一种基于XML文档关键字检索的结构索引

娄 颖 李战怀 郭文琪 陈 群 韩 萌
(西北工业大学计算机学院 西安 710129)

摘 要 XML数据索引对其检索效率有较大的影响。在深入分析现有XML结构索引之后,结合XML文档特点,提出了一种基于关键字检索的结构索引——LSS(Level Structure Summary)。LSS采用了把具有相同标签路径的结点进行合并的策略,具有高效判断结点之间同构异构关系的能力。实现了LSS索引生成算法CSCAN,并在LSS索引的基础上设计了XML关键字检索算法LSSearch。该算法依据LSS索引,将各个关键字的原始倒排表集合分拆成不同类型的子集合,最后在所有子集合上进行查询。实验结果表明,LSS可以帮助减少XML文档中关键字倒排表的规模,提高检索效率。

关键词 XML,关键字检索,索引,倒排表
中图法分类号 TP311 **文献标识码** A

Structure Summary for Keyword Search over XML Documents

LOU Ying LI Zhan-huai GUO Wen-qi CHEN Qun HAN Meng
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract The index of XML Data is crucial for retrieval efficiency of XML document. After analysis of existing XML structure summaries, this paper proposed a structural summary over keyword search called LSS combining the XML document. LSS merges the nodes in the XML tree with the same label path so as to determine nodes' homogeneity and heterogeneity efficiently. This paper implemented LSS constructing algorithm called CSCAN, and designed a XML keyword retrieval algorithm called LSSearch based on LSS. This algorithm split keywords' inverted list into different type subsets, finally retrieved to get all results quickly on these subsets. Experimental results demonstrated that LSS can help to reduce the size of the keyword inverted list in XML document dramatically and improve retrieval efficiency.

Keywords XML, Keyword search, Indices, Inverted list

1 引言

由于具有灵活性和可扩展性的特点,XML[1]已经成为互联网上数据存储和交换的重要标准之一。如何能在海量的XML数据中进行高效的信息检索,成为XML数据管理的重要问题。

目前,基于XML关键字检索的方法大都是基于LCA(Lowest Common Ancestor)的思想,首先定位包含所有关键字LCA结点,然后返回以该LCA结点为根结点的片段。XML数据索引对检索效率有较大的影响,如传统信息检索中的关键字倒排索引(Inverted Index),通过记录每一个关键词在XML文档出现位置的编码信息,从而快速地计算出LCA结果。但是,随着XML数据规模的不断增大,被检索关键字的倒排索引也随之迅速递增,那么在大规模倒排索引中进行LCA计算的效率显得越来越重要。

为了解决这个问题,本文设计一种基于XML关键字检索的结构索引——LSS。

本文的主要工作总结如下:

1)提出了基于分层的结构索引LSS(Level Structure Summary),并设计和实现了相关的算法。

2)分析了新索引的可行性和优点,并对索引的大小进行了分析。

3)实现了基于LSS索引的LSSearch检索算法,同时实现了经典的LCA算法,并进行了大量的实验,比较和分析了LSS在XML关键字检索上的性能。

本文第2节介绍关于基于关键字检索的相关工作,重点介绍文献[2]中的结构索引方法和思想,深入分析目前比较流行的LCA计算方法;第3节描述LSS的新思路,给出LSS的结构模型;第4节介绍LSS的生成算法,并其进行详细的分析;第5节介绍基于LSS索引的检索算法LSSearch,并分析

到稿日期:2010-01-10 返修日期:2010-04-12 本文受863国家重点基金项目(2009AA1Z134),国家自然科学基金(60803043,60720106001)资助。

娄 颖(1980—),男,博士生,主要研究方向为信息检索和XML数据管理等,E-mail:louying@mail.nwpu.edu.cn;李战怀(1961—),男,博士生导师,主要研究方向为数据管理、数据挖掘等;郭文琪(1986—),女,硕士生,主要研究方向为信息检索和XML数据管理等;陈 群(1976—),男,教授,博士生导师,主要研究方向为XML数据管理、RFID数据管理等;韩 萌(1984—),男,硕士生,主要研究方向为信息检索和XML数据管理等。

