计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
48(Suppl.): 343-349, 2011

基于拓扑序列的 DAG 子图包含查询算法研究

奚业雷　吕建华　张柏礼
(东南大学计算机科学与工程学院　南京　211189)
(xiyelei0112@163.com)

Topological Sequence Based DAG Subgraph Containment Query

Xi Yelei, Lü Jianhua, and Zhang Baili
(*School of Computer Science & Engineering, Southeast University, Nanjing*　211189)

Abstract　For the powerful modeling ability of graph model, graph data is ubiquitous in various data applications, such as proteins, chemical compound, XML documents, and wireless sensor networks. Efficiently subgraph containment query processing in large-scale graph database is one of the key challenges to the database community. Given a graph database G, and a query graph q, the graph containment query is to retrieve all graphs in G which contain q as subgraph. Since subgraph isomorphism is an NP-complete problem, the graph feature based index structure is widely used in many algorithms to improve the performance of subgraph containment query processing. However, the time and space complexities to construct the indexes are relatively high, and therefore it is very difficult to maintain the indexes dynamically. This paper proposes a topological sequence based subgraph containment query processing algorithm for directed acyclic graph data. In this algorithm, directed acyclic graphs are transformed to topological sequences based on the partial relationships between graph nodes, then a candidate set is filtered out by sequence matching, and a subgraph isomorphism validating procedure is conducted to obtain final results. The experimental results show that the graph topological sequences computation and database maintaining cost less time and space overheads, and the performance of online subgraph containment query processing is better than the existed methods.

Key words　subgraph containment query; DAG; topological sequence; subgraph isomorphism; graph indexing

摘　要　图模型具有强大的表达能力,被广泛用于各种应用领域的数据建模.如何在大规模图数据库中进行高效子图包含查询是当前的研究难点之一.由于子图同构是一个 NP 完全问题,在现有的子图包含查询算法中,基于图特征的索引技术被广泛用来提高查询处理性能,但是这些索引结构的维护代价较高.针对有向无环图提出了一种基于拓扑序列的子图包含查询算法,首先根据图中节点的偏序关系将有向图分层拓扑为一个序列,然后利用序列间的匹配关系过滤出候选结果集,最后通过子图同构检测验证得到最终结果集.相关性能测试表明,该算法无需构造复杂的索引结构,便于图数据库的动态维护,在有向无环图在线查询性能上表现出色.

关键词　子图包含查询;有向无环图;拓扑序列;子图同构;图索引

中图法分类号　TP311.13

收稿日期:2011-07-15
基金项目:国家自然科学基金项目(61073059);江苏省自然科学基金项目(BK2010409)

图被广泛用于复杂结构化数据建模. 近年来,科学与工程领域积累了大量可用图来建模的数据,如化合物的分子结构,蛋白质交互网络以及社会网络等. 随着时间的推移,所累积的图数据库越来越大,大数据量的图数据查询问题逐渐受到数据管理界研究者的关注. 作为实际应用中重要的问题之一,子图包含查询也成为相关研究的热点.

研究者们已经提出了多种子图包含查询处理方法. 这些方法广泛采用了"过滤-验证"机制,首先根据离线建立的索引,利用过滤规则获得候选结果集,然后通过子图同构检测验证每个候选结果. 由于同构验证的复杂度较高,研究者提出了不同的基于图特征的索引技术来提高过滤能力.

GraphGrep[1]是数据库领域第 1 篇研究图数据中子图查询问题的论文. 作者提出了以图中的路径作为特征构建索引的方法. 在离线阶段 GraphGrep 将图中所有长度小于 lp 的路径都列举出来,作为索引项. 如果数据图 G 包含查询图 q,则所有在 q 中出现的路径,在 G 中也应该出现. 根据这样的必要条件,通过倒排索引可以很快地确定候选集合.

gIndex[2-3]的作者认为路径不能很好地描述图数据库中的结构信息,提出选择一部分频繁子图作为索引项来提高过滤能力. gIndex 引入了 2 种技术(size-increasing support constraint 和 discriminative fragments)提出了频繁子图过滤能力的衡量标准. gIndex 总是选择那些具有较高过滤能力的频繁子图作为索引项. FG-Index[4]也使用频繁子图作为索引项. 不同于 gIndex,FG-Index 并不根据过滤能力来选择一部分频繁子图作为索引项,而是选择所有的频繁子图作为索引项. 同时,为了避免由于索引项增多带来的问题,FG-Index 采用了 2 层的索引策略. 内层的索引被维护在内层中,外层的索引维护在硬盘上. 这样的设计策略的好处在于减少 I/O 开销.

