软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1792−1812 [doi: 10.13328/j.cnki.jos.005268]

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

基于包含度的子图匹配方法*

李瑞远[1], 洪 亮[2,3]

[1](西安电子科技大学 计算机学院,陕西 西安 710071)

[2](武汉大学 信息管理学院,湖北 武汉 430072)

[3](武汉大学 深圳研究院,广东 深圳 518000)

通讯作者: 洪亮, E-mail: hong@whu.edu.cn

摘 要: 子图匹配是图论中最基本的操作.研究子图匹配的一个变种,即:在一个节点拥有若干元素的大图数据库中,找到与给定查询图结构同构并且对应节点元素的加权集合包含度大于给定值的所有子图,称作基于包含度的子图匹配(subgraph matching with inclusion degree,简称 SMID).该查询能够应用于多种场景,包括论文检索、社区发现、企业招聘等.为高效实现 SMID,设计了同时包含节点元素和图结构信息的数据签名与查询签名,在离线处理阶段,利用数据签名为数据图建立动态签名树(DS-tree),以加快在线处理时图节点的匹配过程.为解决 DS-Tree 占用空间大的问题,设计了一种 DS-Tree 压缩方法,在对查询效率影响不大的情况下减小了索引空间.为进一步加快查询效率,还提出了支配子图查询算法.在真实数据和人工数据上的实验结果表明,所提出的方法在效率和扩展性方面优于现有其他方法.

关键词: 子图匹配;包含度;图数据库;索引;支配子图

中图法分类号: TP311

中文引用格式: 李瑞远,洪亮.基于包含度的子图匹配方法.软件学报,2018,29(6):1792−1812. http://www.jos.org.cn/1000-9825/5268.htm

英文引用格式: Li RY, Hong L. Method for subgraph matching with inclusion degree. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1792−1812 (in Chinese). http://www.jos.org.cn/1000-9825/5268.htm

Method for Subgraph Matching with Inclusion Degree

LI Rui-Yuan[1], HONG Liang[2,3]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

[2](School of Information Management, Wuhan University, Wuhan 430072, China)

[3](Shenzhen Research Institute of Wuhan University, Shenzhen 518000, China)

Abstract: Subgrpah matching is a basic operation in graph theory. This paper focuses on a variant, namely subgraph matching with inclusion degree (SMID), which retrieves subgraphs that are structurally isomorphic to the query graph while satisfying the condition that the inclusion degree between matched vertexes is greater than a given value. SMID can be applied to many applications, including paper search, crowdsourcing, and recruitment. To efficiently process SMID, this paper designs a novel signature mechanism for data graph and query graph respectively by holding the information of both vertex elements and graph structure. Based on graph signature, a dynamic

* 基金项目: 国家重点研发计划(2016YFB1000603); 国家自然科学基金(61303025); 深圳市基础研究项目(JCYJ20160523160953223); NSFC-广东联合基金; 国家超级计算广州中心

Foundation item: National Key Research and Development Program of China (2016YFB1000603); National Natural Science Foundation of China (61303025); Shenzhen Basic Research Project NSFC-Guangdong Joint Fund (JCYJ20160523160953223); National Supercomputing Center in Guangzhou Project

收稿时间: 2016-10-26; 修改时间: 2016-12-12, 2017-01-19; 采用时间: 2017-02-17; jos 在线出版时间: 2017-03-31

CNKI 网络优先出版: 2017-03-31 21:54:36, http://kns.cnki.net/kcms/detail/11.2560.TP.20170331.2154.004.html

signature tree (DS-Tree) is built to speed up the SMID processing. A compression method is proposed to reduce the memory usage of DS-Tree. To achieve a better performance, an efficient dominating-set-based subgraph matching algorithm is also developed. Extensive experiments on both real and synthetic datasets demonstrate that the method introduced in this paper outperforms state-of-the-art methods by an order of magnitude in terms of efficiency and scalability.

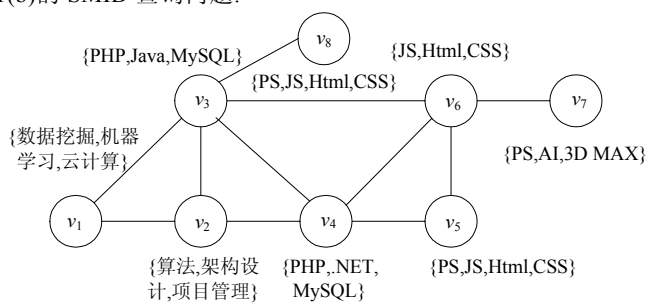
Key words: mesh parameterization; parameter domain; parameterization quality; bijective mapping; metric distortion

近年来出现的许多网络,如社会网络[1]、语义网络[2]、通信网络以及生物网络[3],大部分都可以自然地建模成图数据库(graph databases).图数据库越来越广泛地成为一种建模和查询图数据的重要工具.在各种图操作中,子图匹配(subgraph match),亦称子图查询(subgraph query)[2−6]是最基本的操作:给定一个由实际网络建模成的图数据库 G(称为数据图)和用户指定的查询图 Q,找出 G 中所有精确匹配 Q 的子图,这些子图不仅结构与 Q 相同,且对应节点的标签也完全相同[3].然而在实际的图数据库中,每个节点可能包含很多表示该节点特征的元素.在查询时,一方面,用户可能只关心部分节点元素是否匹配,而对其他元素是否匹配并不关心,即,用户对不同元素的关心程度不同;另一方面,用户可能无法知道图数据库中节点的所有元素,因此难以给出精确匹配的条件,导致精确图匹配无法进行.

