基于频繁闭图的图包含查询算法

李先通,安 实
(哈尔滨工业大学交通科学与工程学院,黑龙江哈尔滨 150000)

摘 要: 交通网络可利用图数据进行描述与分析,常用的方法包括挖掘、查询、分类等.提高大规模图集上查询算法效率的问题是当前图数据分析领域中一个重要的研究方向.给定图集,图包含查询返回图集中所有查询图的子图.本文提出一种基于频繁闭图的包含查询算法.算法首先通过选择比消除频繁闭图之间的冗余,然后将具有强选择性的频繁闭图通过树的结构组织起来建立索引,并在此索引基础上实现图包含查询.在文章的最后,给出了理论与实验的分析结果.结果表明,该算法不但能高效的进行索引筛选,而且能显著的减小候选集尺寸,进而大大的降低了查询图与索引模式之间以及与候选集之间的子图同构测试次数,提高了查询效率.

关键词: 交通网络; 图数据库; 图索引; 包含查询; 频繁子图
中图分类号: TP311.131 **文献标识码:** A **文章编号:** 0372-2112 (2010) 12-2937-07

A Closed Frequent Subgraph Based Containment Query Algorithm

LI Xian-tong, AN Shi
(*School of Transportation Science and Engineering, Harbin Institute of Technology, Harbin, Heilongjiang* 150000, *China*)

Abstract: Transportation network can be well described and analyzed through graph data. The analyzing methods include mine, query, and classification, etc. The improvement on the efficiency of scalable algorithms on large graph dataset is important in graph analyzing. Given a graph dataset and query graph, graph containment query retrieves graphs from the dataset which are subgraphs of query graph. In this paper, a frequent closed subgraph (CFG) based graph containment query algorithm is proposed. The algorithm selects discriminative-redundancy-aware CFG to build a tree structure index, which satisfies such query. Theoretical analysis and experimental evaluation are presented at the end. The results show that this algorithm is not only filtering out correlated indexed features efficiently, but also reducing subgraph isomorphism tests between query graph and indexed features effectively.

Key words: transportation network; graph database; graph index; containment query; frequent subgraph

1 引言

图是信息处理技术中一种通用数据结构,能利用节点与边有效描述很多领域内的结构化数据.在交通网络中,可用节点描述公路交叉口,用边描述公路段.

采用图数据描述公路交通网络,既方便交通历史数据查询与管理,也使通过查询实时分析交通数据更为便利.给定图集,图查询可分为两类:子图查询与包含查询.子图查询返回查询图在图集中的超图,例如:在交通图中,分析当前拥塞情况可能造成的严重后果,即可通过子图查询找到历史数据中覆盖区域更大的子图.与此相反,包含查询则返回图集中被查询图包含的图,例如:当大规模拥塞发生时,分析历史数据中最接近的某公路交叉口指挥方案,可通过包含查询分析该交叉口子图数据,其查询结果有助于研究人员确定交通疏导方案.

1.1 相关工作

目前,子图查询已提出若干算法.无论是精确查询还是近似查询,均采用"过滤-验证"机制,其过滤方式为包含逻辑,即:若查询图 q 是某个图 g 的子图,则 q 所包含的所有子模式也必被 g 包含.反之,即使 q 中仅有一个模式不被 g 包含,g 也不可能是 q 的超图,将从候选集中删除.在文献[1~4]中已提出的算法,以及在文献[1]算法上建立的近似查询算法[5,6]均采用图挖掘算法[7]生成子图模式(如路径,树或子图等)构建索引.而文献[8~12]提出的算法采用与前者不同的构建方式,如图闭包,有向无环图,图集特殊构件等.另外,在专用领域内,存在效率更高的算法,如文献[13,14]提出的应用于 XML 中的算法等.

然而,子图查询算法无法用于包含查询.与前者不同,包含查询利用"排除逻辑"过滤.排除逻辑假定:一个查询图 q 是某目标图(图集中的图)g 的超图,那么 g 中包含的所有图模式也必被 q 包含.根据排除逻辑,如果一个图模式 f 不是 q 的子图,则该模式的任何超图均可被排除.