在同样的框架下,TreePi[5]用频繁子树作为索引项. 频繁树比频繁子图更易于操作,而且比路径作为索引项提供了更多的结构信息. Tree+Δ[6]除了利用频繁子树外,还结合一些小的子图结构作为索引项.

除了上述基于特征的方法外,gString[7]考虑了分子结构数据库的特点,提出了一种 String 的编码方式来表示图结构. 然而这种方法只是针对于分子结构数据库,无法推广到其他类型的图数据库中. Closure-Tree[8]在图数据库中引入了一种偏序的概念. 类似于空间数据库中的 R-Tree 索引,Closure-Tree 索引是一种平衡树的索引方式. 如果 Closur-Tree 一个中间结点所表示的 Closure-Tree 不能包含查询图 q,则这个结点分支下所包含的所有数据图均不可能包含 q.

以上各种算法主要研究如何利用图特征建立索引结构来提高过滤能力,而较少考虑索引构造过程的代价. 然而,索引特征的提取和索引的构造都是耗时的操作. 因此,在这些算法中,特征提取与索引构造都是离线进行的. 对于更新频繁的图数据库,这些算法的索引动态维护代价较高,不适用于在线查询.

本文针对有向无环图 DAG,提出了一种基于拓扑序列的 DAG 子图包含查询算法,不需要进行复杂的特征提取,降低了数据库动态维护代价,有效提高了在线查询性能. 本文的主要贡献如下:

1) 定义了 DAG 中的严格偏序关系,利用偏序关系提出了 DAG 的分层拓扑算法,以较低的时间复杂度得到包含 DAG 结构信息的拓扑序列.

2) 提出了序列匹配算法. 相对于图的匹配,序列的匹配较为简单,通过序列间的匹配过滤出候选结果集,从而减少同构测试的次数.

3) 采用通用图数据测试集,进行了相关性能测试与分析,结果表明本文算法在特征提取、索引构造以及在线查询等方面都具有较高的性能.

1 基于拓扑序列的 DAG 子图包含查询算法

算法要解决的问题是 DAG 子图包含查询,DAG 中的边可以看作一种偏序关系,利用这种偏序关系可将 DAG 拓扑成一个序列,利用序列匹配算法过滤出候选结果集,最后通过同构测试验证候选结果. 本节介绍分层拓扑算法和序列匹配算法.

1.1 相关定义和算法描述

定义 1. 有向图. 一个有向图定义为一个四元组 $G=(V,E,\Sigma,l)$,其中,V 代表顶点的集合,E 代表边的集合(有向图中的边是由 2 个顶点组成的有序对,如$\langle v_i,v_j\rangle$表示一条有向边,其中 v_i 是边的始点,v_j 是边的终点),Σ 代表顶点标签的集合,l 是标签函数,用来对顶点分配标签,即 $l:V\rightarrow\Sigma$.

在下文中,对于图 G,用 V_G,E_G,Σ_G 和 l_G 分别表示 G 的顶点集、边集、标签集和标签函数. $N_G(v)$ 表示在 G 中由 v 引出的边指向的顶点的集合. 图 G 的大小定义为 $size(G)=|E_G|$.

定义 2. DAG 上的严格偏序. 对于顶点集 V,对 E 中的每一条边$\langle a,b\rangle$,定义偏序关系 $a<b$,则"$<$"

是顶点集 V 上的严格偏序,满足反自反,非对称和传递性.

定义 3. 子图包含查询. 给定图数据库 $D=\{G_1,G_2,\cdots,G_n\}$(n 为有穷自然数)和查询图 q,找出包含查询图 q 的所有 $G_i(G_i\in D)$.