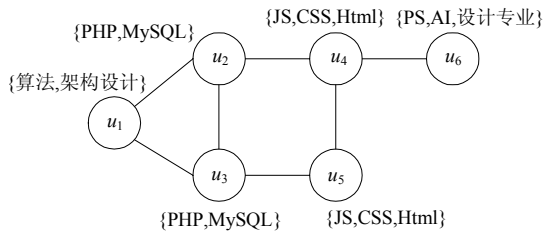
基于上述观察,本文提出子图匹配的一个变种——基于动态权重包含度[7]的子图匹配(subgraph match with inclusion degree,简称 SMID).在 SMID 查询中,每个图节点拥有一个元素集合,而不是一个标签,且每个元素对应一个动态权重,元素的权重由用户在查询时指定.具体而言,给定拥有 n 个节点$\{u_1,u_2,\ldots,u_n\}$的查询图 Q, SMID 查询能够找出图数据库 G 中所有包含 n 个节点$\{v_1,v_2,\ldots,v_n\}$的子图 X,满足:(1) $S(u_i)$和 $S(v_j)$的加权包含度大于用户指定的阈值,其中,u_i 与 v_j 对应,$S(u_i)$与 $S(v_j)$分别表示 u_i 与 v_j 的元素集合,$i,j\in\{1,2,3,\ldots,n\}$;(2) X 与 Q 结构上同构.

例 1:众包(crowdsourcing).

众包是互联网带来的新的生产组织形式,企业利用互联网分配工作、发现创意或解决技术问题.图 1(a)是某众包社区中用户的合作关系图,其中:每个节点代表一个用户,节点旁的集合表示用户的技能,节点之间的连线表示用户的合作关系.设想某企业拟从中选择 5 位用户为其开发项目,其要求如图 1(b)所示.于是,该问题便转换成了在图 1(a)中完成图 1(b)的 SMID 查询问题.



(a) 众包的合作图



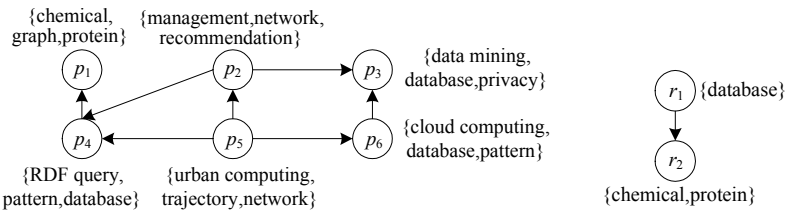
(b) 项目需求

Fig.1 An example of crowdsourcing

图 1 众包的例子

例 2:论文检索[8].

图 2(a)表示部分论文的引用图,其中,每个节点表示一篇论文,节点旁的单词表示该论文的关键字集合,边表示论文之间的引用关系.例如,论文 p_4 引用了论文 p_1.如果需要找到引用了“生物蛋白质”论文的“数据库”论文,那么该检索就转化成了图 2(b)在图 2(a)上的 SMID 查询.



(a) 论文引用图 (b) 论文检索图

Fig.2 An example of paper search

图 2 论文检索的例子

实际上,若将图 1(a)的节点看成社交网络的用户,节点旁的集合表示对应用户的兴趣集合,边表示用户之间的好友关系,那么社区发现[9]也可以转化为 SMID 查询问题.可见,SMID 在很多应用中都非常有用.然而,目前关于结构同构和节点元素包含度子图匹配问题的研究并不多.由于对子图匹配定义不同,此前精确子图匹配或模糊子图匹配的相关技术[2−5,10,11]不能直接用于 SMID.

有两种直接修改现有算法的方法来解答 SMID 问题.

- 第 1 种方法首先利用现有子图同构算法[10,12]找出查询图的所有同构子图,然后验证对应节点的包含度,过滤掉那些不满足条件的同构子图;
- 第 2 种方法顺序相反:首先找出每个查询图节点的候选节点,这些候选节点满足包含度限制;然后,从这些候选节点验证结构是否同构.

然而,这两种方法通常会产生很高的查询代价,对大图数据库而言更是如此.因为在第 1 步的过程中,第 1 种方法忽略了包含度的限制,而第 2 种方法没有考虑到图结构信息.

由于现有方法在效率上存在问题,本文提出一种更为高效的 SMID 方法:首先为图数据库节点签名,并将节点签名组织成动态签名树(dynamic signature tree,简称 DS-Tree),在查询过程中同时考虑了查询图的拓扑结构和图节点加权包含度的限制,减少了搜索空间.此外,本文还提出了支配子图(dominating subgraph)查询算法,进一步加快 SMID.

本文主要贡献总结如下:

(1) 提出了一种新的子图查询问题 SMID.与传统子图查询不同的是,SMID 要求图结构同构,且对应节点的加权包含度大于用户给定阈值.这种问题在现实生活中非常普遍;

(2) 设计了一种利用 DS-Tree 进行 SMID 查询的策略,DS-Tree 是一种动态平衡签名树.首先,分别针对数据图和查询图设计了两种签名机制,并将数据图节点签名组织成 DS-Tree.注意,这两种签名均包含了节点元素信息和图结构信息.然后,利用 DS-Tree 找出每个查询图节点的候选节点集合.最后,根据候选节点集合确定同构子图;

(3) 为减少 DS-Tree 占用的内存空间,设计了一种 DS-Tree 的压缩方法,在对查询效率影响不大的情况下,减少了内存空间占用;

(4) 为进一步提高 SMID 的查询效率,提出了支配子图查询算法.首先找到查询图的最小支配子图,然后利用签名树找到支配子图每个节点(称为查询图的支配节点)的候选节点集合.根据距离保留定理,利用支配节点的候选节点找到非支配节点的候选节点集合,最后确定查询图的同构子图;

(5) 通过人工数据和真实数据的实验结果表明,本文提出的 SMID 查询方法具有较好的效率和扩展性.