收稿日期:2008-11-19;修回日期:2010-05-12

目前,已经发表的包含查询算法是 Chen 等在文献[15]中提出的 cIndex 算法,采用"过滤-验证"机制.算法基于强选择性无冗余图模式(discriminative redundant-aware features).cIndex 主要效率损耗包括选择构建索引的图模式,鉴别索引模式是否被 q 包含(该过程需要进行查询图与索引模式之间的子图同构测试)和验证阶段查询图与候选图之间的子图同构测试.作者共提出三个算法:cIndex-Basic,cIndex-TopDown,cIndex-BottomUp.这些算法的区别在索引模式筛选.cIndex-Basic 采用逐个比较的方法,显然效率低下.cIndex-BottomUp 采用层次结构.最底层是目标图,在此基础上,循环调用 cIndex-Basic 算法形成更小的图模式,直到层数达到用户给定值为止.相邻两层节点是子图与超图的关系,高层模式能覆盖尽量多的目标图,达到减小模式筛选过程中子图同构次数.然而,这种分层结构很可能在查询过程中引入额外的子图同构,导致效率降低.cIndex-TopDown 同样是层结构,它采用类似判断树的方式组织索引模式.假设模式 f_i 位于第 i 层,若 f_i 是 q 的子图,算法检查第 $(i+1)$ 层的一部分模式,若 f_i 不是 q 的子图,算法则检查第 $(i+1)$ 层的另一部分模式.cIndex-TopDown 减小了模式筛选中子图同构次数,但却加入了对索引模式分类的附加计算.

1.2 本文的贡献

本文提出 CFG-Index (Closed Frequent SubGraph Based Index)算法解决图包含查询问题.本文在给出频繁闭图集的定义的基础上,提出一种新的索引结构,提高图包含查询的效率.理论分析和实验结果证明了算法的高效性和可扩展性.本文的主要贡献在于:(1)设计了一个基于频繁闭图的树状索引结构和此索引之上的查询算法,该算法能高效处理图包含查询;(2)通过理论与实验两部分分析了该算法的效率.

2 问题定义

2.1 标号图与同构

定义 1(标号图) 标号图是一个四元组 (V,E,l,Σ),其中 V 代表节点的集合,$E\subseteq V\times V$ 代表边的集合,l 表示标号函数,用于实现标号集合 Σ 向节点或边的映射 $l: V\cup E\rightarrow\Sigma$.

标号图可应用于很多领域,在不同的应用中,标号可用于描述不同的属性,边表示属性之间的关联.在本文剩余部分,$V(g)$ 和 $E(g)$ 分别表示节点和边的集合,而标号图 g 的尺寸为该图包含的边数,记为 $|g|=|E(g)|$.

定义 2(图的同构) 图的同构是一个双射 $f: V(g)\leftrightarrow V(g')$.对于图 $g=(V,E,l,B)$ 与图 $g'=(V',E',l',B')$,若它们是同构的,则满足如下条件:

(1) $\forall u\in V, l(u)=l'(f(u))$

(2) $\forall u,v\in V, ((u,v)\in E)\Leftrightarrow((f(u),f(v))\in E')$,且 $\forall(u,v)\in E, l(u,v)=l'(f(u),f(v))$.

定义 3(子图同构) 给定标号图 g 与 g',若 g' 中存在某子图 g'' 与图 g 同构,则称 g 与 g' 是子图同构的,g'' 为 g 在 g' 中的一个"嵌入",记为 $E(g,g')$.此时,称 g 是 g' 的子图,记为 $g\subseteq g'$.称 g' 是 g 的超图,记为 $g'\supseteq g$.

例 1 图 1 所示为标号图集 D 中频繁子图集合.在这里,边被赋予标号,如:'a','b'或'c'.为方便描述,我们省略节点标号.节点被指定唯一一个编号值,该值来自于图挖掘算法的遍历顺序.图 1 中,f_2 包含 f_1 的两次嵌入,即 $|E(f_1,f_2)|=2$.类似地,$|E(f_7,f_{10})|=1$.

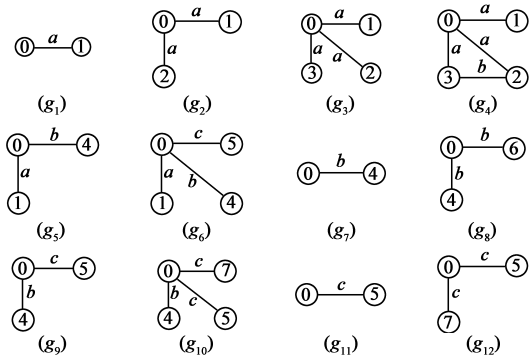

图1 标号图

2.2 图的标准代码

标准代码用于将标号图一一对应地转换为序列,并用该序列来描述该图.图可通过邻接表或邻接矩阵表达,为标号图转化为序列提供可能.然而,给定图 g,存在多个序列与之对应.设图 g 的多个序列按字母序排列,则无论其排列方式如何,总存在唯一的最大值或最小值.实际上,无论最大值或最小值,均使标号图与其编码建立起一个一一对应 $f: G\rightarrow S$,其中 G 代表图,而 S 代表表示图的序列.通过这种转换,给定两个标号图 g 与 g',如果它们是同构的,则有 $f(g)=f(g')$,否则,有 $f(g)\neq f(g')$.

本文用邻接表的一种变形描述标号图.标号图中每条边用 5 元组 (n_1,n_2,v_1,e,v_2) 描述,其中 n_1,n_2 为节点的序号,该序号是在进行先深挖掘时生成的,v_1,v_2 是节点的标号,e 是边的标号.本文约定,在所有边的 5 元组序列中,字母序最小的序列为标号图的标准代码[16].

