的路径刚好包含了 3 个查询关键字;虽然节点 1.3 也是查询 $Q$ 的 LCA,但它的子树有 5 条叶子到 LCA 的路径,说明包含了冗余信息.

**定义 6(关键字与 LCA 的联系度 *CSize*(*T*,*Q*,*u*,*v*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,$u\in LCASet(T,Q)$,$v\in KMS(k_i)(1\leqslant i\leqslant m)$,存在一条从 $u$ 到 $v$ 的路径 $p(u,t_1,t_2,\ldots,t_n,v)$,$CSize(T,Q,u,v)$=路径 $p$ 的长度.

定义 6 表明了 LCA 与查询 $Q$ 的关键字实例之间纵向紧密度,*CSize* 越小,表明关键字与 LCA 越密切.如图 2 所示,1.1.3 和 1.3 都是关键字查询 $Q=\{$BigData,Felix,James$\}$的 LCA,$CSize(T,Q,$1.1.3,Felix$)=4$,表明在 1.1.3 中,关键字 Felix 与该 LCA 的距离是 4,而 $CSize(T,Q,$1.3,Felix$)=3$,表明在 1.3 中,关键字 Felix 与该 LCA 的距离是 3,说明后者比前者更接近用户需求.

为了对规则 1 进行计算,提出了 LCA 边密度的概念,它表示 LCA 子树中查询关键字实例到 LCA 边的数目与查询关键字之间的关系.

**定义 7(边密度 *LCAEDen*(*T*,*Q*,*v*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,某个 LCA 关键字的边密度为

$$LCAEDen(T,Q,v)=\mathrm{Min}\left\{\sum_{i=1}^{m}CSize(T,Q,v,k_i)/MOPSize(T,Q)\right\} \tag{1}$$

$v\in LCASet(T,Q)$,$k_i\in Q$ 且是 $v$ 的后裔,$1\leqslant i\leqslant m$.

边密度 $LCAEDen(T,Q,v)$表示查询关键字实例与 LCA 的纵向联系紧密度.该值越小,说明查询关键字实例与 LCA 联系得越密切,当然,查询结果就越好.如图 2 所示,节点 1.4 是 $Q=\{$BigData,Felix,James$\}$的 LCA,它的边密度 $LCAEDen(T,Q,$1.4$)=8/3$;节点 1.1.3 也是 $Q$ 的 LCA,它的边密度是 $LCAEDen(T,Q,$1.1.3$)=11/3$,因此 1.4 比 1.1.3 更符合用户查询意图.但是 1.3 和 1.4 都是 $Q$ 的 LCA,它们的边密度 $LCAEDen(T,Q,$1.4$)=LCAEDen(T,Q,$1.3$)=8/3$,但是显然,1.3 比 1.4 包含了更多的冗余信息,因此 1.3 没有 1.4 好.

针对边密度相同而 LCA 存在冗余信息的情况,根据规则 2,提出了路径密度这一概念.它表示 LCA 子树中查

询关键字实例在整个 LCA 路径中的关系,即查询关键字的横向关系.

**定义 8(LCA 的实际路径数量 *RPSize*(*T*,*Q*,*u*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,$u\in LCASet(T,Q)$,$RPSize(T,Q,u)$=等于 $u$ 的叶子节点数量.

定义 8 表明了对于关键字查询 $Q$,从 LCA 到叶子节点的路径数量.如图 2 所示,节点 1.4 是关键字查询 $Q$={BigData,Felix,James}的一个 LCA,$RPSize(T,Q,1.4)=3$,说明该 LCA 无冗余信息,而 $RPSize(T,Q,1.3)=5$,说明节点 1.3 存在冗余信息.

**定义 9(路径密度 *LCAPDen*(*T*,*Q*,*v*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,某个 LCA 节点的路径密度 $LCAPDen(T,Q,v)=RPSize(T,Q,v)/MOPSize(Q)$,其中,$v\in LCASet(T,Q)$.

路径密度 $LCAPDen(T,Q,v)$表示了在整个 LCA 子树中,查询关键字实例路径存在的稀疏度.该值越大,表示越稀疏,说明该 LCA 中的冗余信息越多,可能更不符合用户的查询意图.如图 2 所示,节点 1.4 是 $Q$={BigData,Felix,James}的 LCA,它的路径密度 $LCAPDen(T,Q,1.4)=3/3$;节点 1.3 也是 $Q$ 的 LCA,它的路径密度 $LCAPDen(T,Q,1.3)=5/3$,因此 1.4 比 1.3 更符合用户查询意图.

定义 7 和定义 9 分别从纵向和横向的角度来考虑 LCA 与用户查询意图的相关度.但存在另一种情况,如果两个不同 LCA 的边密度和路径密度都相同,我们如何区别它们?XML 文档节点分为连接节点、实体节点、属性节点和值节点[32].不同节点的含义不同,连接节点表示不同实体之间或者实体与属性之间的连接关系,相当于关系数据系统中的连接表,实体节点表示现实世界的对象,相当于关系数据库系统中的实体关系,属性节点相当于关系数据库系统表中的属性,值相当于属性的记录值.在实际查询中,用户往往对实体比较感兴趣,因此在 LCA 排名中,实体节点比连接节点、属性节点和值节点更重要.

LCA 节点与用户查询意图的契合度是综合性的因素,定义 7 和定义 9 考虑了 LCA 节点与查询关键字的纵向和横向关系,定义 10 建立了综合考虑各种情况下的 LCA 评分公式.

**定义 10(*LCAValue*(*T*,*Q*,*v*))**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,$v\in LCASet(T,Q)$,LCA 的分值为

$$LCAValue(T,Q,v)=\alpha\times LCAEDen(T,Q,v)+\beta\times LCAPDen(T,Q,v)+\gamma \tag{2}$$

式(2)中,$\alpha$表示边密度的权重,$\beta$表示路径密度的权重,$\gamma$表示节点类型权重.这里,$\alpha$和$\beta$取值为 1.

$$\gamma=\frac{|LCA|-t}{|LCA|} \tag{3}$$

其中,$|LCA|$表示 LCA 的孩子节点个数,$t$ 表示与根节点类型相同的节点个数.

在如图 2 所示的 XML 文档中,对于查询 $Q$={BigData,Felix,James},计算每个 LCA 所得分数见表 1.