1 预备知识

本节首先总结子图匹配的相关工作,然后形式化定义 SMID 查询,最后给出 SMID 查询的基本算法框架.

1.1 相关工作

子图匹配问题可以大致分为两类:精确子图匹配和模糊子图匹配.

精确子图匹配要求所有的节点和边都完全匹配.Ullmann 提出的子图同构算法[12]和 VF2 算法[10]没有利用任何索引结构,因此对于大图数据库来说代价通常很高.Zou 等人[2]提出了一种新的签名索引和过滤规则来解决语义子图查询问题.Zhao 等人[4]研究的 SPath 算法利用节点周围的最短路径作为索引单元.Shang 等人[13]提出的 QuickSI 算法主要解决子图同构的检查,它基于图的一些静态信息确定搜索顺序,从而减少了搜索空间.近年来,Han 等人[14]提出的 Turboiso 算法在效率和扩展性两个方面都优于之前子图同构算法.Ma 等人[15]结合传统的关系型数据库技术设计了一种高效图查询引擎,能够处理精确子图匹配问题.然而,现有精确子图匹配没有考虑节点元素的相似性,在候选节点很多的情况下,查找同构子图的代价很高.

模糊子图匹配允许一些节点或边不匹配,Closure-tree[16]是第一个同时支持精确子图匹配和模糊子图匹配的图索引.TALE[3]模糊查询算法是一种基于索引技术的算法,它允许某些节点不匹配和某些边缺失.SAPPER[5]算法提出了一种基于混合网络单元的新型索引结构来解决模糊子图查询,它允许一些边缺失.文献[17]提出了复杂生物分子网络中基于半 Markov 随机游走的迭代加权子图查询算法,综合考虑了节点相似性及结构相似性,但其主要目标是提高生物分子的匹配准确率,并未建立有效的索引机制.Ye 等人[18]在集群上实现了网页图结构的近似子图匹配.本文要研究的问题与这些模糊子图查询问题不同,并不允许任何边和节点缺失,但只要求对应匹配节点元素集合的加权包含度大于一给定值.

社交网络中的组队问题[19,20]与 SMID 查询问题也不同.

- 首先,组队问题不需要指定查询图的结构,只需给出技能集合;而 SMID 需要用户给定查询图的结构以及每个查询节点的元素集合;
- 其次,二者的匹配条件不同:组队问题要求在社交网络中找出满足给定条件(如包含所有技能、地理位置相近、任务分配均匀等)的节点,而 SMID 查询需要从数据库中找到所有与查询图结构同构,并且对应节点的包含度大于给定阈值的所有子图;
- 最后,组队问题不考虑边的连接条件,也不要求查询结果中的节点连接;而 SMID 查询要求查询结果与给定查询图同构.

近年来,研究人员提出了一些新的近似子图查询问题.NeMa[21]集中于满足下列两个条件的子图查询问题:(1) 多对一的子图匹配;(2) 节点的元素相似.S4 系统[1]能够找出那些与查询图结构完全相同并且语义相似的子图.与它们不同的是,SMID 考虑的是一对一的结构同构,并且节点元素加权包含度满足一定限制.Zou[22]提出了 top-k 子图匹配问题,这种方法考虑了两个匹配实体间的相似性,但它假定节点间的相似性已经给定,也没有利用集合相似的过滤技术来优化匹配性能.Peng 等人[23]提出了一个查询 RDF 图的分布式处理框架.文献[24,25]分别提出了两种基于图编辑距离的 RDF 近似查询问题.Lian[26]针对 RDF 图可能不一致提出了一种基于概率的近似匹配问题,但是 RDF 图节点通常只有一个标签,这与 SMID 查询不同.Liang 等人[27]提出的 SMS^2 查询问题首次考虑了图节点拥有多个元素,它要求图结构同构,并且对应节点元素的集合相似度大于给定阈值.然而,集合相似度不能较好地对很多现实问题进行建模,例如,用户通常不知道图数据库节点的所有元素,因此无法完整给出 SMS^2 的查询条件.本文提出的 SMID 查询,只需要用户给出关心的部分元素及其权重即可.

1.2 问题定义

一个网络可以建模成图 $G=\langle V(G),E(G)\rangle$,称为数据图,其中,$V(G)$表示顶点集合,$E(G)\subseteq V(G)\times V(G)$是边集合,每个顶点 $v\in V(G)$拥有一些元素,记为 $S(v)$,所有节点的元素全集记为$\Sigma(G)$.

类似地,查询图也可以表示成 $Q=\langle V(Q),E(Q)\rangle$.本文讨论无向简单图中的 SMID 查询,不失一般性,我们的算法能够扩展到有向简单图中.