2.3 图包含查询

定义 4(包含查询) 给定图数据集 $D=\{g_1,g_2,\cdots,g_n\}$,并给定查询图 q,包含查询的返回结果 Q_c 是 D 中所有被 q 包含的目标图,即 $Q_c=\{g_i\mid g_i\in D\wedge g_i\subseteq q\}$.

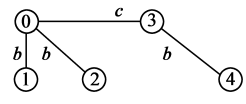

图2 查询图q

例如,设图 2 中给出查询图 $q,g_1,g_2,\cdots,g_{12}$ 为目标

图集,则包含查询的结果为 $Q=\{g_7,g_8,g_9\}$.

3 包含查询算法 CFG-Index

3.1 频繁子图与频繁闭图

定义 5(频繁子图) 给定图数据集 $D=\{g_1,g_2,\cdots,g_n\}$和最小支持度阈值 min_sup,如果一个子图 g 是频繁的,当且仅当它的支持度不小于 min_sup.其中,支持度为图集 D 中 g 的超图所占比例:

$$\text{support}(g,D)=\frac{|\{g_i|g_i\in D\wedge g\subseteq g_i\}|}{|D|} \tag{1}$$

频繁子图 f 的支持集是图集 D 中 f 所有超图的集合,记为:$D_f=\{g_i|g_i\in D\wedge f\subseteq g_i\}$.我们也可通过支持集来表示支持度:$\text{support}(g,D)=|D_g|/|D|$.

图集的频繁子图通过挖掘算法得到.由于效率较高的图挖掘算法均采用深度优先搜索(DFS,Depth First Search)来生成频繁子图,DFS 方式将谱系关系引入频繁子图之间.所有频繁子图构成一棵 DFS 树.该树的根是不包含任何节点与边的空图,即不包含任何节点和边的图.树的节点为频繁子图,任意两个邻居节点的关系为相差一条边的子图与超图.节点至根的距离为该节点的高度,与其包含的边数相等.从根节点至任意节点的路径为该节点所包含边的序列.

例 2 图 3 所示为图 1 中频繁子图对应的 DFS 树.其中,图 f_5 是图 f_6 的子图,在 DFS 树中,f_5是 f_6 对应的父亲节点.f_5 与 f_6 分别包含 2 条边和 3 条边,它们在 DFS 树中的高度分别为 2 和 3.由根至 f_6 存在一条路径,经过边的序列为 a,b,和 c,即 f_6 用边的序列表示为"abc".f_6 是 f_5 的一边超图,超出边的代码为'c'.在本例中,树中节点的描述形式为 f_i:$\text{support}(f_i,D)$.

频繁子图数量巨大,甚至远超图集本身.在索引的构建中,需要尽量减少模式之间的冗余.

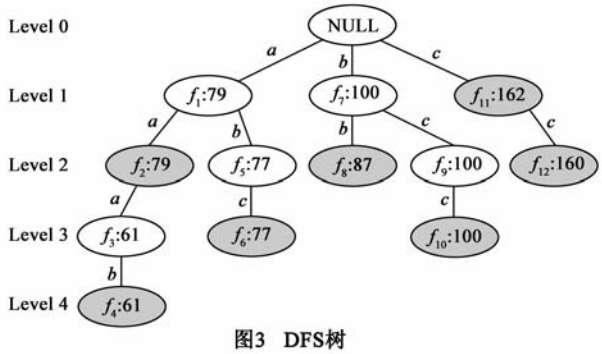

图3 DFS树

定义 6(频繁闭图) 给定图集上频繁子图集合 FG(Frequent Subgraph),频繁闭图集合 CFG(Closed Frequent Subgraph)是 FG 这样一个子集,CFG 中仅包含相同频率 FG 中的最大子图,即:

$$\text{CFG}=\{g\,|\,g\in\text{FG}(\wedge\nexists\, g'\in\text{FG such that } g\subseteq g' \text{ and } \text{support}(g,D)=\text{support}(g',D)\} \tag{2}$$

通常情况下,对于相同图集与 min_sup,FG 数量远大于相应 CFG 数量.例如,在 NCI/NIH 数据集中,支持度为 0.05 时,FG 个数约为 1,000,000 个,而 CFG 个数仅为 2,000 个左右.另外,实验结果表明[17],在图数据分析中,分析算法应用于 CFG 时,结果的精度与应用于 FG 上相比并无损失.该结果说明,CFG 的结构信息完全覆盖 FG.

CFG 集是 FG 集的一个子集,它具自己的性质.

性质 1 给定图集和支持度阈值,任意一个 FG 总存在一个具有相同支持度的 CFG 与之相应.

性质 2 给定图集和支持度阈值,CFG 集合可导出完整的 FG 集合.