定义 4. DAG 的拓扑序列. 对于有向无环图 $G=(V,E,\Sigma,l)$,设 $V=C_1\cup C_2\cup\cdots\cup C_n$,且 $C_i\cap C_j=\varnothing(i\neq j)$,序列:$C_1/C_2/\cdots/C_n$ 称作有向无环图 G 的拓扑序列. C_i 为删除所有 $C_k(k<i)$ 包含的顶点及其引出的边之后,入度为 0 的点的集合.

表 1 本文定义的一些记号和术语

符号	说明
q	query graph
G_i	$G_i\in D$
$q.str$	the topological sequence of q
$G_i.str$	the topological sequence of G_i
$G_i.str[s]^+$	the topological sequence from C_s to last of $G_i.str$

算法分成 3 个阶段,主要框架如下.

索引构造阶段:将图数据库中的有向图使用分层拓扑算法拓扑成序列,作为索引项;

过滤阶段:首先将给定的查询图使用分层拓扑算法拓扑成序列,然后匹配数据库中每个 DAG 的拓扑序列,如果匹配则加入候选结果集.

同构验证阶段:将候选结果集中的图与查询图作同构测试,同构算法采用 Ullman 算法[10],得到最终的结果集.

不同于现有的子图查询方法,在基于拓扑序列的 DAG 子图包含查询算法中,图数据库中的每个图只需执行一次分层拓扑算法. 当图数据库更新时,只需对新加入的 DAG 执行一次分层拓扑算法且对以前的索引没有影响,因此不需要耗费大量的时间更新索引,适用于在线操作.

1.2 分层拓扑算法

传统的拓扑排序算法可将图 1 所示有向图拓扑成序列:$abcbade$,若用这样的序列来代表有向图,存在如下的缺陷:1)序列具有不唯一性. 因为传统的拓扑排序算法每次选择一个入度为 0 的顶点输出,则图 1 所示有向图可能对应多个拓扑序列. 2)丢失了顶点间的并行信息. 图 1 有 2 个入度为 0 的顶点 a 和 b,a,b 之间是并行关系,而上面的拓扑序列却无法体现这种并行关系.

为了解决上述的 2 个缺陷,本文提出了分层拓扑算法,可将图 1 所示有向图拓扑成序列:$a,b/b,c/a/d,e$. 从而使每个 DAG 对应一个拓扑序列,而且序列包含了顶点间的并行信息.

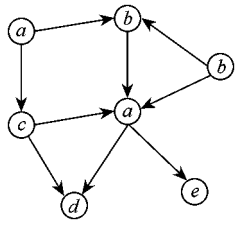

图 1 有向图

算法 1. 分层拓扑算法.

输入:有向无环图 G 的邻接表形式;

输出:G 的拓扑序列.

① While V_G is not empty do
② For each $v\in V_G$ do
/* $v.flag$ 初始为 0 */
③ If $indegree(v)==0$ && $v.flag==0$ do
④ add v to C_i;
⑤ $v.flag=-1$;
⑥ i++; /* 每次循环产生一个 C_i */
⑦ For each $v\in V_G$ do
⑧ If $indegree(v)==0$ && $v.flag==-1$ do
⑨ Delete v;
⑩ $v.flag=1$;
⑪ For $v'\in N_G(v)$ do
⑫ $--Indegree(v')$;

DAG 可以成功地拓扑成一个序列,将算法得到的拓扑序列记作 $C_1/C_2/\cdots/C_n$,C_i 表示第 i 层.

对算法和拓扑序列进行分析不难得出如下的性质:

性质 1. 同一层的任意 2 点间不存在偏序关系.

性质 2. 在 $C_i(i=2,\cdots,n)$ 中每一个顶点存在前驱顶点,且必有一个前驱顶点在 C_{i-1} 中.