包含度(inclusion degree)是模糊集理论中的概念,是一种描述不确定性关系的有效度量方法[7],其定义为:

定义 1(包含度). 设$(L,\leqslant)$为偏序集.若对于任意$x,y\in L$,有数字$ID(y/x)$与之对应,且满足:

(1) $0\leqslant ID(y/x)\leqslant 1$;

(2) 若$x\leqslant y$,$ID(y/x)=1$;

(3) 若$x\leqslant y\leqslant z$,$ID(x/z)\leqslant ID(x/y)$;

(4) 若$x\leqslant y$,对于任意的$z\in L$,有$ID(x/z)\leqslant ID(y/z)$.

则 ID 称为偏序集$(L,\leqslant)$上的包含度.

定义 2(集合包含度(set inclusion degree)). 对于集合X和Y,X为非空集合,$SID(Y/X)$定义为

$$SID(Y/X)=\frac{|Y\cap X|}{|X|} \tag{1}$$

其中,|*|表示集合内的元素个数,∩表示集合交运算.

容易验证,SID满足:

(1) $0\leqslant SID(Y/X)\leqslant 1$;

(2) 若$X\subseteq Y$,$SID(Y/X)=1$;

(3) 若$X\subseteq Y\subseteq Z$,$SID(X/Z)\leqslant SID(X/Y)$;

(4) 若$X\subseteq Y$,对于任意的非空集合Z,有$SID(X/Z)\leqslant SID(Y/Z)$.

称SID为集合包含度.

定义 3(动态加权集合包含度(dynamic weighted set inclusion degree)). 对于集合X和Y,a是X或Y内的元素,$W(a)$表示元素a的权重,由用户在每次查询前指定,其中,$0\leqslant W(a)\leqslant 1$.

$WID(Y/X)$定义为

$$WID(Y/X)\frac{\sum_{a\in Y\cap X}W(a)}{\sum_{a\in X}W(a)} \tag{2}$$

称WID为动态加权集合包含度.为简单起见,若无特殊说明,下文中的包含度均指动态加权集合包含度.

定义 4(基于包含度的子图匹配(subgraph match with inclusion degree,简称 SMID)). 给定数据图$G,V(G)=\{v_1,\ldots,v_m\}$,查询图$Q,V(Q)=\{u_1,\ldots,u_n\}$,用户指定的各元素权重以及包含度阈值$\tau$,当且仅当满足下面 3 个条件,我们说$Q$与$G$的子图$X,V(X)=\{v_1,\ldots,v_n\}$基于包含度的子图匹配:

(1) 存在双射函数f,对每个$u_i\in V(Q)$和$v_j\in V(X)$,均有$f(u_i)=v_j$.其中,$1\leqslant i,j\leqslant n$;

(2) $WID(S(u_i),S(v_j))\geqslant\tau$,其中,$S(u_i)$和$S(v_j)$分别表示$u_i$和$v_j$的元素集合,$WID(S(u_i),S(v_j))$表示$S(u_i)$和$S(v_j)$的加权包含度;

(3) 对于任意边$(u_i,u_k)\in E(Q)$,均有$(f(u_i),f(u_k))\in E(X)$,其中,$1\leqslant k\leqslant n$.

由于$SMID$与边是否有向无关,因此,我们的方法同时适应于有向图和无向图的情况.

本文利用加权包含度作为节点是否匹配的计量方法有以下两个原因.

(1) 包含度能够较好地建模很多实际问题,例如,用户往往无法知道数据图中节点特征集合的所有元素,通常只需要给出用户关心的部分元素;

(2) 给每个元素赋予不同的权重,真实反映了用户对每个元素关注程度不一样.实际应用中,由用户指定查询图和每个元素的权重.例 1 中,关于设计人员是否为设计专业的要求有时并不重要,只需要精通 PS 和 AI 设计,因此可以为设计专业元素指定一个较低权重.

1.3 方法概览

子图查询问题非常困难,因为子图同构验证问题是 NP 完全问题[28].而实际应用中的网络非常巨大,为保证在短时间内完成子图查询,需要设计良好的索引技术和查询优化技术来解决子图查询问题.

图 3 所示的是 SMID 查询算法的基本框架.SMID 查询算法包含离线索引构建过程和在线查询处理过程两部分.

- 离线索引构建是为数据图建立索引的过程:首先为数据图节点签名,该签名同时包含节点元素信息和数据图的结构信息;然后,将数据签名组织成动态签名树;
- 在线查询处理过程中,首先获得查询图节点签名,同样,此签名也包含了数据图节点元素和结构信息.为进一步加快查询效率,本文提出了支配子图查询算法,选择查询图的最小支配子图;然后,利用 DS-Tree 找出每个支配节点的候选节点集,再利用距离保留定理找出非支配节点的候选集;最后找出所有与查询图匹配的图.