**Table 1** LCAValue of $Q$ corresponding to the XML shown in Fig.2

**表 1** 图 2 所示 XML 文档对应的查询 $Q$ 的 LCAValue

| LCA / 方式 | 1.1.3(1.1.3.1.1.11.1.3.2.2.1.1,1.1.3.1.2.1.1) | 1.2(存在多种组合,其中一种:1.2.1.1,1.2.2.1.1,1.2.3.2.1) | 1.2.4.1(1.2.4.1.1.11.2.4.1.2.1.1,1.2.4.1.2.2.1) | 1.3(1.3.1.1,1.3.2.1.1,1.3.3.2.1) | 1.4(1.4.1,4.2.1.1,1.4.2.2.1) | 1(存在多种组合,其中一种情况:1.3.1.1,1.2.2.1.1,1.1.2.1.1) | 1(存在多种组合,另一种情况:.1.3.1.1.1,1.2.2.1.1,1.3.3.1.1) |
|---|---|---|---|---|---|---|---|
| MOPSize | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| CSize | 11 | 8 | 8 | 8 | 8 | 17 | 13 |
| 叶子节点个数 | 6 | 8 | 3 | 5 | 3 | 25 | 25 |
| $\gamma$ | 0.94 | 0.77 | 0.88 | 0.77 | 0.75 | 0.96 | 0.96 |
| LCAValue | 6.61 | 6.10 | 4.42 | 5.09 | 4.42 | 14.96 | 13.62 |

表 1 中,查询 $Q$ 有 3 个查询关键字,因此节点 1.1.3 的 *MOPSize* 为 3,以 1.1.3 为根的实际路径数(叶子节点数)为 6,关键字 BigData、Felix 和 James 与 LCA 节点 1.1.3 的联系度分别为 $CSize(T,Q,1.1.3,BigData(1.1.3.1.1.1))=3$、$CSize(T,Q,1.1.3,Felix(1.1.3.2.2.1.1))=4$、$CSize(T,Q,1.1.3,James(1.1.3.1.2.1.1))=4$,$\gamma=0.94$,利用公式(2)计算出 $LCAValue(T,Q,1.1.3)=6.61$;同理,节点 1.2 也是 LCA 节点,关键字与 1.2 的联系度之和为 8,叶子节点数是 8,利用公式(2)计算出 $LCAValue(T,Q,1.2)=6.10$;同理,$LCAValue(T,Q,1.4)=4.42$;对于节点 1,存在多种组合情况,表 1 列出

其中的两种,第 1 种对应的节点为 1.3.1.1(Bigdata),1.2.2.1.1(James),1.1.2.1.1(Felix),它们的 *LCAValue*(*T*,*Q*,1)=14.96,而另一种组合的节点为 1.1.3.1.1.1(Bigdata),1.2.2.1.1(James),1.3.3.1.1(Felix),它们的 *LCAValue*(*T*,*Q*,1)=13.62.

表 1 最后一行显示,最符合用户查询意图的节点 1.4 和 1.2.4.1 的 *LCAValue* 值最小,节点 1.3 次之,值较大的节点 1 表明该节点最不符合用户查询意图.因此,可以按照 *LCAValue* 值从小到大排名,把最符合用户查询意图的 LCA 节点优先返回给用户.

## 4 主要算法

### 4.1 计算LCA分值的算法TopLCA-*K*

为 XML 文档树中的每个节点赋予一个体现结构关系的 Dewey 编码,通过该编码仅计算公共前缀即可求出任意两个节点的 LCA.如图 2 所示,节点 1.1.2.1.1 和节点 1.1.3.2.2.2.1,它们的公共前缀是 1.1,因此,节点 1.1 是这两个节点的 LCA.

为了提高计算 *LCAValue* 的效率,对于 XML 文档树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,为每个查询关键字建立倒排索引,并根据它们的 Dewey 编码从小到大排序,按照匹配集合中实例节点个数从小到大的顺序对每个查询关键字进行排序.表 2 对查询 $Q$={BigData,Felix,James}建立了倒排索引表,BigData 关键字实例有 5 个节点,Felix 关键字实例有 7 个节点,James 关键字实例有 6 个节点,按照关键字实例个数从小到大排序,并且每个查询关键字的 Dewey 编码也根据从小到大的顺序排序.此外,还需要为每个非叶子节点建立该子树包含叶子节点数的索引表,见表 3,为 1,1.1,1.1.1,1.1.2 和 1.1.3 建立非叶子节点索引表.

**Table 2** Inverted index table for $Q$={BigData,Felix,James}

**表 2** 查询 $Q$={BigData,Felix,James}的倒排索引表

| 查询关键字 | 查询关键字对应的实例节点 |
|---|---|
| BigData | 1.1.3.1.1.1, 1.2.1.1, 1.2.4.1.1.1, 1.3.1.1, 1.4.1.1 |
| James | 1.1.2.2.1,1.1.3.2.2.1.1,1.2.2.1.1, 1.2.4.1.2.2.1, 1.3.3.2.1, 1.4.2.2.1 |
| Felix | 1.1.2.1.1,1.1.3.1.2.1.1, 1.2.3.2.1, 1.2.4.1.2.1.1, 1.3.2.1.1,1.3.3.1.1,1.4.2.1.1 |

**Table 3** Number of leaf nodes in subtree

**表 3** 子树包含的叶子节点数

| 子树 | 叶子数 | 节点类型 |
|---|---|---|
| 1 | 25 | 连接 |
| 1.1 | 9 | 实体 |
| 1.1.1 | 1 | 属性 |
| 1.1.2 | 2 | 实体 |
| 1.1.3 | 6 | 连接 |

基于倒排表算法计算 *LCAValue* 值的基本思想是利用贪心算法计算所有查询关键字对应节点的各种组合,利用公式(2)计算该组合的 LCA 值并进行排名,输出 TopLCA-*K* 的查询结果给用户.假设有 $t$ 个查询关键字,则该算法时间复杂度为 $O(n^t)$,当查询关键字多并且 XML 文档规模很大时,该算法效率很低.

### 4.2 改进的TopLCA-*K*算法

为了提高 TopLCA-*K* 算法效率以适应大规模 XML 文档的环境,对 TopLCA-*K* 算法进行改进.