结论 1 给定图集和 min_sup,CFG 集尺寸小于相应 FG 集,而且 CFG 集完全覆盖 FG 集结构信息.

结论 1 表明,CFG 比 FG 更具有应用于查询算法中的选择性质.由于 CFG 集包含完整 FG 集结构信息,采用 CFG 作为索引在不失精度的情况下得到更小的候选集和更小的索引集.候选集元素的个数决定验证阶段子图同构验证次数,候选集尺寸的减小带来算法效率的提高.

3.2 CFG-Index

图集的 CFG 集合仍然存在冗余.若模式 f 的支持集与其子模式支持集的交集(即 $\bigcap_{f'\subseteq f}S_{f'}$)近似,则 f 是冗余的.

定义 7(选择比) 一个模式 f 的选择比 δ 是该模式所有子模式支持集的交集与该模式自己支持集的比值,即:

$$\delta=\frac{|\bigcap_k S_{f_k}|}{|S_f|}(f_k\subseteq f) \tag{3}$$

任意模式 f 的支持集均小于其子模式支持集的交集,即 $\delta\geqslant 1$.因此,δ 值越大,表明模式 f 的过滤能力越强.

构建索引时,从最小尺寸的 CFG 开始,即 1-边模式.其余模式从 1-边模式计算得到.给定选择比阈值 $\delta_{\min}$,索引包括:(1)所有 1-边 CFG;(2)满足 $\frac{\bigcap_{f'\in F\wedge f'\subset f}S_{f'}}{S_f}\geqslant\delta$ 的 CFG.

例 3 图 3 中,f_{11}和 f_{12}是 CFG,其支持集为:$|D_{12}|=160$,$|D_{11}|=162$.因此,f_{12}的选择比为:$\delta=162/160=1.01$.模式 f_2 和 f_4 也是 CFG,而 f_4 的选择比则为 $\delta=1.30$.设给定选择比阈值为 $\delta_{\min}=1.2$,则仅有 f_4 被保留在索引中.

为进一步减少子图同构次数,CFG-Index 的索引是树状结构.该树来源于挖掘算法,称 DFS 索引树 T.T 的

根为空图，T 的叶节点均为强选择性 CFG. T 的中间节点，既存在选择性 CFG，也存在非选择性 CFG 或其它 FG，非选择性 CFG 和其它 FG 的加入是保持整棵树的完整.

命题 1 若一个图模式 f 是另一个图模式 g 的子图，则 f 的所有子模式也是 g 的子图. 否则，f 的所有超模式也不是 g 的子图.

命题 1 可用于进一步提高判断一个索引模式是否被 q 包含的筛选效率. 查询处理算法按照先深遍历的顺序遍历 DFS 索引树 T，一旦发现某节点 n 不是 q 的子图，以 n 为根的子树的所有节点必然不是 q 的子图.

3.3 查询算法

查询算法如算法 1 所示. 算法开始时调用 SMining 得到所有不被 q 包含的索引模式集合，即：$F=\{f \mid f\in T \wedge f\not\subseteq q\}$. 每个索引模式在挖掘过程中记录下它们在图集中超图 ID. 在算法第二行，得到这些索引模式支持集的并集，即：$\bigcup_{f\in F}\{G_i \mid G_i\in D \wedge f\subseteq G_i\}$. 根据命题 1，这些目标图不能被 q 包含，因此，候选集是该集合关于图集 D 的补集(算法 1 第 3 行). 最后，算法进行验证，对得到的候选集中的目标图逐个与 q 进行子图同构测试，得到最终结果(算法 1 第 4～9 行).

算法 1 CFG-Index

Input：查询图 q，图集 D，DFS tree T
Output：结果集 C
1: $F \leftarrow$ SMining(q, T);
2: $D_{\text{List}} \leftarrow$ union of supergraph *ID* lists of features in F;
3: $C \leftarrow D - D_{\text{List}}$;
4: for every graph c in C
5: verification(c);
6: if c is contained by q
7: continue;
8: else
9: prune c from C;
10: return C.

函数 SMining 在算法 2 中给出. 它的输入是一个查询图 q 和索引树 T 的当前节点 *current_Node*. 算法开始时，当前节点为 *root* 节点. SMining 的输出为所有不被 q 包含的索引模式.

SMining 筛选索引模式方法如下. 当查询图到达时，SMining 开始在树 T 上执行先深遍历，设 *root* 节点为 *current_Node*. 当 *current_Node* 的儿子节点 s_i 是 q 的子图，记录下 s_i 在 q 中的 *Embedding* 并将 s_i 设为 *current_Node*. 这个过程递归进行，直到 SMining 到达叶节点或某个非 q 子图为止(算法 2 第 4～6 行). 若 *current_Node* 不是 q 的子图，根据命题 1，该节点所有后代均不会被 q 包含(算法 2 第 8～10 行). 在该情况下，SMining 将此节点及其后代中所有选择性 CFG 加入到 F 中去.