证明. 假设存在顶点 $v\in C_i$,v 在 C_{i-1} 中不存在前驱顶点. 因为 v 在 C_i 中入度为 0 且在 C_{i-1} 中不存在前驱顶点,所以在输出 C_{i-1} 时,v 入度为 0,v 在 C_{i-1} 中,与题设矛盾. 证毕.

性质 3. 顶点 a 与顶点 b 之间存在偏序关系 $a<b$,则对应的序列中 b 必出现在 a 之后.

证明. 由 $a<b$ 知存在边$\langle a,b\rangle$,则当 a 的入度

为 0 时,b 的入度至少为 1. 算法 1 每次输出入度为 0 的顶点,则 b 必出现在 a 之后. 证毕.

接下来分析拓扑序列所包含的 DAG 的信息. 以图 1 中的 DAG 为例,序列 $a,b\ /b,c\ /a\ /d,e$ 包含了所有顶点信息. 对于边的信息,如图 2 所示,序列仅仅隐含了部分边的信息,边的信息并不明确.

为了提高拓扑序列的可区分性,代表更多的有向图信息. 在算法 1 输出顶点的同时,将该顶点所引出的边信息附加到顶点中一同输出. 图 1 的拓扑序列就变为:$a[bc],b[ab]\ /b[a],c[ad]\ /a[de]\ /d[],e[]$,方括号中的内容为由前面顶点引出的边的集合.

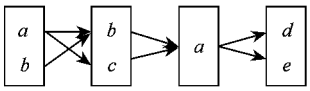

图 2 边信息

1.3 序列匹配算法

问题的分析从最简单的 DAG——无重复标签的 DAG——开始,引出算法,然后分析标签重复的情况,改进算法,最终使算法扩展到所有的 DAG.

1.3.1 无重复标签 DAG

图 3 为两个无重复标签的 DAG,G_i 为图数据库 D 中的一个图,q 为查询图. 利用分层拓扑算法得到拓扑序列:

$G_i.str$: $a[be],c[bd]\ /b[d],e[df]\ /d[fg]\ /f[],g[]$;

$q.str$: $c[bd],e[df]\ /b[d]\ /d[f]\ /f[]$;

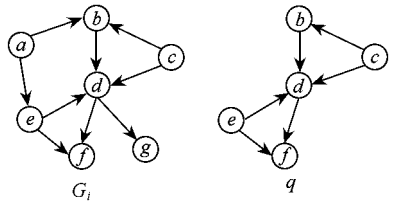

图 3 无重复标签 DAG

定义 5. 顶点的匹配. 顶点 $v\in G_i$,顶点 $v'\in q$,若 v 与 v' 匹配,则 $l_{G_i}(v)=l_q(v')$,$N_{G_i}(v)$ 包含 $N_q(v')$.

序列匹配的思想是 $q.str$ 按层匹配,$q.str$ 每层的顶点在 $G_i.str[s]^+$(s 初始为 1)中找匹配点,选取匹配点位置最靠前的顶点所在层数赋给 s,作为下次匹配的起点. 如果 $q.str$ 的所有层都能在 $G_i.str$ 中成功匹配,则称"$G_i.str$ 匹配 $q.str$",记作 $G_i.str\supseteq q.str$. 若匹配,则 q 可能是 G_i 的子图,G_i 加入候选结果集,否则,$G_i.str$ 不匹配 $q.str$,q 不可能是 G_i 的子图. 具体算法如下:

算法 2. 序列匹配算法.

Input: $G_i.str$ 和 $q.str$;

Output: 匹配输出 true,否则,false.

① for each C_i in $q.str$ do
② for each v in C_i do
③ bool $isFind$ = false;
④ for each C' in $G_i.str[s]^+$ do
⑤ if(C'中存在与 v 匹配的顶点)
⑥ then $isFind$ = true; break;
⑦ if(!$isFind$) then
⑧ return false; /* 不匹配 */
⑨ 令 $s=\min\{C_i$ 中顶点在 $G_i.str$ 中匹配顶点的层数$\}$;
⑩ return true; /* 匹配 */