LSSearch 算法的复杂度；第 6 节给出相关实验结果；最后给出结论。

2 相关工作

当前 XML 信息检索可以通过两种方法实现：基于结构化语言和基于关键字。基于结构化语言的查找方法利用结构加上的关键字进行 XML 信息的搜索，可以对 XML 树结构或图结构进行约减，使得约减后的结构只维持不同的路径信息，就是结构索引技术。基于关键字检索的查找方式采用对 XML 源数据进行适当编码以及倒排索引来提高检索效率。

结构索引(Structure Summary)主要有以下几种技术：DataGuide[3]，1-index[4]，A(k)-index[5] 以及 D(k)-index[2]。提出这些结构索引，是为了帮助估计正则路径表达式的大致结果。D(k)-index 对本文有一定启发，我们在这里介绍 D(k)-index 的相关内容。

D(k)-index 是基于双似关系的动态结构索引，能根据当前查询语句的负载来动态地调节索引的结构。它是在 A(k)-index 的基础上吸收了 APEX 中动态反映查询分布的优点，并将二者进行了结合，其核心思想是动态调整索引结构中每一个结点的局部相似度。

基于 XML 关键字检索的方法大都是基于 LCA 的思想，目前比较流行的算法主要有 ELCA[6]，MLCA[7]，SLCA[8] 以及 VLCA[9]。我们着重介绍 ELCA，SLCA 和 VLCA。

定义 1($ELCASet$) 给定关键字集合 $K=\{k_1,k_2,\cdots,k_m\}$，一个 XML 文档 D，以及 K 中每一个关键字在 D 中对应的结点集合为 $S=\{S_1,S_2,\cdots,S_m\mid k_1\in S_1,k_2\in S_2,\cdots,k_m\in S_m\}$，则关键字 K 在 D 上的 $ELCASet$ 定义如下：

$$ELCASet=\{v\mid k_1\in S_1,k_2\in S_2,\cdots,k_m\in S_m(v=LCA(k_1,k_2,\cdots,k_m)\wedge\forall i(1\leqslant i\leqslant m))\rightarrow\exists(x\in LCA(S_1,S_2,\cdots,S_m)\wedge child\_path(v,k_i)\leqslant x)\}$$

式中，$child_path(v,k_i)$是指 v 到 k_i 路径上 v 的孩子结点。

文献[6]中的 XRANK 提出 ELCA(Exclusive LCA)借助 LCA 的思想来解决关键字完全匹配问题。从定义 1 可以看出，ELCA 的语义是指从一个 LCA 结点到 $k_1,k_2,\cdots,k_m$ 的路径结点集中不能出现另一个 LCA 结点。文中提出了 3 种算法，其中 DIL((Dewey Inverted List)算法的复杂度为 O($kd|S|$)。

定义 2($SLCASet$) 给定关键字集合 $K=\{k_1,k_2,\cdots,k_m\}$，一个 XML 文档 D，以及 K 中每一个关键字在 D 中对应的结点集合为 $S=\{S_1,S_2,\cdots,S_m\mid k_1\in S_1,k_2\in S_2,\cdots,k_m\in S_m\}$，则关键字 K 在 D 上的 $SLCASet$ 定义如下：

$$SLCASet=\{v\mid v=LCA(S_1,S_2,\cdots,S_m)\wedge\rightarrow\exists u,v\prec u,u\in LCA(S_1,S_2,\cdots,S_m)\}$$

文献[8]提出了 SLCA(Smallest Lowest Common Ancestor)，即最小 LCA 的概念。从定义 2 可以看出，$SLCASet$ 实际上是在 $ELCASet$ 的集合中去除可以构成父子或者祖先子孙关系的父子或祖先 LCA 结点。文中提出了 3 种计算 SLCA 的算法，其中 IL 算法在关键字倒排表大小相差较大时表现出较佳的性能，算法复杂度为 O($kd|S_1|\log|S|$)。