算法 2 SMining

Input：查询图 q，T 中当前节点 *current_Node*，子图的嵌入 E(*current_Node*, q)
Output：不包含 q 的索引模式
1: $F \leftarrow \Phi$;
/* Features are not contained by q. */
2: for every son s_i of *current_Node* do
3: if $s_i \subseteq q$
4: *current_Node* $\leftarrow s_i$;
5: $e \leftarrow E(s_i, q)$;//s_i 在 q 中的嵌入
6: SMining(q, s_i, e);
7: else /* $s_i \not\subseteq q$ */
8: $O \leftarrow$ all discriminative CFGs in the offspring of s_i;
9: $F \leftarrow F \cup O$;
10: prune subtree rooted at s_i from T;
11: endif
12: endfor
13: return F;

例 4 给定图 2 查询图 q，SMining 在图 3 所示树 T 先深遍历，将根节点设为当前节点，F 置空. 按先深顺序，SMining 检查儿子 f_1，f_1 为单边，且标号为 'a'，非 q 子图，算法直接将以 f_1 为根的子树删除，并将 f_1 的后代中具有选择性索引模式 f_2，f_4，f_6 加入到 F. 此时，SMining 计算 f_7. 在 q 中，$E(f_7, q)=3$. SMining 记录 $E(f_7, q)$，并继续顺序遍历 f_7 的儿子. 当 SMining 计算 f_7 所有后代后，仅有 f_{10} 被加入 F. 同样地，SMining 检查 f_{11} 为根的子树. 当 SMining 结束时，$F=\{f_2, f_4, f_6, f_{10}, f_{12}\}$. 当 F 返回给 CFG-Index 时，F 中所有索引模式支持集取并，并对 D 取补形成候选集.

CFG-Index 相比 cIndex 有以下三个改进：(1)通过遍历树 T 与 q 进行子图同构增量验证，大大简化子图同构的复杂性；(2)索引按照树的形式组织，可迅速剪枝；(3)索引模式的选择，不但排除冗余，而且能获得更小的候选集.

4 性能分析

当提出 q 时，CFG-Index 查询的反应时间为：

$$T_{response} = T_{search} + |F| \times T_{isSubQ} + |C_q| \times T_{isSuperG} \quad (4)$$

其中，$T_{response}$ 是查询反应时间，$|F|$ 是索引尺寸，$|C_q|$ 是候选集尺寸，T_{search} 是计算候选集时间，T_{isSubQ} 是 q 与索引模式之间子图同构时间，$T_{isSuperG}$ 是验证阶段 q 与候选图之间子图同构时间. 其中，T_{search} 可以忽略不计. 为减小响应时间，算法必须尽量减小 F 和 C_q 的尺寸. 然而，CFG-Index 中 $|C_q|$ 的尺寸被频繁闭图的强选择性制约，能得到更小的候选集. 因此，在本节中，我们将着重分析 CFG-Index 中的 $|F|$.

4.1 索引模式过滤

包含查询利用排除逻辑减小候选集尺寸.算法通过 q 与索引模式之间的子图同构确定哪些模式不被 q 包含.目前广泛采用的子图同构算法是文献[18,19]提出的,和改进算法如文献[20].这类算法基于枚举方法,虽然采用剪枝等手段,其复杂性仍然高达 $O(n!)$.这里,n 是最大边数.

CFG-Index 利用 DFS 索引树来筛选某索引模式是否为 q 的子图.当 q 被提出时,CFG-Index 并非枚举出 q 所包含模式,是通过先深遍历索引树 T,增量进行计算.

定理 1　如果 DFS 索引树上的某个模式被查询图包含,则它必然能被 CFG-Index 检出.

证明　采用归纳法证明.

设 f 为 DFS 树 T 中的节点,q 为查询图.

(1)当 $|f|=1$ 时,f 是根节点的儿子,CFG-Index 将检查 q 中是否存在 $E(f,q)$,因此,如果 f 被 q 包含,则必然能被 CFG-Index 检出.

(2)假设当 $|f|=k$ 时,q 包含 f 时,能被 CFG-Index 检出.

在此假设基础上,若 $|f'|=|f+e|=k+1$ 且被 q 包含时,能被 CFG-Index 检出,则定理得证.

由于 $f\subset f'$,根据 4.3 节规则 1 给定的条件,若 f' 被 q 包含,则 f 必被 q 包含,由该性质得出,当 $|E(f,q)|=n$,$|E(f',q)|=n'$ 时,必有 $n\leqslant n'$,即 $E(f',q)$ 必定在 $E(f,q)$ 的基础上生成.同时,CFG-Index 算法记录下了 f 的每个 $E(f,q)$,并且,DFS 树所有节点按频繁子图挖掘顺序先深生成,DFS 树通过边记录下了 f' 相对于 f 增长边的位置.因此,若 f 被 q 包含能被 CFG-Index 检出时,f' 被 q 包含也必能被 CFG-Index 检出,即当 $|f'|=k+1$ 时,命题成立.