**定义 11(节点集 $V'$的中间路径 $MidP(T,V',u)$)**. XML 有向树 $T(V,E,r)$,节点集 $V'\subseteq V$,节点 $u$ 到根节点 $r$ 构成的路径 $p(u,p_1,p_2,\ldots,p_n,r)$把节点集 $V'$分为左右两部分子集 $V'_l$和 $V'_r$,$V'_l\cap V'_r=\varnothing$,$V'_l\cup V'_r=V'$,$\boldsymbol{MidP(T,V',u)=p(u,p_1,p_2,\ldots,p_n,r)}$.

**定义 12(节点集 $V'$中离节点 $u$ 最近节点 $Nearest(T,V',u)$)**. XML 有向树 $T(V,E,r)$,节点集 $V'\subseteq V$,节点 $u\in V$ 但 $u\notin V'$.$V'$中离 $u$ 最近的节点为 $Nearest(T,V',u)$,计算 $Nearest(T,V',u)$的过程如下.

(1) 中间路径 $MidP(T,V',u)$把 $V'$分为 $V'_l$和 $V'_r$,$V'_l\cap V'_r=\varnothing$,$V'_l\cup V'_r=V'$;

(2) $V'_l$中最大 Dewey 编码节点为 $v_{l\_\max}$,$V'_r$中最小 Dewey 编码节点为 $v_{r\_\min}$;

(3) $LCA(u,v_{l\_\max})=x$,$LCA(u,v_{r\_\min})=y$;

(4) 如果 $x$ 是 $y$ 的祖先,则 $v_{r_\min}$为基于 $u$ 的 $V'$的中心节点,即 $Nearest(T,V',u)=v_{r\_\min}$;如果 $y$ 是 $x$ 的祖先,则 $v_{l_\max}$为基于 $u$ 的 $V'$的中心节点,即 $Nearest(T,V',u)=v_{l\_\max}$;

(5) 如果 $x$ 和 $y$ 是同一个节点,则 $v_{l_\max}$为基于 $u$ 的 $V'$的中心节点,即 $Nearest(T,V',u)=v_{l\_\max}$.

图 1 中,设 $V$={3,4,5,7,10,11,13,15,16},中间路径 $p$(12,9,1)把 $V$ 分为 $V_l$={3,4,5,7,10,11},$V_l$中最大编码为 11,$V_r$={13,15,16},$V_r$中最小编码为 13,$LCA$(11,12,)=9,$LCA$(12,13)=1,根据定义 12(4),则有 $Nearest(T,V,12)=11$.

**定义 13(中心节点 *CenterNode*(*T*,*V*,*u*))**. XML 有向树 $T(V,E,r)$,为 $T$ 的每个节点根据深度优先赋予一个 Dewey 编码,节点集 $V'=\{v_1,v_2,\ldots,v_m\}(v_i\in V)$,$V'$的 $m$ 个节点按照 Dewey 编码从小到大排序后,节点 $u\in V$ 但 $u\notin V'$,则 $CenterNode(T,V',u)=Nearest(T,V',u)$.

**性质 1**. XML 有向树 $T(V,E,r)$的 $m$ 个节点 $u_1,u_2,\ldots,u_m$,$x=LCA(u_1,u_2,\ldots,u_m)$;$T$ 中的 $m$+1 个节点 $u_1,u_2,\ldots,u_m,u'$,$LCA(u_1,u_2,\ldots,u_m,u')=y$,如果 $u'$是 $x$ 的后裔,则 $y$ 是 $x$ 本身,否则 $y$ 是 $x$ 的祖先.

**性质 2**. XML 有向树 $T(V,E,r)$的节点 $x$ 和 $y$,如果 $y$ 是 $x$ 的祖先,则 $y$ 子树的叶子节点数大于等于 $x$ 子树的叶子节点数,即 $leafNum(T,y)\geq leafNum(T,x)$.

**性质 3**. XML 有向树 $T(V,E,r)$,节点集 $V'$的 $m$ 个节点按照 Dewey 编码从小到大排序,$V'=\{v_1,v_2,\ldots,v_m\}$ $(v_i\in V,1\leq i\leq m)$,$u\in V$ 但 $u\notin V'$.$CenterNode(T,V',u)=v_j(v_j\in V')$,则 $leafNum(T,LCA(u,v_j))\leq leafNum(T,LCA(u,v_j'))(v_j\neq v_j'$, $v_j'\in V')$.

证明:因为 $v_j=CenterNode(T,V',u)$,根据定义 12 和定义 13,则 $v_j'\neq v_j$,$LCA(u,v_j')=LCA(u,v_j)$或者 $LCA(u,v_j')$是 $LCA$ $(u,v_j)$的祖先.根据性质 2,$leafNum(T,LCA(u,v_j'))\geq leafNum(T,LCA(u,v_j))$. □

图 2 中,按照 Dewey 编码从小到大排序后,$V'$={1.1.2.1.1,1.1.2.2.1,1.1.3.1.1.1,1.1.3.1.2.1.1,1.1.3.2.1.1, 1.1.3.2.2.1,1.1.3.2.2.1.1,1.1.3.2.2.2.1},对于节点 Bob(1.1.3.1.1.2.1),$V'$的中心点 $CenterNode(T,V',1.1.3.1.1.2.1)=$ 1.1.3.1.2.1.1,根据性质 3,$leafNum(T,LCA(1.1.3.1.1.2.1,1.1.3.1.2.1.1))\leq leafNum(T,LCA(1.1.3.1.1.2.1,v))(v\in V'$ 且 $v\neq$1.1.3.1.2.1.1). 例如,$leafNum(T,LCA(1.1.3.1.1.2.1,1.1.3.1.2.1.1))(=2)\leq leafNum(T,LCA(1.1.3.1.1.2.1,1.1.3.2.2.1.1))$ (=6).同理,$leafNum(T,LCA(1.1.3.1.1.2.1,1.1.3.1.2.1.1))(=2)\leq leafNum(T,LCA(1.1.3.1.1.2.1,1.1.3.1.1.1))(=3)$.

在 TopLCA-$K$ 算法中,如何确定 $K$ 是一个重要问题.如对于图 2 中的 $Q$={BigData,Felix,James},$KMS$(BigData) ={1.1.3.1.1.1,1.2.1.1,1.2.4.1.1.1,1.3.1.1,1.4.1.1}共有 5 个节点;$KMS$(Felix)={1.1.2.1.1,1.1.3.1.1.1.1,1.2.3.2.1,1.2.4.1.2.1.1, 1.3.2.1.1,1.3.3.1.1,1.4.2.1.1}共有 7 个节点;$KMS$(James)={1.1.2.2.1,1.1.3.2.2.1.1,1.2.2.1.1,1.2.4.1.2.2.1,1.3.3.2.1, 1.4.2.2.1},共有 6 个节点.按照 LCA 的概念共有 5×7×6=210 种组合情况,但是满足用户查询意图的 LCA 比这个数值要小得多.实际上,真正满足用户意图的 LCA 的个数由 $\min\{|KMS(\text{BigData})|,|KMS(\text{Felix})|,|KMS(\text{James})|\}$的值确定,即具有最少关键字实例的个数就是用户需要寻找的 LCA 个数.