定义 3(同构/异构关系) 给定两个结点 u 和 v，$w=LCA(u,v)$，如果在从 w 到结点 u 和 v 的路径上的所有结点中，除 u 和 v 本身的标签可能相同外没有其它任意两个结点的标签相同，则称 u 和 v 是同构的；反之，u 和 v 是异构的。

定义 4($CVLCA$) 给定关键字集合 $K=\{k_1,k_2,\cdots,k_m\}$，w 为其 LCA，如果 K 中任意两个结点是同构的，则 w 是一个 VLCA；如果存在另一个 VLCA 结点 $w'=VLCA(k_1',k_2',\cdots,k_i,k_m')$，$w$ 是 w' 的子孙结点，则 w 是一个 CVLCA。

文献[9]提出了 VLCA(Valuable Lowest Common Ancestor)的概念。为了快速计算 VLCA，文中又提出了 CVLCA(Compact VLCA)的概念，如定义 4 所示，最后给出了一种栈算法 VLCAStack，算法复杂度为 O($kd|S|+kd\log(kd)*|S_1|$)。从定义 4 可以看出，CVLCASet 是在 ELCASet 的基础上去除非 VLCA 的结点。

从以上定义可以看出 $ELCASet$，$SLCASet$ 和 $CVLCASet$ 的关系：$ELCASet\supseteq SLCASet$，$ELCASet\supseteq CVLCASet$。

3 LSS(Level Structure Summary)介绍

XML 数据通常是以一棵有序的标签树 $T=(r,V,E,L,\lambda)$为模型的，其中 r 是 XML 文档树的根结点；V 是结点集合，包括元素结点、属性结点和文本结点；E 是 XML 文档中各个结点之间的关系集合；λ 用来标识结点的名称；L 是所有结点名称集合。对于每一个结点 $v\in V$，都有 $\lambda(v)\in L$。我们采用二元组(Deweyid，Label)标识一个结点，其中 Deweyid 代表结点的 Dewey 编码，Label 代表结点的名称。我们将一个元素结点的属性结点视为一个该元素结点的子元素结点。例如，图 1 表示一个简单的 XML 文档树，矩形代表 XML 文档中的元素结点，椭圆代表文本结点，其他的代表属性结点，根节点 r 表示为("0","conference")。

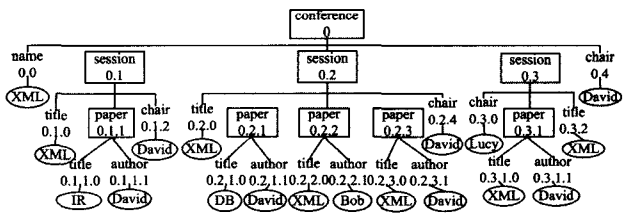


图 1 带有 Dewey 编码的 XML 文档树

3.1 LSS 的思路

为了提高 XML 关键字检索的检索效率，通常从两方面入手：改进查找索引和改进倒排索引。一方面，对查找索引进行压缩，如 SLCA 中对二进制形式的 Dewey 编码进行压缩，甚至还可以设计较好的语义规则以及满足此语义规则的编码方式来提高性能，如 VLCA 中提出的同构和异构规则。另一方面，通过对倒排索引进行某种扩展或者压缩来改进。结构索引在一定程度上降低了 XML 文档树中的结点个数和文档树的大小。如果在关键字检索中利用结构索引的优势和特点，将能够提高检索性能。目前，现有的相关研究还没有将结构索引和关键字检索联合起来。

各种 LCA 算法的复杂度都与关键字的倒排表(Inverted List)有紧密关系，从 DIL 中 O($kd|S|$)提高到 SLCA 中的 O($kd|S_1|\log|S|$)，效率上都是尽量减少关键字倒排表的规模大小。那么，根据计算 LCA 的方法，如果能有效地减少各个关键字倒排表的大小，如将算法中的 $|S_i|(1\leqslant i\leqslant n)$减少到 $|S_i'|(|S_i'|\ll|S_i|)$，则计算 LCA 的时间就可以大幅度减少。经过观察 XML 文档，发现某一"层"中会出现大量相同标签的结点，这些结点大都用于描述某一类实体。按照此特征，我