综合 1,2,定理得证.

根据定理 1,CFG-Index 是准确的和完备的.CFG-Index 通过增量的子图同构测试提高了查询处理效率.根据 SMining 的处理方式,设当前节点 n_c 的标准代码为 $S_c=(n_0e_0n_0')(n_1e_1n_1')\cdots(n_ke_kn_k')$,且其为 q 的子图,SMining 将检查它的儿子节点 $s_1,s_2,\cdots,s_r$ 是否为 q 的子图.根据 DFS 树的特点,每个儿子 s_i 相比 S_c 增加一条特定位置的边 e_i',即 $s_i=S_c\cup(n_te'_in_i'')$,其中 n_i 为该边的起始节点,e_i' 为该边的标号,n_i'' 为该边的终止节点.此时,CFG-Index 仅仅按照 T 的儿子 s_i 所增加边的位置进行检验,即,若 q 中存在一条边始于 $E(S_c,q)$ 中位置 n_i,且其边的标号为 e'_i,则 S_c 的这个儿子存在 $E(s_i,q)$.因此,SMining 仅需检查 $E(S_c,q)$ 中节点 n_i 是否存在这样的一条边即可.通过这种方式,CFG-Index 大大节省了子图同构的响应时间.

4.2　折半查找

若当前节点是查询图 q 的子图,CFG-Index 将逐步考察当前节点为根的子树中所有索引模式.在这个过程中,可采用折半查找方法来进一步提高效率.假设有一条从跟到叶的路径 $rn_1n_2\cdots n_k$,其中 r 为树 T 的根,n_i 是该路径上的一个节点,i 代表该节点的高度,k 代表该路径的长度.此时,若 n_1 是 q 的一个子图,算法并非线性考察该路径上所有节点,而选择该路径上的中间节点,即 $n_{\lceil k/2\rceil}$,而不是 n_2.此时,遍历的时间复杂性由 $O(k)$ 降低至 $O(\log k)$.

5　实验结果

为评价算法 CFG-Index 的效率与准确性,我们与目前唯一的算法 cIndex 进行了比较.试验分两部分,分别采用真实数据和模拟数据.

实验环境为 Pentium VI 3.2GHz,512M 内存,120G 硬盘,操作系统为 Windows XP Professional SP3.CFG-Index 和 cIndex 的编译环境均为 gcc/g++.

5.1　真实数据(AIDS Antiviral Screen Dataset)

实验数据做如下预处理:随机从 NCI/NIH 数据集中选择 10,000 个图作为查询集.同时,将该集合 5 等分,记为 $D_1,\cdots,D_5$,其中 D_1 作为查询图集,另 4 个作为查询日志.cIndex 中的相对子图按照 $\sigma_c=0.05$ 挖掘得出.最小平均支持度介于 0.1 和 0.01 之间.而且,在 cIndex-TopDown 算法中,每个叶节点的最小查询数被设置为 100.CFG-Index 的最小支持度 min_sup 是一个可变的参数,用于调整索引尺寸.这个变化的过程是这样的,图的平均尺寸越小,则 min_sup 的值越大.通过可变 min_sup 和选择比 δ 进行调整,我们将索引模式个数控制在 200 个左右.图集 D_{init} 来自挖掘结果,支持度在[0.5% ~ 10%]之间.最终的测试数据集 $D_{10,000}$ 包含 10,000个图,在 D_{init} 中随机选取.

实验中,CFG-Index 的查询效率与 cIndex-TopDown 进行了比较,之所以采用 cIndex-TopDown,是由于该算法在文献[15]中提出的三种算法中效率最高,它的平均查询时间大约是 cIndex-Basic 和 cIndex-BottomUp 的 1/2.为便于比较这两种算法,2,000 个查询图按照查询结果的平均个数(即数据集中被查询图包含的图的个数)划分为 8 个单元,分别为[0,10),[10,20),[20,30),[30,40),[40,100),[100,200),[200,500),[500,∞).

图 4 给出了查询效率的比较,结果为 CFG-Index 的效率大约是 cIndex-TopDown 的 3 倍左右.查询时间的降低来自于平均候选集尺寸的减小,同时,索引与 q 之间增量的子图同构测试也是一个原因.

为进一步验证频繁闭图的强选择性,在图 5 中给出候选集尺寸的比较结果.图中 X 轴与 Y 轴分别表示查询

结果的平均尺寸和候选集的平均尺寸.由于 cIndex 的候选集明显大于 CFG-Index 的候选集,参照图 4,查询效率与候选集尺寸成正比.这也证明了闭图的选择性强.