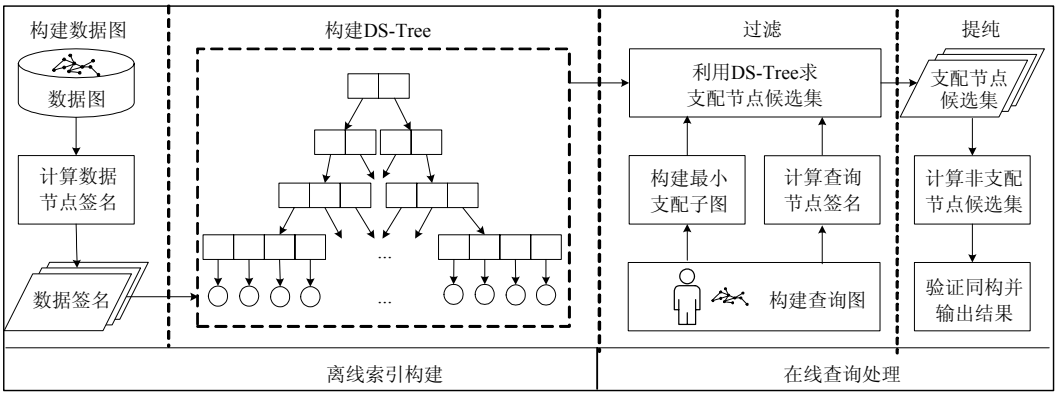


Fig.3 Framework of SMID

图 3 SMID 的基本框架

SMID 查询算法与 SMS^2 算法的不同点在于以下 3 个方面.首先,子图匹配的定义不同.SMS^2 要求对应节点元素的集合相似度大于给定阈值,而 SMID 要求它们的动态加权包含度满足一定条件;其次,索引结构不同.SMS^2 在离线处理阶段为数据图创建了倒排模式格(inverted pattern lattice)和签名桶(signature buckets)索引,而 SMID 提出了针对动态权重集合包含度特点的 DS-Tree 索引;最后,查询处理过程不同.SMS^2 采用了多种剪枝技术加快在线查询处理过程,而 SMID 使用了支配子图以加快查询.

2 离线索引构建

本节详细介绍离线索引构建过程.

- 首先,针对数据图和查询图分别设计一种签名机制:数据签名和查询签名.这两种签名既保留了数据(查询)节点元素信息,也保留了数据(查询)图的部分结构信息;
- 然后,将数据签名组织成 DS-Tree,作为数据图的索引;
- 最后,还针对索引空间较大的问题探讨了 DS-Tree 的空间压缩算法.

2.1 数据签名

为查询节点 u 和数据节点 v 定义了两种不同的签名,分别称为查询签名 $Sig(u)$(query signature)和数据签名 $Sig(v)$(data signature).利用这两种签名,我们能快速过滤掉一些不满足条件的数据节点.

定义 5(位向量). 数据图 G 中所有节点的元素并集为$\Sigma(G)$,$\Sigma(G)$中元素按指定顺序排序,对于某个节点 u,其包含的元素集 $S(u)$对应的位向量 $BV(u)$是一个长度为$|\Sigma(G)|$的二进制串,若元素 $e\in S(u)$,那么 e 在 $BV(u)$对应的位设为 1,否则为 0.

定义 6(查询签名). 给定查询图中的节点 u,设 u 有 m 个相邻节点 $u_i(i=1,\ldots,m)$,则查询节点 u 的签名 $Sig(u)=\{\langle BV(u),BV(u_1)\rangle,\ldots,\langle BV(u),BV(u_m)\rangle\}$,其中,$BV(u)$和 $BV(u_i)$分别是 $S(u)$和 $S(u_i)$元素集合对应的位向量.

例 1 中,若指定所有元素的顺序为〈.NET,3D MAX,AI,CSS,Html,Java,JS,MySQL,PHP,PS,云计算,数据挖掘,机器学习,架构设计,算法,设计专业,项目管理〉,$Sig(u_5)$={〈00011010 00000000 0,00011010 00000000 0〉,〈00011010

00000000 0, 00000001 10000000 0⟩}.

定义 7(数据签名). 给定数据图中的节点 v,设 v 有 n 个相邻节点 $v_i(i=1,\ldots,n)$,则数据节点 v 的签名:

$$Sig(v)=\langle BV(v), \vee_{i=1}^{n} BV(v_i)\rangle,$$

其中,$BV(v)$是 $S(v)$的位向量,$\vee$是按位或操作,$\vee_{i=1}^{n} BV(v_i)$称为合并位向量(union bit vector),记为 $BV_{\cup}(v)$,其值等于 v 所有相邻节点的 $S(v_i)$的位向量按位或.

由于数据节点非常多,数据签名中采用相邻节点的合并位向量既能保留部分图结构信息,又不会让图索引太大.例 1 中,$Sig(v_5)$={⟨00011010 01000000 0,10011011 10000000 0⟩}.而查询图通常不会太大,因此查询签名保留所有的邻居节点信息,方便后续的过滤.

文献[2]提出的 gStore 也为查询图和数据图设计了签名以加快 SPARQL 查询过程.与之不同的是:

- 首先,本文处理的是边无标签的图查询,在编码时不考虑边的标签,而 gStore 需要考虑边的标签;
- 其次,在哈希函数的选择上,本文使用了位向量,gStore 针对边标签和节点值选用了不同的哈希函数;
- 最后,gStore 对数据签名和查询签名均采用了类似合并位向量的策略,而本文为提升过滤效果,查询签名中保留了所有的邻居节点信息.

定义 8(签名面积). 给定一个签名 sig,定义其面积为 sig 中 1 的个数,记为 $Area(sig)$.为简单起见,节点 v 的签名面积记为 $Area(v)$.