们采用结构索引的技术，对满足条件的同层结点进行合并，来约减 XML 文档树的大小。

定义 5(结点类型)　XML 文档中一个结点 n 的结点类型为从根结点到 n 的标签路径。一个结点的结点类型表示为 ntype。T 中任意两个结点 u，v，如果其结点类型相同，则为等价结点。

例如，图 1 中的 u=("0. 1. 1. 0","title") 和 v=("0. 2. 1. 0","title")，ntype(u)=conference/session/paper/title，ntype(v)=conference/session/paper/title，由于 u 和 v 的结点类型相同，因此 u 和 v 是等价结点。

取 XML 文档树中的根结点所在的层为 1，那么称结点类型 p 中与第 i 层结点对应的字符串为 p 的第 i 层祖先结点的标签，表示为 $pre_p(i)$，结点所在的层为结点类型的长度，记为 lp。

结点类型之间的基本关系如下：

设 u 和 v 表示两个结点，结点类型分别为 ntype_u 和 ntype_v，长度分别为 $lp(\text{ntype}_u)$ 和 $lp(\text{ntype}_v)$。

性质 1(相等关系)　如果 $lp(\text{ntype}_u)=lp(\text{ntype}_v)$，并且从左到右逐层比较结点类型各层的标签字符串都相等，则 u 和 v 是等价的，记为 $\text{ntype}_u=\text{ntype}_v$。

性质 2(包含关系)　如果 $lp(\text{ntype}_u)<lp(\text{ntype}_v)$，且 $pre_u(lp(u))=pre_v(lp(u))$，则称 v 包含 u，或者 u 包含在 v 中。

3.2　LSS 的结构

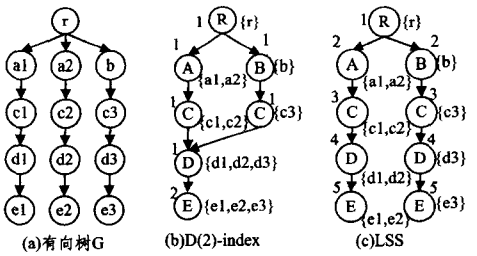
LSS 是将 XML 文档树中的等价结点进行合并而生成的索引，索引中的每一项代表一个关键字对应的所有等价结点的集合，称之为一个 GN。组结点集合中的每一个元素都是由二元组〈GDeweyid，CSet〉组成的，其中 GDeweyid 表示等价结点所属的 Deweyid 编码，CSet 表示 XML 文档树中具有相同 GDeweyid 的同一种关键字的集合。

定义 6(LSS)　LSS 是一棵对应于 XML 文档树的有序的标签树 $LT=\{GV,GE\}$，满足下列条件：

1)GV 代表 LSS 中所有组结点 GN 的集合，GV 集合中的每一个元素都是由二元组〈$termName$，〈$GDeweyid$，$CSet$〉〉构成。

2)GE 代表 GV 中的各个 GN 之间的关系，如果一个组结点 GN 的结点类型包含另一个组结点 GN' 的结点类型，那么 GN 和 GN' 之间存在父子或者祖先子孙关系。

定义 6 给出了 LSS 的定义。结合 $D(k)$-index 的思想，LSS 其实就是将 $D(k)$-index 中 k 设置成 XML 文档树的深度，索引中每一个结点的局部相似度(local similarity)就是该结点所处的层，如图 2 所示。