**规则 3**. 对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\ldots,k_m\}$,满足用户查询意图的 LCA 个数,即 $LCANum(T,Q)\leq\min\{|KMS(k_1)|,|KMS(k_2)|,\ldots,|KMS(k_m)|\}$,$k_i\in Q$,$1\leq i\leq m$.

证明:设 $k_i$的$|KMS(k_i)|$具有最小值 $t$,则当 $k_j\in\{k_1,k_2,\ldots,k_{i-1},k_{i+1},\ldots,k_m\}$时,$|KMS(k_j)|\geq t$.对于 $KMS(k_i)$中的任意查询实例节点 $u\in KMS(k_i)$,$LCAValue(T,Q,v)$的最小值为 $\alpha$,$v$ 为包含了 $u$ 的一个 LCA,该 LCA 是满足用户查询意图的 LCA.由于$|KMS(k_i)|$的值为 $t$,因此这样的 LCA 不多于 $t$ 个,即 $LCANum(T,Q)\leq\min\{|KMS(k_1)|,|KMS(k_2)|,\ldots,|KMS(k_m)|\}$. □

备注:如果多个 $LVAValue(T,Q,v)$的值相同,则看作是同一个 LCA.

利用规则 3 能够解决过滤语义,如 SLCA、MLCA 和 ELCA 等查不全的问题.例如,针对图 2 中的查询 $Q$={BigData,Felix,James},按照 SLCA 语义,不能返回 1.2 节点,但是根据规则 3,能够把节点 1.2 作为满足查询意图的一个节点返回给用户.

规则 3 确定了 TopLCA-$K$ 算法中 $K$ 的值,利用定义 13 确定搜索倒排表的起始位置,利用性质 3 能够对搜索

的倒排表进行剪枝,从而提高算法效率.

改进后的 TopLCA-$K$ 算法的思想是:首先建立每个查询关键字的倒排表,对查询关键字实例的个数从小到大排序,并把每个倒排表中的 Dewey 编码按照从小到大的顺序排序,逐层建立中心位置索引,根据中心位置索引逐层寻找 LCA 并进行评分,一直完成最少个数的查询关键字实例的搜索.

建立中心位置索引 CI(center index)的方法是:对于 XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\dots,k_m\}$,查询关键字 $k_1$ 的实例 $KMS(k_1)=\{a_{[1,1]},a_{[1,2]},\dots,a_{[1,t1]}\}$,查询关键字 $k_2$ 的实例 $KMS(k_2)=\{a_{[2,1]},a_{[2,2]},\dots,a_{[1,t2]}\}$,查询关键字 $k_m$ 的实例 $KMS(k_m)=\{a_{[m,1]},a_{[m,2]},\dots,a_{[m,tm]}\}$.每个查询关键字实例按照 Dewey 编码从小到大排序,即 $a_{[j,1]}\leqslant a_{[j,2]}\leqslant\dots\leqslant a_{[j,tj]}(1\leqslant j\leqslant m)$,且 $t_1\leqslant t_2\leqslant\dots\leqslant t_m$.构建如下的中心位置索引如图 3 所示.第 1 层存储了 $k_1$ 实例,第 2 层存储了 $CenterNode(T,KMS(k_2),a_{[1,j]})$节点,第 3 层到第 $m$ 层存储了 $CenterNode(T,KMS(k_i),CenterNode(T,KMS(k_{i-1}),a_{[i-1,j]}))$节点.

例如,对于图 2 所示的 XML 文档树 $T$ 和查询 $Q$={BigData,Felix,James},按照前面的方法建立的中心位置索引如图 4 所示.

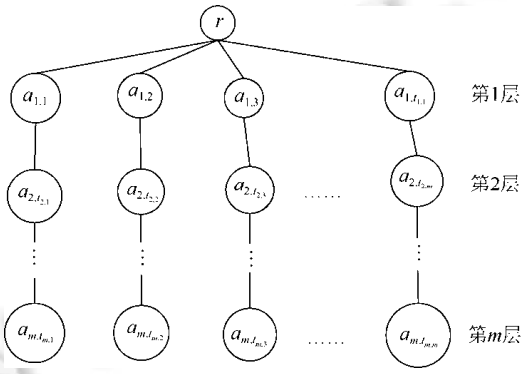



Fig.3 Central location index

图 3 中心位置索引

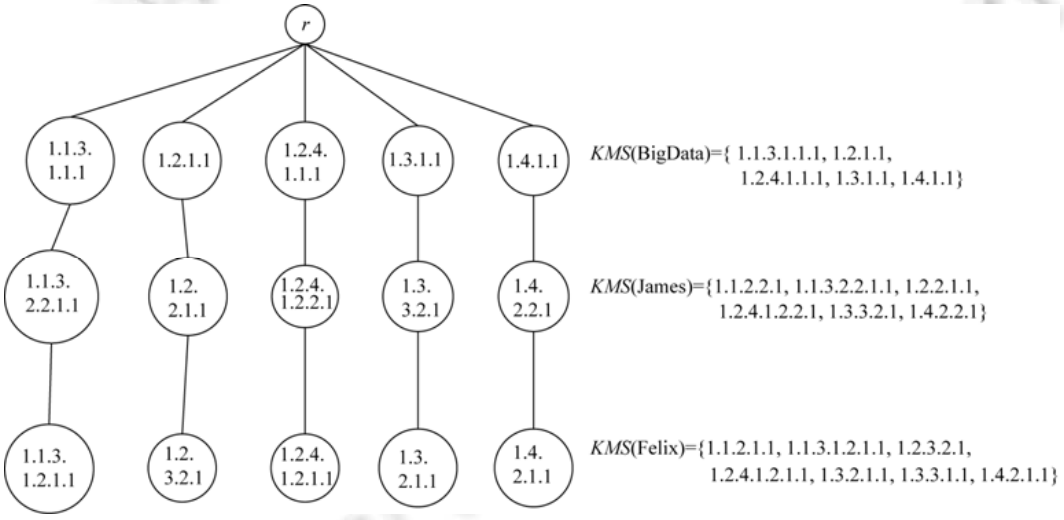



Fig.4 Central location index corresponding to Fig.2 and $Q$={BigData,Felix,James}