定义 9(签名增量面积). 签名 sig_j 相对于签名 sig_i 的增量面积由公式(3)给出:

$$\Delta Area(sig_i,sig_j)=Area((\neg sig_i)\wedge sig_j) \tag{3}$$

先将 sig_i 按位取反,然后与 sig_j 对应位做与运算,得到的结果签名面积.其含义为:sig_j 对应的集合加入到 sig_i 对应的集合中产生的新集合的签名相对于 sig_i 的增长面积.注意,签名增量不满足交换律,即,一般地:

$$\Delta Area(sig_i,sig_j)\neq\Delta Area(sig_j,sig_i).$$

2.2 构建DS-Tree

为更好地利用签名信息,我们将数据签名组织成动态签名树(dynamic signature tree,简称 DS-Tree).本节介绍如何构建数据图的 DS-Tree;下一节我们将会看到,DS-Tree 能够很好地帮助我们完成 SMID.

DS-Tree 是 S-Tree[29−31]的变形.S-Tree 是一种形如 B+树的动态平衡树.S-Tree 中,每个节点包含若干个形如⟨sig,ptr⟩的条目.在叶子节点中,sig 是一个数据节点的签名,ptr 是指向该数据节点的指针.对于非叶子节点,ptr 指向该条目对应的子树根节点,sig 是对应子树根节点所有条目签名的按位或称为该子树根节点的节点签名.节点 $node$ 内可允许的最大条目数称为该节点的容量,记为 $Cap(node)$.图 4 是例 1 对应的 S-Tree(灰色背景是树节点签名,白色背景是条目签名),对于所有的 $node$,$Cap(node)$=3.

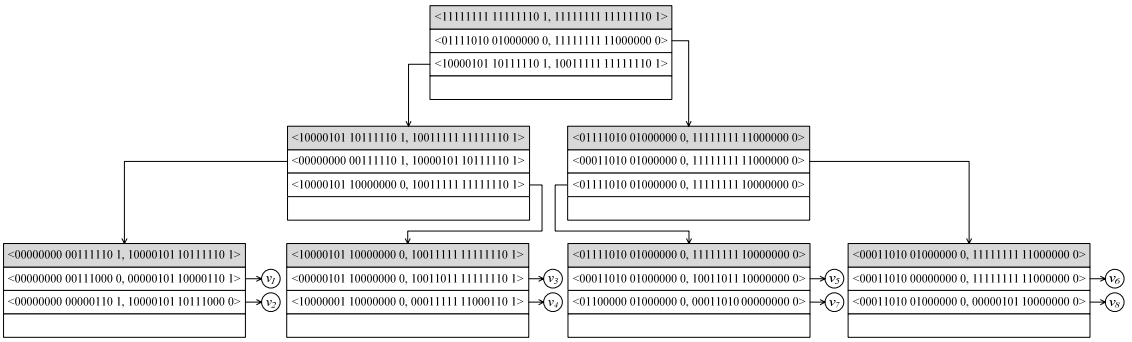


Fig.4 An example of S-tree

图 4 S-tree 示例

图 4 中,灰色背景的签名表示树节点签名,在实现时不会存储,可通过父节点的条目获得.

在S-Tree中,每层的节点容量都相同,这样会导致一个问题:在构造S-Tree的过程中,越底层的节点越容易装满,而高层节点的条目一般比较少.若统一将容量设为较大的值,一个树节点内将包含很多条目,这会导致S-Tree的过滤性能低下.若将容量设为较小的值,在构建过程中,树节点不断地分裂,导致构建过程非常慢.这是因为上一层节点的一个条目对应了下一层单个节点的所有条目.因此,本文提出一种基于S-Tree的动态DS-Tree索引.DS-Tree采用动态容量策略.容量的选定与DS-Tree节点的层数、数据节点数目和数据节点元素的总数有关.直观上看,DS-Tree节点所在的层数越高,其容量应该越小;数据节点数目越多或者数据节点元素总数$|\Sigma(G)|$越少,数据节点分配到同一个DS-Tree节点的数目就越多,DS-Tree节点容量应该越大.因此,$Cap(node)$的计算可采用公式(4):

$$Cap(node)=\max\left\{3,\left\lfloor s'\times\frac{|V(G)|/|\Sigma(G)|}{e^{r'\times l}}\right\rfloor\right\} \tag{4}$$

其中,$|V(G)|$,$|\Sigma(G)|$分别表示数据图G节点和元素的个数,l表示DS-Tree节点所在的层数,e为自然对数的底数.我们取叶子节点所在层为第0层.r',s'是正实数,r'表示容量随层数的增加而递减的速度,s'决定每一层容量的大小.取3为下限是防止无限分裂.给定一个数据集,其节点数目和元素总数固定,因此,树节点$node$的容量仅取决于s',r'和l.故,公式(4)可以简写成公式(5):

$$Cap(node)=\max\{3,\lfloor s\times e^{-r\times l}\rfloor\} \tag{5}$$

其中,s,r为正实数.

为DS-Tree的不同层节点设置不同的容量,一方面控制了不同层树节点的分裂时机,加快了树的构建过程;另一方面,也能加快查询过程.高层(靠近根节点)的DS-Tree节点具有更粗粒度的过滤效果,而低层(靠近叶子节点)具有更细的过滤效果.若高层节点的容量较大,节点签名面积将更大,即签名中1的数目将会更多,过滤效果往往不好;若低层叶子节点容量设置较小,会导致DS-Tree的高度过大,减慢了查询过程.

算法1是DS-Tree的构建算法,主要包括两个步骤——*InsertDSTree*和*Split*.

- *InsertDSTree*方法为条目e自顶向下选择合适的路径插入到DS-Tree的一个叶节点中;
- *Split*方法自底向上分裂那些超过最大容量的DS-Tree节点.