为评价算法的可扩展性,在 D_{init} 中随机选择 4 个数据集,它们包含图的个数分别为 10,000 个到 70,000 个,彼此之间相差 20,000 个.查询效率在图 6 中显示,而查询的候选集在图 7 中显示.

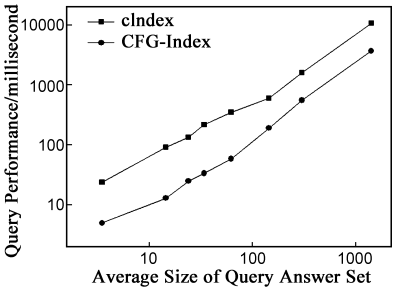

图4 算法的查询效率

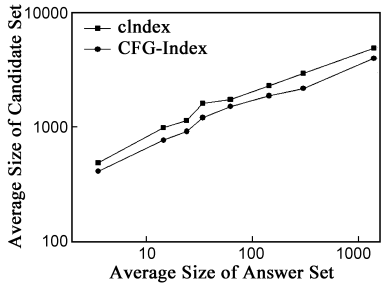

图5 平均候选集尺寸

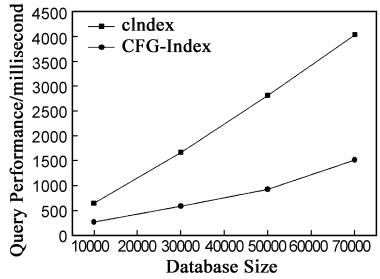

图6 不同数据集尺寸下的查询效率

综合上述实验结果,CFG-Index 效率的提升比例与候选集尺寸的降低并不成比例,这由于 CFG-Index 在索引模式鉴别阶段采用了索引树 T 进行剪枝及简化子图同构复杂性的工作.

5.2 模拟数据

本小节给出基于模拟数据的实验结果,利用文献[21]中给出的图数据生成器生成数据.

为弥补模拟数据的不足,在模拟数据实验中,图数据集和查询图分别生成.生成数据的命名规则如下:若数据集图的个数为 10,000 个,图的平均尺寸为 15 条边,潜在的频繁图平均尺寸为 5 条边,频繁图的个数为 100 个,不同的标号个数为 5 个时,该数据集命名为 D10kN5I5T15L100(可参照表 1 中给出的参数).模拟的图数据集为 D10kN5I5T15L100,而查询图集分别为 D10kN5I5T40L100,D10kT45,D10kT50,D10kT55,D10kT60.为简化描述,我们按顺序称这些查询图集为 Q1,Q2,Q3,Q4 和 Q5.每个查询集划分为 8,000 个查询日志和 2,000 个实际查询.

表 1 图数据模拟器的控制参数

参数	描述	例
D	生成模拟数据集中图的个数	D10k
N	标号数	N5
T	生成数据中图的平均边数	T15
I	频繁子图平均尺寸	I5
L	频繁子图个数	L100

图 8 和图 9 分别给出平均查询效率和平均候选集尺寸的比较结果.可以看出,CFG-Index 在模拟数据集上的表现也明显优于 cIndex-TopDown.而且,随着查询图复杂性的提高,两者的差别也基本维持在同一比例上.

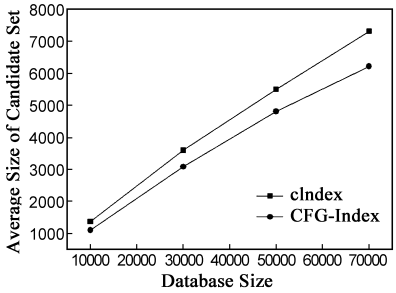

图7 不同数据集下的平均候选集尺寸

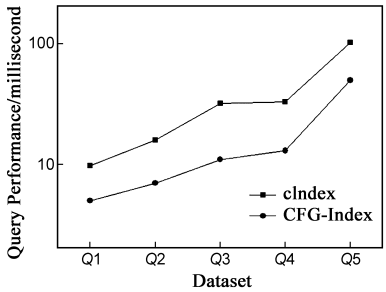

图8 模拟数据集上的查询效率

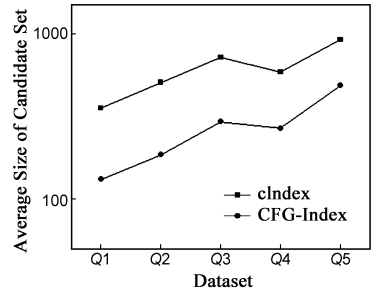

图9 模拟数据集上的平均候选集尺寸

6 结论

本文提出一种基于闭图索引的图包含查询算法.此算法采用过滤-验证机制.与 cIndex 中采用选择性无冗余图模式作为索引不同,本文利用频繁闭图集的子集作为索引,并采用树状索引结构,使得在索引模式过滤阶段增量进行子图同构,降低了子图同构的复杂性.性能分析和实验结果表明,CFG-Index 的执行效率和可扩展性均明显优于 cIndex.