以图 3 为例,序列的匹配过程如图 4 所示. $q.str$ 按层匹配,在①中取 $q.str$ 的第 1 层 $c[bd],e[df]\ /$ 与 $G_i.str[1]^+$ 匹配,$G_i.str[1]^+$ 中存在对应的匹配点(图 4 中用下划线标出),将 s 置为最靠前的匹配点($c[bd]$)所在的层数 $s=1$,作为下次匹配的起点;在②中取 $q.str$ 的第 2 层 $b[d]\ /$与 $G_i.str[1]^+$ 匹配,存在匹配点 $b[d]$,s 设为匹配点所在层数,即 $s=2$;在③中 $q.str$ 的第 3 层 $d[f]/$与 $G_i.str[2]^+$ 匹配,存在匹配点 $d[fg]$,置 $s=3$;在④中 $q.str$ 的最后一层 $f[]$在 $G_i.str[3]^+$ 也得到了匹配. 因此,$G_i.str$ 匹配 $q.str$,G_i 可能包含查询图 q,G_i 加入候选结果集.

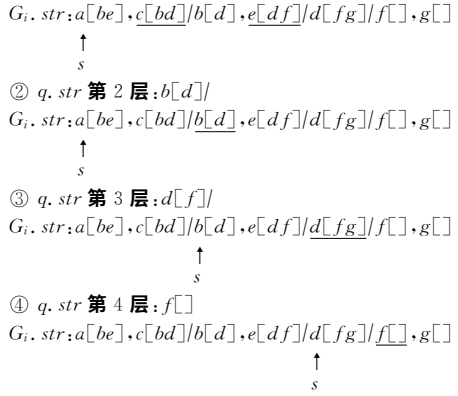

图 4 序列匹配过程

由上述的算法可以推导出如下命题:

命题 1. 若 G_i 和 q 为标签不重复的有向无环图且 G_i 包含 q,则 $G_i.str$ 必匹配 $q.str$.

证明. 假设 G_i 包含查询图 q,$G_i.str$ 不匹配 $q.str$,则在 $q.str$ 的某一层 j 存在一个顶点 x,x 在 $G_i.str[s]^+$ 不存在匹配点.

1) 当 $j=1$ 时,$s=1$,因为 G_i 包含 q,$G_i.str[1]^+$

必然包含 x 匹配点 x',推出矛盾.

2) 当 $j>1$ 时,由性质 2 知,x 在 $q.str$ 的 $j-1$ 层存在前驱顶点 y,即存在一条有向边$\langle y,x\rangle$. $q.str$ 的 $j-1$ 层中顶点 y 在 $G_i.str$ 中有对应的匹配点 y',G_i 包含 q 且标签不重复,所以也包含有向边$\langle y',x\rangle$,即 $y'<x$,由性质 3 知,在 $G_i.str$ 中存在一个 x 出现在 y'之后. 当匹配 $q.str$ 的第 j 层时,起始层 $s\leqslant y'$所在的层,所以 x 出现在 $G_i.str$ 的 s 层之后,即 $G_i.str[s]^+$ 存在与 x 匹配的点,推出矛盾. 证毕.

对命题 1 的证明也是对算法不丢解的证明.

1.3.2 标签可重复 DAG

标签不重复时,$q.str$ 中的顶点在 $G_i.str$ 中可以找到明确的匹配顶点. 而当标签可重复时,$q.str$ 中的顶点在 $G_i.str$ 中可能存在多个匹配顶点.

如图 5 中 G_i 和 q 的拓扑序列分别为 $G_i.str$: $a\,|\,c,g\,|b_2,d\,|e,h\,|b_1|f\,|k$ 和 $q.str$:$b_1,g\,|b_2,f\,|h\,|k$(其中 b_1,b_2 的标签都为 b,此处为了区分标记为 b_1,b_2). 显然 q 是 G_i 的子图,但是 $G_i.str$ 和 $q.str$ 中 b_1,b_2 的位置发生了互换. $q.str$ 中的 b_1 就会匹配到 $G_i.str$中的 b_2,不同于标签不重复时的唯一匹配.