稍后我们将看到,DS-Tree过滤效果的好坏与相似数据节点的聚集程度相关.因此,*InsertDSTree*与*Split*有一个共同目标:对于同层的DS-Tree节点,节点内的数据节点尽可能相似,节点间的数据节点尽可能不同.这取决于算法1中的第11步和第20步.

算法1. DS-Tree Construction.

输入:数据图G;

输出:DS-Tree.

方法:

```
/*DS-Tree 构建的主方法*/
(1)  function DSTreeBuild (DataGraph G) {
(2)      DSNode root;                          //DS-Tree 的根节点
(3)      foreach data vertices v in G {
(4)          Entry e=⟨Sig(v),v.ID⟩;            //获得对应的条目
(5)          InsertDSTree(root,e);             //将条目插入 DS-Tree 中
(6)      }
(7)      return root;
(8)  }
/*将 e 插入到 DS-Tree 中*/
(9)  function InsertDSTree (DSNode node,Entry e) {
(10)     while (node is not a leaf) {
```

```
(11)            node=ChooseSubtree(node,e);     //选择某个子树
(12)        }
(13)        Insert e into node;
(14)        if (node overflows) {                //若超过了最大容量,则分裂
(15)            Split(node);
(16)        }
(17)    }
/*DS-Tree 节点分裂*/
(18)    function Split (DSNode node) {
(19)        while (node overflows) {
(20)            Split node into two nodes;       //将 node 分裂
(21)            node=node.parent;                //向上传递分裂
(22)        }
(23)    }
```

第 11 步的 *ChooseSubtree* 方法选择使节点 *node* 内面积增量最小的条目对应的子树.条目增长的面积可以通过公式(3)算出,将 *node* 中每个条目的签名取反后与 *e* 的签名按位与运算.也就是说,对于 *node* 内的每一个条目 e_i,选出使得 $\Delta Area(e_i,e)$ 最小的条目.若存在多个这样的条目,为使 DS-Tree 更平衡,选择对应子树根节点条目最少的条目.

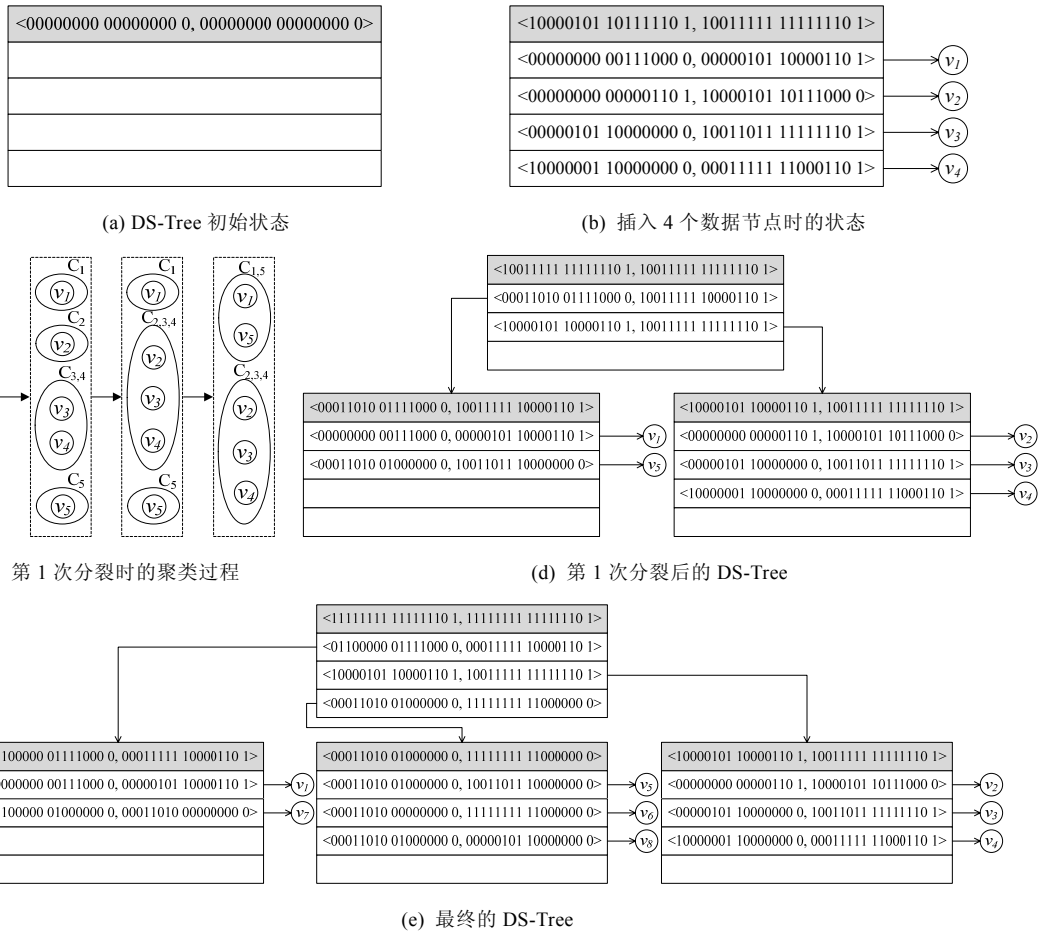
第 20 步将 DS-Tree 节点 *node* 分裂成两个节点.有多种分裂方法可供选择[29,30],根据文献[30]的实验结果,采用基于层次合并聚类(agglomerative hierarchical clustering,简称 AHC)算法,将 *node* 内的条目聚成两类,分别放入两个新节点中,并删除原来的节点 *node*.