图 4 与图 2 和查询 $Q$={BigData,Felix,James}对应的中心位置索引

**推论 1**. XML 有向树 $T(V,E,r)$的关键字查询 $Q=\{k_1,k_2,\dots,k_m\}$,对应的中心位置索引 CI,第 $i$ 层的中心位置 $a_{[i,ci]}$,$LCA(a_{[1,c1]},a_{[2,c2]},\dots,a_{[m,cm]})=C,a_{[1,c1]},a_{[2,c2]},\dots,a_{[m,cm]}$是 CI 树中一条从叶子节点到第 1 层的路径;$LCA(a_{[1,x1]},$

$a_{[2,x2]},\ldots,a_{[i,ci+k]}$)=$R(c_i+k\leqslant t_i,t_i$ 是第 $i$ 层的关键字实例个数),$LCA(a_{[1,x1]},a_{[2,x2]},\ldots,a_{[i,ci-j]})=L(c_i-j\geqslant 0)$,则如下公式成立:

(1) 如果 $leafNum(R)+\sum_{j=1}^{c_i+k} CSize(T,Q,R,a[j,x_j])\geqslant leafNum(C)+\sum_{t=1}^{i} CSize(T,Q,a[t,x_t])$, 则 $LCAValue(T,Q,LCA(R,a_{[i,xi+k+1]}))\geqslant LCAValue(T,Q,LCA(C))$;

(2) 如果 $leafNum(L)+\sum_{j=1}^{c_i-j} CSize(T,Q,L,a[j,x_j])\geqslant leafNum(C)+\sum_{j=1}^{i} CSize(T,Q,L,a[j,x_j])$, 则 $LCAValue(T,Q,LCA(L,a_{[i,xi-j-1]}))\geqslant LCAValue(T,Q,LCA(C))$.

根据推论 1,可以快速寻找出最小 LCA 的值.图 4 显示了图 2、$Q$={BigData,Felix,James}的 CI.通过树的第 1 个分支计算出 LCA 为 1.1.3,该节点有 6 个叶子,关键字与 LCA 的联系度为 11,$LCAValue(T,Q,1.1.3)=(6+11)/3=17/3$;在第 3 层以 1.1.3.1.2.1.1 为出发点,从两边搜索其他 Felix 节点的组合情况,左边的 Felix 节点 1.1.2.1.1,则前面两个节点与 1.1.2.1.1 的 LCA 为 1.1,现在考虑只有 BigData 和 James 两个节点的情况下 $LCAValue(T,Q,1.1)=(9+9)/3=18/3$,则 $LCAValue(T,Q,1.1)\geqslant 18/3>LCAValue(T,Q,1.1.3)(=17/3)$,根据推论 1,可以不用考虑 1.1.2.1.1 左边的节点;再观察右边的情况,右边的 Felix 为 1.2.3.2.1,则前面两个节点与 1.2.3.2.1 的 LCA 为 1,这种情形下叶子节点有 25 个,$LCAValue(T,Q,1)=(25+4)/3>LCAValue(T,Q,1.1.3)(=17/3)$,根据推论 1,可以不用考虑 1.2.3.2.1 右边的节点.因此,利用中心位置索引可以对搜索空间进行快速剪枝,提高了搜索效率.

针对有序 XML 文档树 $T$ 的关键字查询 $Q$,按照深度优先遍历 $T$ 并为每个节点赋予 Dewey 编码,然后建立每个查询关键字 $k_i$ 对应的实例节点倒排表,并按照 Dewey 编码从小到大地对该倒排表进行排序;建立每个子树的叶子节点数列表;建立中心位置 CI.寻找最佳 $K$ 个 LCA 方法,见算法 1 和算法 2.

```
//Location 的数据结构
class Location{
    int level; //表示 IL 中某个节点的层次
    int colmum; //表示 IL 中某层次的某个列
    String dewey; //表示 level 层 colmun 列的 dewey 编码
}
```

**算法 1.** getMinLcaValue(T,Q)//取得最小的 LCA 值.

输入:XML 文档树 T 和查询关键字 Q;

输出:K 个值最小的 LCA.

```
BEGIN
  IL←getCI(T,Q); //产生倒排表,见表 3
  ciT←getCI(IL); //根据倒排表产生中心位置索引表,如图 4 所示
  FOR(i=0;i<ciT 的列数;i++)
     leftLca=getLcaValue(ciT,IL,i,left); //得到第 i 列左边值最小的 LCA
     rightLca=getLcaValue(ciT,IL,i,left); //得到第 i 列右边值最小的 LCA
     IF(leftLca>rightLca)
         return rightLca;
     ELSE IF(leftLca<rightLca)
         RETURN leftLca;
     ELSE
          RETURN leftLca∪rightLca;
     ENDIF
  ENDFOR
 END
```

**算法 2**. getLcaValue(ciT,IL,col,direction); //根据倒排表 IL 和 CI 计算第 col 个的 LCA 值.
输入:中心位置索引树 ciT,关键字倒排表 IL,ciT 的列 col,搜索 ciT 的方向 direction;
输出:LCA 节点和值.

```
BEGIN
        curMinLca←getCIValue (ciT,col); //取得 CI 路径上的 LCA 值
        WHILE(!isEmpty(stkLocation) ) //保存 Location 信息的栈 stkLocation 不为空
            IF(如果 stkLocation 栈顶的 level 到达最高层)
                FOR(根据 direction 方向遍历最高层的所有元素)
                  curLCA←getStkValue(stkLocation); //计算栈中 LCA 的值
                  IF(curLca<curMinLca) //当前 LCA 值<当前最小 LCA 值
                      curMinLca=curLca;
                  ENDIF
                  IF(curLca 满足推论 1) //满足推论 1,不用计算之后的 LCA
                      break;
                  ENDIF
                ENDFOR
                pop(stkLocation);
            ELSE IF(如果 stkLocation 栈顶没有到达最高层)
              curLocation←getTop(stkLoccation); //获得栈顶的层数及位置信息;
              WHILE(curLocation 位置信息不是该层的边界)
                  pop(stkLocation);
                  curLocation←getTop(stkLoccation); //获得栈顶的层数及位置信息;
              ENDWHILE
               stkLocation 栈顶位置信息 location+1;
               根据 level 和 location 修改 stkLocation 栈顶的 Dewey 信息;
               stkLoc 的栈顶位置信息+1,stkElem 对应的值入栈
              WHILE(stkLocation 栈顶的层次数<最大层次数时)
                    下一层的层次数入栈,位置信息 0 入栈 stkLoc;
                    对应的集合的 0 号元素入栈 stkElem
              ENDWHILE
            ENDIF
        ENDWHILE
END
```