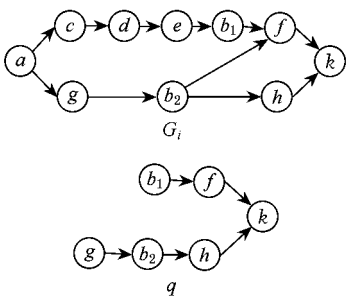

图 5 标签可重复 DAG

对算法 2 进行分析不难得出如下的命题:

命题 2. $q.str$ 的第 i 层 C_i 成功匹配后设置的 s 值小于等于 C_i 中任意顶点在 $G_i.str$ 中对应顶点所在层数.

此处用对应顶点来区别匹配顶点. 如图 5 中 $q.str$中的 b_1 在 $G_i.str$ 中的对应顶点为 b_1,匹配点为 b_2,b_1.

证明. 对于 $q.str$ 第 i 层 C_i 中的任一顶点 v,算法 2 在 $G_i.str$ 中查找 v 的匹配点时,选取第 1 个匹配顶点作为 v 的匹配点 v',则 v 的对应点所在层数必大于等于 v'所在层数. 而 $s=\min\{C_i$ 中顶点在 $G_i.str$ 中匹配顶点的层数$\}$,则 s 小于等于 C_i 中任意顶点在 $G_i.str$ 中对应顶点所在层数. 证毕.

将算法 2 推广到标签可重复 DAG,则需证明下面的命题,以保证不丢解.

命题 3. 若 G_i 和 q 为 DAG 且 G_i 包含 q,则 $G_i.str$必匹配 $q.str$.

证明. 假设 G_i 包含查询图 q,$G_i.str$ 不匹配 $q.str$. 则在 $q.str$ 的某一层 j 存在一个顶点 x,x 在 $G_i.str[s]^+$ 不存在匹配点.

1) 当 $j=1$ 时,$s=1$,因为 G_i 包含 q,$G_i.str[1]^+$ 必然包含 x 匹配点 x',推出矛盾.

2) 当 $j>1$ 时,由性质 2 知,x 在 $q.str$ 的 $j-1$ 层存在前驱顶点 y,即存在一条有向边$\langle y,x\rangle$. $q.str$ 的 $j-1$ 层中的顶点 y 在 $G_i.str$ 中的对应点为 y',G_i 包含 q,所以也包含有向边$\langle y',x\rangle$,即 $y'<x$,由性质 3 知,在 $G_i.str$ 中存在一个 x 出现在 y'之后. 由命题 3 知,当匹配 $q.str$ 的第 j 层时,起始层 $s\leqslant y'$所在的层,所以 x 出现在 $G_i.str$ 的 s 层之后,即 $G_i.str[s]^+$ 存在与 x 匹配的点,推出矛盾. 证毕.

2 实 验

实验首先验证所提出算法的正确性,然后与 iGraph[11] 中实现的子图包含算法进行比较,分析算法的性能.

实验的硬件环境为:Intel® Xeon® CPU E5506 @2.13 GHz 2.13 GHz,12 GB 内存,操作系统采用 64 位 Windows 7.

2.1 实验数据生成

实验所用的图数据由 GraphGen[11] 生成. 它通过输入参数控制产生的图数据,如图的个数、图的平均大小、顶点/边标签的数目、图的平均密度. 图 $g=(V,E)$ 的密度 d 定义为 $\frac{\#\text{ of edges in } g}{\#\text{ of edges in a complete graph}}$(分母的大小为 $\frac{|V|\times(|V|-1)}{2}$)[10]. 例如数据集 synthetic.10K.E30.D5.L50 表示图的个数 10 000,图的平均大小 30、图的平均密度 0.5、顶点/边标签 50.

对于查询图数据,实验根据图的大小分成了 Q_4,Q_8,Q_{12},Q_{16},Q_{24} 5 类. Q_n 包含了 1 000 个大小为 n 的图.

2.2 算法比较

为了分析在不同图数据大小情况下算法的性能,实验分别以 synthetic.10K.E30.D3.L50,synthetic.10K.E30.D5.L50 作为图数据库.