参考文献:

[1] X Yan, P S Yu, et al. Graph indexing: a frequent structure based approach[A]. Proc SIGMOD'04[C]. USA: ACM, 2004.335 – 346.

[2] S Zhang, M Hu, et al. TreePi: a novel graph indexing method [A]. Proc ICDE'07[C]. USA: IEEE, 2007.966 – 975.

[3] J Cheng, Y Ke, et al. FG-index: Towards verification free query processing on graph databases[A]. Proc SIGMOD'07[C]. USA: ACM, 2007.857 – 872.

[4] P Zhao, J X Yu, et al. Graph Indexing: Tree + Delta > = Graph [A]. Proc VLDB'07[C]. USA: VLDB, 2007. 938 – 949.
[5] X Yan, P S Yu, et al. Substructure similarity search in graph Databases[A]. Proc SIGMOD'05[C]. USA: ACM, 2005. 766 – 777.
[6] X Yan, F Zhu, et al. Searching substructures with superimposed distance[A]. Proc ICDE'06[C]. USA: IEEE, 2006. 88.
[7] 高琳,覃桂敏,周晓峰.图数据中频繁模式挖掘算法研究综述[J].电子学报,2008,36(8):1603 – 1609.
Gao Lin, Tan Gui-Min, et al. An overview of algorithms for mining frequent patterns in graph data[J]. Acta Electronic Sinica, 2008, 36(8): 1603 – 1609. (in Chinese)
[8] D Shasha, J T Wang, et al. Algorithmics and applications of tree and graph searching [A]. Proc PODS' 02 [C]. USA: ACM, 2002. 39 – 52.
[9] H He, A K Singh. Closure-tree: an index structure for graph queries[A]. Proc ICDE'06[C]. USA: IEEE, 2006. 38.
[10] D W Williams, J Huan, et al. Graph database indexing using structured graph decomposition [A]. Proc ICDE' 07 [C]. USA: IEEE, 2007. 976 – 985.
[11] H Jiang, H Wang, et al. GString: a novel approach for efficient search in graph databases [A]. Proc ICDE' 07 [C]. USA: IEEE, 2007. 566 – 575.
[12] Zou L, Chen L, et al. A novel spectral coding in a large graph database[A]. Proc EDBT'08[C]. German: Springer-Verlag, 2008. 181 – 192.
[13] P Bohannon, W Fan, et al. Information preserving XML schema embedding [A]. Proc VLDB [C]. USA: VLDB, 2005. 85 – 96.
[14] K Gupta, D Suciu. Stream processing of XPath queries with predicates[A]. Proc SIGMOD[C]. USA: ACM, 2003. 419 – 430.
[15] C Chen, X Yan, et al. Towards graph containment search and indexing[A]. Proc VLDB'07[C]. USA: VLDB, 2007. 926 – 937.
[16] LI Xian-Tong, LI Jian-Zhong, GAO Hong. An efficient frequent subgraph mining algorithm [J]. Journal of Software, 2007, 18, (10): 2469 – 2480.
[17] X Yan, J Han. CloseGraph: mining closed frequent graph patterns[A]. Proc KDD[C]. USA: ACM, 2003. 286 – 295.
[18] J R Ullmann. An algorithm for subgraph isomorphism[J]. J ACM, 1976, 23(1): 31 – 42.
[19] L P Cordella, P Foggia, et al. A (sub) graph isomorphism algorithm for matching large graphs [J]. IEEE Trans Pattern Anal Mach Intell, 2004, 26(10): 1367 – 1372.
[20] M Kuramochi, G Karypis. An efficient algorithm for discovering frequent subgraphs [J]. Tran Knowledge and Data Eng, 2004, 16(9): 1038 – 1051.
[21] M Kuramochi, G Karypis. Frequent subgraph discovery[A]. Proc of ICDM 2001[C]. USA: IEEE, 2001. 313 – 320.

作者简介:

李先通　男,1973 年生于黑龙江哈尔滨.1995 年毕业于哈尔滨工业大学金属材料及工艺系,2003 年于哈尔滨工业大学计算机科学与技术学院攻读博士学位,研究内容包括数据挖掘、图挖掘、图查询、海量数据处理等,现就职于哈尔滨工业大学交通科学与工程学院.
E-mail: lxt@hit.edu.cn

安　实(通讯作者)　男,1968 年出生,博士,教授、博士生导师.1990 年于哈尔滨工业大学机械制造与工艺专业毕业,2000 年获得哈尔滨工业大学管理科学与工程专业的博士学位,2002 年破格晋升为教授,现任哈尔滨工业大学交通安全特种材料与智能化控制技术交通行业重点实验室主任和黑龙江省智能交通管理与技术重点实验室主任.
E-mail: anshi@hit.edu.cn