## 5 实验分析

我们设计了一套全面的实验来评估本文提出的 TopLCA-*K* 算法的性能,采用真实数据集 SIGMODRECORD、Mondial 和 DBLP 作为测试数据,比较了与 XReal、XReason 和 SLCA 等算法在查全率、查准率和时间性能等方面的效果.所有实验在 CPU 为 Intel 双核 3.6GHz,RAM 为 4GB,操作系统为 Windows 7 的机器上运行,实现语言为 Java SE,通过使用 dom4j-1.6.1.jar 来解析 XML 文档.

### 5.1 数据集与查询测试

我们选取了来自于现实情况的 SIGMODRECORD、Mondial 和 DBLP 作为测试数据集.表 4 显示了这些数据集的统计信息,它们具有不同特征,SIGMODRECORD 节点数和不同标签数最小,Mondial 数据集的节点数处

于中间水平,使用这两个测试数据集的主要目的是测试算法的查全率和查准率.DBLP[①]的规模很大,节点数超过 1 亿,我们选择的测试数据仅仅是其中的一部分,但也保持了一定的规模,该数据集主要测试在数据规模较大情况下各算法的时间性能.本实验没有考虑数据集中 ID/IDREF 的关联情况.

**Table 4** Data sets

**表 4** 测试数据集的基本信息

| 数据集 | 大小(KB) | 节点总数 | 不同标签数 | 平均深度 | 最大深度 |
|---|---|---|---|---|---|
| SIGMODRECORD | 467 | 15 263 | 12 | 4.60 | 6 |
| Mondial | 1 743 | 69 846 | 50 | 3.59 | 5 |
| DBLP | 39 782 | 2 050 096 | 50 | 2.9 | 6 |

针对每个测试数据集,我们选取了具有不同查询意图的测试查询,并且每个查询测试的查询意图是明确的.为每个数据集设计了 7 个测试查询,S1~S7、M1~M7 和 D1~D7 分别表示针对 SIGMODRECORD、Mondial 和 DBLP 数据集的测试查询.具体见表 5.

**Table 5** Keyword query case

**表 5** 关键字查询案例

| 数据集 | 查询编号 | 关键字 | 满足要求的 LCA 总数 |
|---|---|---|---|
| SIGMODRECORD | S1 | author,Anthony,article | 9 |
| | S2 | David,Randy,article | 3 |
| | S3 | Randy,Data | 4 |
| | S4 | author,title,article | 1 504 |
| | S5 | volume,27,article | 4 |
| | S6 | System,initpage,endPage | 285 |
| | S7 | initPage,47,article | 9 |
| Mondial | M1 | Singapore,country | 1 |
| | M2 | United States,border | 1 |
| | M3 | AFG,border,country | 6 |
| | M4 | United,Nations,organization | 24 |
| | M5 | Moutain,Sweden | 4 |
| | M6 | Russion,lake | 8 |
| | M7 | desert,Syrian,country | 1 |
| DBLP | D1 | Frank,Michael,article | 15 |
| | D2 | 2013,Springer,Net | 50 |
| | D3 | ITC ,Machael,inproceedings | 8 |
| | D4 | 688,2011 Security | 13 |
| | D5 | phdthesis,information 2011 | 119 |
| | D6 | California,university,2014 | 169 |
| | D7 | url Vassiliou author | 7 |

### 5.2 查询质量

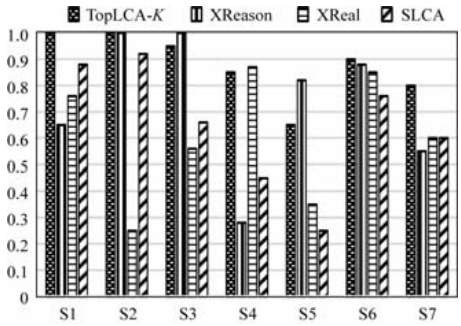
把本文提出的 TopLCA-*K* 算法与已有 XReason、Xreal 和 SLCA 等方法在查准率和查全率进行对比来比较它们的查询质量.

图 5 显示了本文提出的 TopLCA-*K* 与其他方法在查询准确率方面的比较情况.

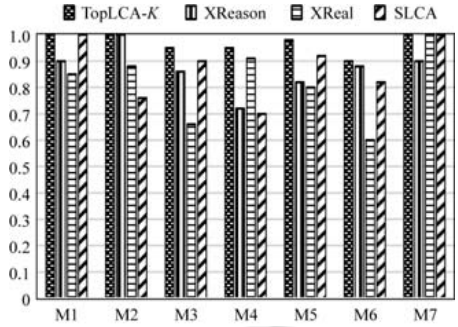
图 5 显示,在查准率方面,一般情况下,TopLCA-*K* 要高于其他 3 种方法,因为该算法考虑了 LCA 中横向关键字密度和纵向关键字密度两方面的情况,因此排在前面的更符合用户查询意图.但在某些查询情况下,准确率也不能达到 100%,主要是因为查询关键字存在一些歧义,例如关于 DBLP 数据集的 D4 查询,查询意图是查询 688 页,2011 年出版的标题中含有 Security 的论文、书籍或者会议出版的文章,但是由于 688 不仅仅出现在 page 中,而且在节点 ee 中也出现了该关键字.同理,2011 不仅仅出现在 year 节点中,而且在 incollection 的 url 中也出现了 2011.

图 6 显示了召回率的比较情况.