表 2 列出了在密度 $d=0.3,0.5$ 时,4 种算法的索引构造时间. gIndex 总是选择那些具有较高过滤

能力的频繁子图作为索引项,导致索引的构建耗费了大量的时间. FG-Index 挖掘所有的特征,使用 2 层索引,耗时较多. Tree+Δ 挖掘频繁树和少量的频繁图作为索引项,表现较好. TopoS 对数据库中的每个图仅需使用分层拓扑算法,4 种算法中性能最优.

表 2 索引构造时间比较 s

算法	when $d=0.3$, construction time	when $d=0.5$, construction time
TopoS	1.563	1.476
gIndex	56.984	436.234
FG-Index	7.646	12.604
Tree+Δ	6.442	5.948

图 6 显示了在密度 $d=0.3,0.5$ 时,随查询图大小的改变,4 种算法所得候选结果集平均大小的变化情况. TopoS 所得候选集小于 gIndex 和 FG-Index,具有较强的过滤能力.

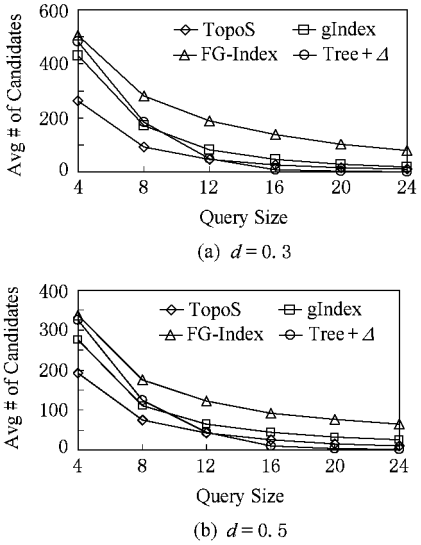

图 6 候选结果集平均大小比较

图 7 显示了在密度 $d=0.3,0.5$ 时,4 种算法过滤出候选结果集的平均运行时间比较情况. FG-Index 所用时间最少,但是从图 6 中可以看出 FG-Index 的过滤能力最差. TopoS 和 gIndex 性能相差不大. Tree+Δ 随图大小的增加运行时间呈指数级增加,但过滤能力最强.

对于在线查询,当数据库中存在更新时,索引必须重新构造,而此时的在线查询时间为索引构造时间与索引构造完后的查询时间之和(假设算法使用相同的同构算法,此处忽略验证阶段所用时间). 图 8 显示了在密度 $d=0.3,0.5$ 时,4 种算法在线平均运行时间的比较情况. gIndex 因索引的构造耗费了大量的时间,在线查询效果与其他 3 种算法相差很大,不适于在线查询. TopoS 效果优于 FG-Index 与 Tree+Δ. 当查询图较小时, Tree+Δ 优于 FG-Index,但是 Tree+Δ 查询时间随查询图增大呈指数级增长,而 TopoS 性能比较平稳.