(a)有向树G　(b)D(2)-index　(c)LSS

(1)每一个标签结点都有一个局部相似度和候选结点集；(2)每一个标签结点旁边的数字表示该结点的实际局部相似度

图 2　一个关于 $D(k)$-index 和 LSS 的例子

4　构建 LSS 结构索引

CSCAN 算法采用将 XML 文档解析和 LSS 索引构建有效结合起来的策略，即在使用 SAX 解析 XML 文档的同时构建 LSS 索引。

当 SAX 处理 XML 文档中的元素时，检测当前元素的结点类型，若 B+树(用来记录当前文档中所有可能的标签路径)中还没有此结点类型的信息，则向 B+树中添加该结点类型的 GDeweyid 编码信息，否则读取该结点类型的 GDeweyid 信息。并将该元素结点的结点类型编码设置成 GDeweyid。若该元素有属性，那么将属性按照元素处理。当 SAX 处理元素结束后，在 LSS 索引中查找一个与当前关键字相同的信息。若没有找到，则创建一个新的结点，插入到 LSS 索引中，此结点的关键码是当前元素结点的标签名，同时新建一个 GN，将当前元素结点放入 GN 中。若找到了一个结点，则更新此结点的 GN 集合信息。当 SAX 处理属性的属性值或者元素的文本值时，首先对该文本进行有效的分词并且过滤无意的单词，然后对于处理后所产生的每一个单独的关键字，按照元素的方式处理。具体如算法 1 所示。

例如，在算法 1 中，当处理第二层中的 3 个"session"元素结点时，由于它们的结点类型(ntype)都是 conference/session，所以在第二层的处理中，将它们合并成一个组结点〈session，〈0. 1，"0. 1，0. 2，0. 3"〉〉。当解析到所有"paper"结点时，所有的"paper"彼此之间是等价的，所以它们也被合并成一个组结点〈paper，〈0. 1. 0，"0. 1. 1，0. 2. 1，0. 2. 2，0. 2. 3，0. 3. 1"〉〉。同时由于"session"组结点的结点类型包含在"paper"的结点类型中，故"session"和"paper"之间存在着父子关系。依此类推，直到 XML 文档解析结束。

算法 1　CSCAN 算法

```
Begin
SAX2XMLReader parser;SAX2Handler handler;
    parser. parse(XMLDocument);
    handler. StartDocument(){
        startElement:
            if(the B+ tree not contains the current node's ntype)
                new node's ntype of GDeweyid gid into B+ tree;
            else
                read gid from B+ tree;
            node. GDeweyid=gid;
            for atti in node's attributes
              Treat atti as element to process
        endElement:
            if(LSS. find(node. name))
                  update the node's GN;
            else
                  LSS. insert(〈node. name,new GN〉);
                  GN. GDeweyid=node's GDeweyid;
                  GN. CSet. insert(node. deweyid);
        Character:
            Vector words=Splitword();
            For(each word in words)
                Goto:startElement;Goto:endElement;
}handler. EndDocument();
Ends
```

5 基于 LSS 的查询处理

基于 LSS 结构索引的关键字检索，首先根据用户输入的关键字集合，从 LSS 中取出关键字集合对应的各自 GDeweyid 集合 invertGList[i]，以及每一个关键字对应的每一个 GDeweyid 的 CSet 集合 GMapList[i][j]。然后，在 invertGList 上采用 XRANK 中 DIL(我们对其稍有修改)算法计算出所有的 GLCAS。根据生成每一个 GLCA 的关键字的 GDeweyid 信息，取出关键字对应的 GDeweyid 的 CSet 集合，然后在每个关键字的 CSet 集合上采用快速查找算法得出所有的 LCA，并根据 CVLCA 的判断方法得到最终的 CVLCA 集合。具体算法如算法 2 所示。

假如用户输入的关键字集合为{XML,David}，用来查找名叫 David 的作者写的关于 XML 的文章或者名叫 David 的专家主持的 XML 会议。第一步计算得到 GLCAS={0(0.0,0.2),0.1(0.1.0,0.1.2),0.1.1(0.1.1.0,0.1.1.1)}，然后，从 LSS 中得到生成 GLCA 的结合集合。例如编码为 0.1.1 的 GLCA 中，S[1]={0.2.2.0,0.2.3.0,0.3.1.0}，S[2]={0.1.1.1,0.2.1.1,0.2.3.1,0.3.1.1}，然后从 S[1]和 S[2]中计算出最终的 CVLCA={0.2,0.2.3,0.3.1}，编码为 0 的 GLCA 最终得到的 CVLCA={0}；编码为 0.1，最终得到的 CVLCA={0.1,0.2}，所以最终的 CVLCA={0,0.1,0.2,0.2.3,0.3.1}。再比如用户输入的关键字集合为{DB,XML}，用来查找关于 DB XML 方面的文章，第一步得到的 GLCAS={0.1.1.0(0.1.1.0,0.1.1.0)}，最终得到的 LCA 是{0.2}。由于此 LCA 不是一个合法的 VLCA，因此最终的结果为空。