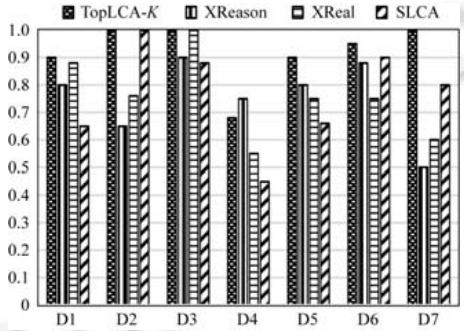
很明显,TopLCA-*K* 的召回率为 100%,不存在漏检问题,其他方法都不能达到 100%,XReal 采用扩展网页排名方法进行排名,在关键字歧义情况下,很容易漏检;SLCA 由于去掉了父 LCA,显然会丢失一些符合要求的结果;而 TopLCA-*K* 返回了所有的 LCA,包括有歧义的 LCA,然后从深度和广度计算 LCA 的值,把排名小的优先返回,当 K 值合理时,将返回所有可能满足查询意图的 LCA.




(a) SIGMODRECORD 数据集上的查准率



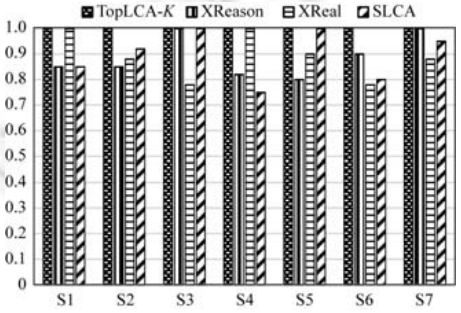

(b) Mondial 数据集上的查准率



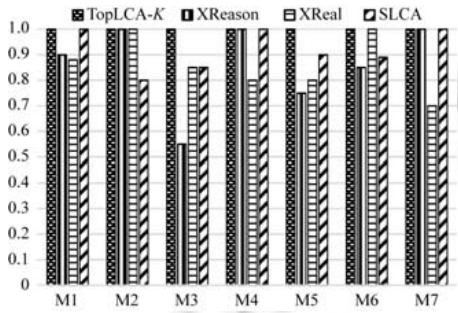

(c) DBLP 数据集上的查准率

Fig.5 Precision rate

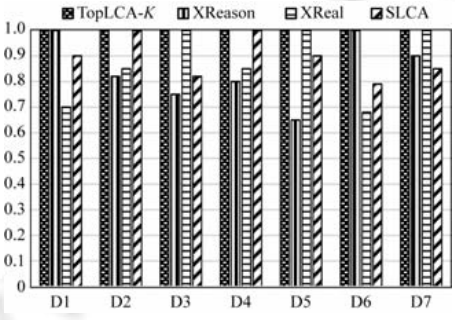
图 5 比较查准率




(a) SIGMODRECORD 数据集上的召回率




(b) Mondial 数据集上的召回率



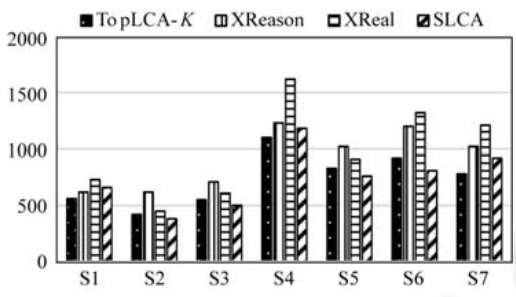

(c) DBLP 数据集上的召回率

Fig.6 Recall rate

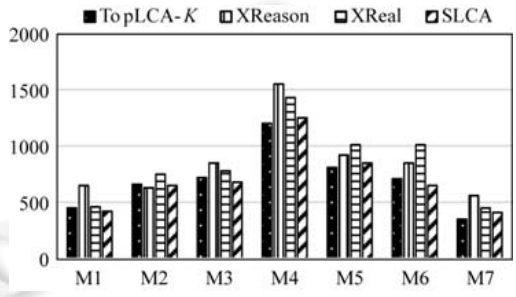
图 6 比较召回率

### 5.3 查询性能

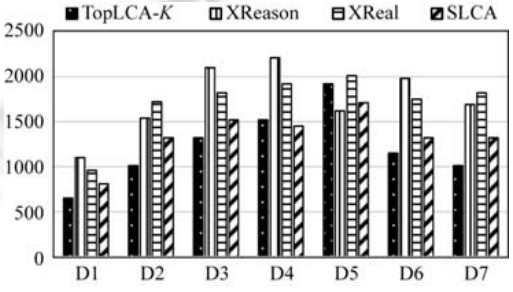
图 7 显示了 3 个数据集上算法运行时间情况,为了准确记录查询时间,每个关键字查询执行 5 次,取它们的平均值作为查询时间,根据规则 3 设置了 TopLCA-*K* 中 *K* 的值.从图 7 可以发现,TopLCA-*K* 的算法时间性能明显优于 XReason 和 XReal,与 SLCA 比较接近.因为 TopLCA-*K* 算法利用 CI 能够对查询空间树进行有效剪枝,例如在 D6 中,共有 8 849 个节点包含关键字 2014,其中 year 节点值有 4 051 个,其他节点,如 ee 包含的 2014 有 520 个.这种情况对 XReason 产生有效结构模式形成了很大干扰,这种情况也对采用 TF*IDF 排名的 XReal 方法形成了很多干扰信息.




(a) SIGMODRECORD 数据集上的算法时间性能




(b) Mondial 数据集上的算法时间性能




(c) DBLP 数据集上的算法时间性能

Fig.7 Time performance

图 7 算法时间性能对比

## 6 结束语

本文首先介绍了 LCA 过滤语义和结果排名方法,指出了在 XML 关键字查询中 LCA 过滤语义存在漏报问题,提出了用户查询意图与查询关键字在纵向和横向方面的两个规则,建立了利用边密度和路径密度对 LCA 节点进行评分的公式,采取中位节点索引 CI 来提高 TopLCA-*K* 算法效率.实验结果表明,本文提出的对 LCA 进行评分排名的方法在查准率和召回率方面效果较好,并且查询时间性能也较好,但需要进一步优化提高.下一步的研究重点考虑当在 LCA 之间存在包含、重复和交叉关系情况时,如何对 LCA 进行排序以及结果展示的问题,同时进一步优化算法.在未来的工作中,将研究如何减少编码长度以及基于新编码方案的 XML 关键字查询处理.

**References**:

[1] Guo L, Shao F, Botev C, Shanmugasundaram J. XRANK: Ranked keyword search over XML documents. In: Proc. of the SIGMOD Conf. 2003. 16–27.

[2] Schmidt A, Kersten M, Windhouwer M. Querying XML documents made easy: Nearest concept queries. In: Proc. of the Int'l Conf. on Data Engineering. IEEE, 2001. 321–329.