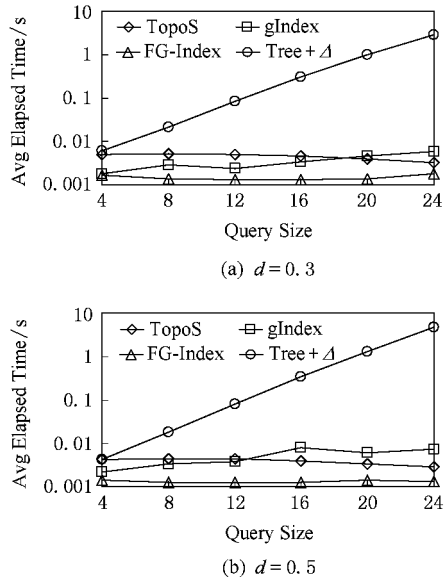

图 7 平均运行时间比较

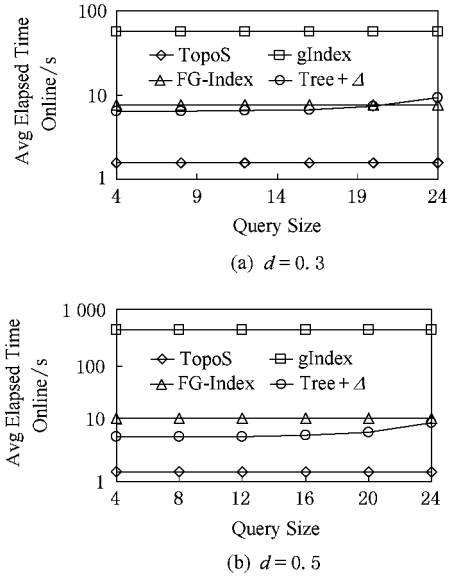

图 8 在线平均运行时间比较

此外,当数据库存在更新时, gIndex, FG-Index, Tree+Δ 需要对整个数据库重新构造索引,而 TopoS 仅需对更新的数据构造索引,因此效果还会优于图 8 中所示.

综上所述, TopoS 无需构造复杂的索引,便于数据库的动态维护,在线查询表现出色.

3 总结和展望

本文提出了 DAG 的分层拓扑算法和序列匹配算法. 算法将 DAG 拓扑成序列,通过序列间的匹配过滤出候选结果集,然后通过同构测试验证候选结果,从而得到最终的结果集. 相对于以特征为索引的子图包含查询算法,该算法无需构建复杂索引结构,便于数据库的动态维护,可用于在线查询.

算法目前只针对于 DAG,对于有向有环图还没有提出好的处理方法. 一种简单的思想是将环收缩成一个点,从而将有向有环图变为有向无环图,解决了这个问题便可将算法扩展到所有的有向图.

参 考 文 献

[1] Dennis Shasha, Wang Jason T L, Giugno Rosalba. Algorithmics and applications of tree and graph searching // Proc of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems(PODS). New York: ACM, 2002: 39-52

[2] Yan Xifeng, Yu Philip S, Han Jiawei. Graph indexing: A frequent structure based approach //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2004: 335-346

[3] Yan Xifeng, Yu Philip S, Han Jiawei. Graph indexing based on discriminative frequent structure analysis. ACM Trans on Database Systems(TODS), 2005, 30(4): 960-993

[4] Cheng James, Ke Yiping, Ng Wilfred, et al. FG-Index: Towards verification free query processing on graph databases //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2007: 857-872

[5] Zhang Shijie, Hu Meng, Yang Jiong. Treepi: A novel graph indexing method //Proc of the 23rd Int Conf on Data Engineering (ICDE). Los Alamitos: IEEE Computer Society, 2007: 966-975

[6] Zhao Peixiang, Yu Jeffrey Xu, Yu Philip S. Graph indexing: Tree + delta >= graph //Proc of the 33rd Int Conf on Very Large Data Bases (VLDB). New Yodk: ACM, 2007: 938-949

[7] Jiang Haoliang, Wang Haixun, Yu Philip S, et al. Gstring: A novel approach for efficient search in graph databases // Proc of the 23nd Int Conf on Data Engineering(ICDE). Los Alamitos: IEEE Computer Society, 2007: 566-575

[8] He Huahai, Singh Ambuj K. Closure-tree: An index structure for graph queries //Proc of the 22nd Int Conf on Data Engineering (ICDE). Los Alamitos: IEEE Computer Society, 2006: 38-47

[9] 邹磊. 图数据库中的子图查询算法研究. 武汉: 华中科技大学, 2009

[10] Ullmann J R. An algorithm for subgraph isomorphism. Journal of ACM, 1976, 23(1): 31-42

[11] Han Wook Shin, Lee Jinsoo, Pham Minh Duc, et al. iGraph: A framework for comparisons of disk-based graph indexing techniques //Proc of the 36rd Int Conf on Very Large Data Bases (VLDB). New York: ACM, 2010: 449-559

奚业雷 男,1986 年生,硕士研究生,主要研究方向为不确定图数据管理、数据挖掘.

吕建华 男,1977 年生,副教授,硕士生导师,主要研究方向为传感器网络、不确定图数据管理技术等.

张柏礼 男,1970 年生,副教授,主要研究方向为不确定图数据管理、传感器网络、数据仓库等.