算法 2 LSSearch 算法

```
Procedure LSSearch(k1,k2,k3,…,km) retruns VLCAS;
  //k1,…,km are the query keywords
  //invertGList[i] is the GDeweyid inverted List for keywords ki from LSS
  //GMapList[i][gid] is the CSet of gid for keywords ki from LSS
  //S[i] is the partly inverted List in XML fro keywords ki orderd by size
    GLCAS=null;VLCAS=null;
  //use the DIL algorithm idea within invertGList so as to get all GLCAS;
  GLCAS=DIL'(k1,k2,…,km,invertGList[])
  for each glca in GLCAS{
      for each keyword get S[i] from GMapList[i][glca[i].gid]
      while( there are more nodes in S[1]){
          Read P nodes of S[1] into buffer B
          for i=2 to m
              B=get_lca(B,S[2])
          VLCAS.insert(B);B={};
      }//end while
  }//end for
  output VLCAS

subroutine get_lca(S1,S2)
  Result={}
  for each node v in S1{
      u=lca(v,lm(v,S2));w=lca(v,rm(v,S2));
      if(u isdescendant(w) )
          if u is cvlca then Result∪{u}
          else if w is cvlca then Result∪{w}
  }//end for
  return Result;
```

我们分析一下 LSSearch 算法的复杂度。由文献[6]得知，DIL 算法的复杂度为 $O(kd|S|)$，那么 LSSearch 算法第一部分的复杂度(计算 GLCAS)为 $O(kd|S_g|)$，其中 S_g 是指 |invertGList[i]| 的最大值。由文献[8]得知，第二部分的复杂度为 $O(mkd|S_1|\log|S|)$，其中 m 为第一步计算出的 GLCA 的个数，S_1 为生成 GLCA 的关键字组结点在原始倒排索引中对应的部分集合大小的最小值，S 对应最大值。k 为关键字集合大小，d 对应 XML 文档树的深度，所以 LSSearch 算法的最终复杂度为 $O(kd|S_g|+mkd|S_1|\log|S|)$。

6 实验与分析

为了检验本文提出的算法的性能，进行了大量的实验。本文实现了文献[6]中的 IL 算法、文献[9]中的 VLCAStack 算法以及基于本文 LSS 索引的 LSSearch 算法。实验数据采用来自华盛顿大学的 XML 数据仓库中 Sigmod Record，NASA，TREEBANK 以及 DBLP 数据集[10]。这 4 个真实数据集的大小分别为 704kB，25MB，84MB，134MB。

所有的实验都是在 HP dx2355 business PC 机上实现的，机器的 CPU 为 AMD Athlon(tm) Dual Core 5000B，主频为 2.61GHz，物理内存为 1.87G，所采用的操作系统是 Windows XP SP3。所有的算法都是用 java 语言实现的，采用 SAX2 API 作为 XML 的解析器，使用 MYSQL 5.2 来存储索引。

6.1 索引大小

图 3 描述了 LSS，Raw Data 在 NASA，TREEBANK，DBLP3 个数据集上的空间占用情况。

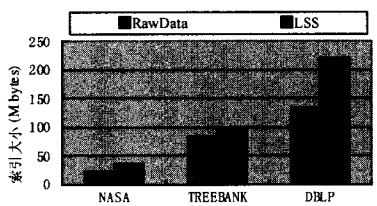


图 3 索引大小比较

由图 3 可以看出，TREEBANK 数据集对应的 LSS 和原数据相差最小，其原因在于 TREEBANK 的层数较大，分布在每一层上的数据比较平均。NASA 次之，DBLP 最差。