[3] Zhang CJ, Wang XL, Zhou AY. XML filtering based-on probabilistic SLCA. Chinese Journal of Computers, 2014,(9):1959–1971 (in Chinese with English abstract).

[4] Bao Z, Ling TW, Chen B, *et al*. Effective XML keyword search with relevance oriented ranking. In: Proc. of the IEEE Int'l Conf. on Data Engineering. IEEE, 2009. 517–528.

[5] Bao Z, Lu J, Ling TW, *et al*. Towards an effective XML keyword search. IEEE Trans. on Knowledge & Data Engineering, 2010, 22(8):1077–1092.

[6] Chen LJ, Papakonstantinou Y. Supporting top-*K* keyword search in XML databases. In: Proc. of the IEEE Int'l Conf. on Data Engineering. IEEE, 2010. 689–700.

[7] Liu Z, Chen Y. Processing keyword search on XML: A survey. World Wide Web-Internet & Web Information Systems, 2011, 14(5-6):671–707.

[8] Liu X, Wan C, Chen L. Returning clustered results for keyword search on XML documents. IEEE Trans. on Knowledge & Data Engineering, 2011,23(12):1811–1825.

[9] Cohen S, Mamou J, Kanza Y, Sagiv Y. XSEarch: A semantic search engine for XML. VLDB Journal, 2003, 45–56.

[10] Li Y, Yu C, Jagadish HV. Schema-free XQuery. VLDB Journal, 2004, 72–83.

[11] Xu Y, Papakonstantinou Y. Efficient keyword search for smallest LCAs in XML databases. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Baltimore: DBLP, 2005. 527–538.

[12] Li G, Feng J, Wang J, *et al*. Effective keyword search for valuable lcas over XML documents. In: Proc. of the 16th ACM Conf. on Information and Knowledge Management. ACM, 2007. 31–40.

[13] Chen ZY, Wang X, Tang X. Efficiently computing RKN for keyword queries on XML data. Journal on Communications, 2014, 35(7):46–55 (in Chinese with English abstract).

[14] Termehchy A, Winslett M. Using structural information in XML keyword search effectively. ACM Trans. on Database Systems, 2011,36(1):1–39.

[15] Li G, Li C, Feng J, *et al*. SAIL: Structure-aware indexing for effective and progressive top-*k*, keyword search over XML documents. Information Sciences, 2009,179(21):3745–3762.

[16] Li J, Liu C, Zhou R, *et al*. Top-*k* keyword search over probabilistic XML data. In: Proc. of the IEEE Int'l Conf. on Data Engineering. IEEE, 2011. 673–684.

[17] Amer-Yahia S, Lalmas M. XML search: Languages, INEX and scoring. ACM SIGMOD Record, 2006,35(4):16–23.

[18] Hristidis V, Papakonstantinou Y, Balmin A. Keyword proximity search on XML graphs. In: Proc. of the Int'l Conf. on Data Engineering. IEEE, 2003. 367–378.

[19] Zhou J, Bao Z, Ling TW, *et al*. MCN: A new semantics towards effective XML keyword search. Lecture Notes in Computer Science, 2009,5463:511–526.

[20] Zhou R, Liu C, Li J. Fast ELCA computation for keyword queries on XML data. In: Proc. of the Int'l Conf. on Extending Database Technology. ACM, 2010. 549–560.

[21] Zhou J, Wang W, Chen Z, *et al*. Top-down XML keyword query processing. IEEE Trans. on Knowledge & Data Engineering, 2016, 28(5):1340–1353.

[22] Dimitriou A, Theodoratos D, Sellis T. Top-*k*-size keyword search on tree structured data. Information Systems, 2014,(47):178–193.

[23] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword search in relational databases. VLDB Journal, 2002,26(2):670–681.

[24] Li QS, Wang QY, Wang S. Query understanding for XML keyword search. Ruan Jian Xue Bao/Journal of Software, 2012,23(8): 2002–2017 (in Chinese with English abstract). http://www.org.cn/1000-9825/4122.htm [doi: 10.3724/SP.J.1001.2012.04122]

[25] He H, Wang H, Yang J, *et al*. BLINKS: Ranked keyword searches on graphs. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Beijing: ACM, 2007. 305–316.

[26] Aksoy C, Dimitriou A, Theodoratos D, *et al*. XReason: A semantic approach that reasons with patterns to answer XML keyword queries. In: Database Systems for Advanced Applications. Berlin, Heidelberg: Springer-Verlag, 2013. 299–314.

[27] Aksoy C, Dimitriou A, Theodoratos D. Reasoning with patterns to effectively answer XML keyword queries. VLDB Journal, 2015, 24(3):441–465.

[28] Li J, Liu C, Zhou R, *et al*. Suggestion of promising result types for XML keyword search. In: Proc. of the Int'l Conf. on Extending Database Technology. ACM, 2010. 561–572.

[29] Zhou J, Bao Z, Wang W, *et al*. Efficient query processing for XML keyword queries based on the IDList index. VLDB Journal, 2014,23(1):25–50.

[30] Liu X, Wan C, Liu D. Keyword query with structure: towards semantic scoring of XML search results. Information Technology & Management, 2016,17(2):151–163.

[31] Nguyen K, Cao J. Top-*k*, answers for XML keyword queries. World Wide Web-Internet & Web Information Systems, 2012,15(5-6): 485–515.

[32] Liu Z, Chen Y. Identifying meaningful return information for XML keyword search. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2007. 329–340.

**附中文参考文献**:

[3] 张晨静,王晓玲,周傲英.基于概率 SLCA 的 XML 过滤.计算机学报,2014,(9):1959–1971.

[13] 陈子阳,王璿,汤显.面向 XML 关键字查询的高效 RKN 求解策略.通信学报,2014,35(7):46–55.

[24] 李求实,王秋月,王珊.XML 关键词检索的查询理解.软件学报,2012,23(8):2002–2017. http://www.org.cn/1000-9825/4122.htm [doi: 10.3724/SP.J.1001.2012.04122]

**覃遵跃**(1974－),男,湖南张家界人,博士,副教授,主要研究领域为数据库技术.

**徐洪智**(1974－),男,副教授,主要研究领域为嵌入式系统,并行计算.

**汤庸**(1964－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为协同工作,数据库.

**黄云**(1976－),男,博士,副教授,CCF 专业会员,主要研究领域为数据挖掘,智能信息计算.