6.2 压缩率

图 4 描述了 LSS 结构索引对于 2 个不同深度的数据集在每一层上的压缩率。压缩率的计算公式如式(1)所示。

$$(\text{压缩率})=1-\frac{\text{压缩后的结点数}}{\text{原始结点数}} \tag{1}$$

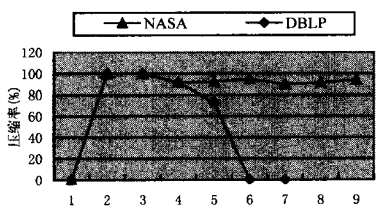


图 4 LSS 在不同数据集每一层上的压缩比例

从图 4 可以看出，DBLP 数据集上，由于该数据集描述的是计算机科学的主要期刊和论文集的信息，那么在数据集的第 2 层中大部分都是关于 artile，proceedings 和 inproceedings

的信息，同时 artile，proceedings 和 inproceedings 的结构又很相似，所以第 2，3，4 层压缩率较高，在第 2，3 层都达到了 99% 以上，第 6 层上的压缩率接近于 0；NASA 数据集上，除第一层，其余各层压缩率都较高；由于 XML 数据中第一层为一个结点，压缩后也为一个结点，因此第一层的压缩率为 0。

6.3 检索性能

为了检验本文提出的 LSS 结构索引的检索性能，将 LSSearch 算法和 SLCA，CVLCA 在不同数据集（Sigmod Record，DBLP）上进行比较，分析它们的检索时间。SLCA 算法是查询效率较高的算法，本文算法的查询语义和查询结果与 CVLCA 相同。

对于每一个数据集，选取 5 个不同的关键字集合，其对应的关键字个数是从 2 个到 6 个，每一个数据集上采取 2 种不同频率的关键字集合，分别是低频率和高频率关键字集合。例如图 6 中的 QSl6 表示在 Sigmod Record 数据集上包含 6 个关键字，并且这 6 个关键字都是低频率关键字。查询时间的统计是从输入关键字到查询结果的返回。

从图 5、图 6 可以看出，本文提出的 LSSearch 算法在大部分情况下优于 CVLCA 算法，原因在于 CVLCA 算法是一种栈算法，计算过程需要将每一个关键字倒排表都扫描一遍，尤其是当关键字的倒排表较大时，CVLCA 表现出较差的性能，如 DBLP 数据集中 QDh1，QDh2 和 QDh4。而 LSSearch 算法可以高效地判断 VLCA，因为计算出的 GLCA 的各个参与关键字的组结点彼此之间肯定是同构的。然后在所有子集合上计算得到的 LCA 只要与当前 GLCA 进行比较就可以得到最终的结果。

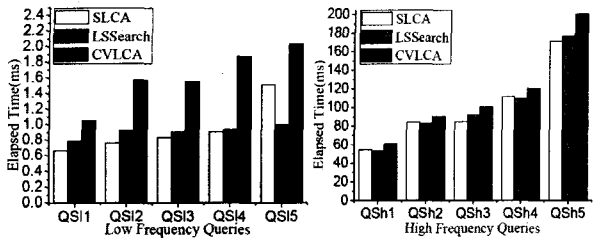


图 5　在 SIGMOD RECORD 数据集上的检索时间比较

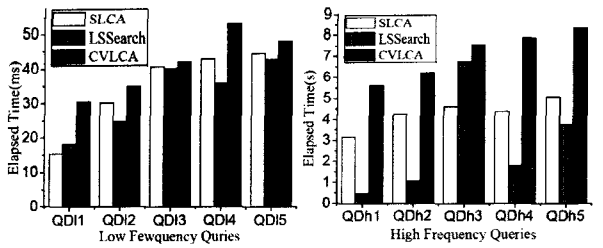


图 6　在 DBLP 数据集上的检索时间比较

在关键字原始倒排表比较小时，LSSearch 算法和 SLCA 中的 IL 算法相差很小，如 QSl1，QSl2 等。因为 LSSearch 算法此时关键字倒排表的规模减少不是很明显，同时在第一步计算 GLCA 时，LSSearch 算法还需要得到每一 LCA 子树是如何构建的。但是，当关键字的倒排表较大时，LSSearch 算法体现出了较佳的性能，如在图 6 中的 QDh1，QDh2 和 QDh4 等。关键字倒排表规模最小是 11 万个结点，在这种情况下，LSS 可以减少查询关键字倒排表的规模，从而减少检索时间，提高检索性能。

结束语　有别于基于 XML 查询语言的 XML 数据处理，基于关键字的 XML 信息检索的处理方式以其便于普通用户使用的特点，在最近几年受到广泛关注。先后出现了大量的文献，提出了各种方法解决 XML 关键字检索的种种问题。如文献[6]的 ELCA、文献[7]的 MLCA、文献[8]的 SLCA、文献[9]的 CVLCA。这些方法都采用传统的倒排索引来提高检索性能。然而在一个 XML 文档中，某一个关键字出现的频率较高时，同时输入的关键字集合中恰好包含此关键字，ELCA 方法的性能较差，在检索结果时就会消耗较多的时间。当然，如果输入的关键字集合中至少有一个关键字的倒排表很小时，SLCA 中的 IL 算法的性能表现优越。当输入的关键字集合中各个关键字的倒排表规模都较大时，目前所有方法的性能都并不理想。与之不同，本文提出了将结构索引和关键字检索相结合的方法来缩减倒排表的规模大小，在结构索引上创建倒排索引。实验表明，大部分关键字的倒排表的大小较之前有明显的缩减，然后在其上进行关键字检索。

后续研究将着眼于在结构索引结果基础上进行相关性判断并排序。目前大量文献设计了多种方法计算返回结果和用户输入的关键字之间的相关性程度。如 SLCA 返回最小的、最紧致的 XML 片段，CVLCA 通过同构和异构判断来提高查询的准确度。那么，在结构索引的前提下，设计相关性判断方法将是我们后续研究的方向。由于现实世界的复杂性，概率 XML 数据管理[11]也是我们关注的方向。

参 考 文 献

[1] eXtensible Markup Language(XML)[EB/OL]. http://www.w3.org/xml

[2] Chen Qun, Lim A, Ong K W. D(k)-index: an adaptive structural summary for graph-structured data[C]// Proceedings of the 2003 ACM SIGMOD. New York: ACM, 2003: 134-144

[3] Roy G, Jennifer W. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases[C]//Proceedings of the 23rd International Conference on Very Large Data Bases. CA: Morgan Kaufmann Publishers Inc., 1997: 436-445

[4] Raghav K, Pradeep S, Philip B, et al. Exploiting Local Similarity for Indexing Paths in Graph-Structured Data [C]// Proceedings of the 18th International Conference on Data Engineering. Washington: IEEE Computer Society, 2002: 129-131

[5] Tova M, Dan S. Index Structures for Path Expressions [C]// Proceedings of the 7th International Conference on Database Theory. London: Springer-Verlag, 1999: 277-295

[6] Guo Lin, Shao Feng, Chavdar B, et al. XRank: Ranked keyword search over xml documents[C]// Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2003: 16-27

[7] Li Yunyao, Yu Cong, Jagadish H V. Schema-free XQuery[C]// Proceedings of the Thirtieth international conference on Very large data bases. VLDB Endowment, 2004: 72-83

[8] Xu Yu, Yannis P. Efficient keyword search for smallest LCAs in XML databases[C]//Proceedings of the 2005 ACM SIGMOD. New York: ACM, 2005: 527-538

[9] Li Guoliang, Feng Jianhua, Wang Jianyong, et al. Effective keyword search for valuable lcas over xml documents[C]// Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management. New York: ACM, 2007: 31-40

[10] http://www.cs.washington.edu/research/xmldatasets

[11] 王建卫，郝忠孝. 概率 XML 数据管理技术研究进展[J]. 计算机科学，2009，36(11)：14-17