具体实现为:最开始,将 *node* 内的每个条目分别视为一个簇,簇的签名为其内所有条目的签名按位或运算;然后,将两个 Jaccard 相似度最大的簇合并成一个簇.如此反复,直到剩下两个簇为止.为使分裂后 DS-Tree 平衡,当其中一个簇的条目数大于原 DS-Node 容量的一半时,直接将其他簇合并成一个簇;接着,将剩下的两个簇中的条目放入到两个 DS-Tree 新节点中,同时更改相应的父子关系,最后删除 *node*.由于分裂使父节点的条目数增加了,因此验证父节点是否溢出:若溢出,分裂父节点.如此递归下去.

图 1(a)的数据图中,$|V(G)|=8$,$|\Sigma(G)|=17$.假设公式(4)中 $s'=10$,$r'=0.45$,那么最底层的 DS-Tree 节点的容量为 4,第 1 层的 DS-Tree 节点的容量为 3.最开始,DS-Tree 只有一个空节点,如图 5(a)所示(灰色背景是 DS-Node 签名,白色背景是条目签名).逐一插入数据图节点到 DS-Tree 中,并更新相应的 DS-Tree 节点签名.前 4 个数据节点插入后的状态如图 5(b)所示.

继续插入数据节点 v_5,此时,该 DS-Tree 节点的条目数为 5,超过了最大容量,将其分裂成两个节点.分裂过程如图 5(c)所示:开始,每个条目单独形成一个簇,找到最相似的两个簇 C_3,C_4 合并成一个新簇 $C_{3,4}$;继续找最相似的两个簇 $C_{3,4}$,C_2 合并成 $C_{2,3,4}$,此时,$C_{2,3,4}$ 中的条目数大于原 DS-Tree 节点容量的一半,直接将其他的簇合并成一个簇;将最后两个簇的条目放入两个新的 DS-Tree 节点中,并为两个新的 DS-Tree 创建一个父节点,父节点的两个条目分别指向两个新的 DS-Tree 节点;最后,删除原来的 DS-Tree 节点,此时,DS-Tree 如图 5(d)所示.

当插入 v_6 时,自顶向下找到一条插入路径.首先找到使图 5(d)根节点中条目签名增量面积最小的条目,指向左子树的条目增量面积为 3,指向右子树的条目增量面积为 5,选择左子树节点插入 v_6,同时更改根节点和左子树节点的节点签名.同样办法插入 v_7,v_8.最后,DS-Tree 如图 5(e)所示.注意:除根节点外,每个 DS-Tree 节点的签名对应其父节点的一个条目,因此在实现中只保存根节点的签名.



(a) DS-Tree 初始状态

(b) 插入 4 个数据节点时的状态

(c) 第 1 次分裂时的聚类过程

(d) 第 1 次分裂后的 DS-Tree

(e) 最终的 DS-Tree

Fig.5 Construction of DS-tree

图 5 DS-Tree 的构建过程

2.3 DS-Tree压缩存储

许多应用中,数据签名都非常稀疏,即,单个数据节点包含的元素很少.也就是说,元素的种类很多,单个节点的签名面积非常小.因此,按位存储签名会浪费很多空间.为解决这个问题,可以采用一种压缩技术:只记录位向量中为 1 的位置.例如,数据图节点 v_5 的压缩签名为 $Sig(v_5)_{压缩}$={3,4,6,9,17,20,21,23,24,25}.

另一方面,可以利用 DS-Tree 的特点对 DS-Tree 进行压缩.文献[32]指出:根据 DS-Tree 的特点,若父节点中某一位为 0,其对应子树内所有条目签名的该位均为 0,因此,只需记录父节点的 0,所有子节点条目签名对应位的 0 均可删除.此外,利用 DS-Tree 同一树节点内的签名相似度高的特点,首先将 DS-Tree 每个树节点内的条目重新排序,使得相邻条目签名更接近,这样,相邻条目签名异或操作将会得到一个稀疏签名;然后,利用上述稀疏签名的压缩方法对 DS-Tree 进行压缩.

由于在查询过程中,上述提到的方法均需要解压缩,因此降低了查询效率.本文提出了一种新的压缩策略.同一个数据集中不同元素的出现次数一般不同,甚至相差很大.基于上述观察,区别对待不同频率的元素,只对低频元素进行压缩,如图 6 所示.首先统计数据集中每个元素的出现次数.然后按出现次数对元素进行降序排列,对应的节点签名按降序进行编码.然后,将元素及其对应的编码分别分成高频低频两部分,如图 6(a)所示.对于低频部分,进行折叠压缩.设低频部分的编码下标为 0,1,…,n−1,将第 0 编码与第 n−1 编码按位或,得到的结果放入

压缩结果的第 0 位,将第 1 位编码与第 n–2 位编码按位或的结果放入压缩结果的第 1 位,以此类推,直到每个低频编码都参与了压缩运算(n 为偶数),或者只剩下一个低频编码(n 为奇数),直接将剩下的编码加入压缩结果.最后将压缩结果替换低频部分的编码. 注意数据节点 v 自身签名 $BV(v)$及其邻居签名 $BV_{\cup}(v)$分别执行上述压缩操作,最终得到图 6(b)的压缩结果.查询时,查询节点编码也采用相同的压缩方法,查询策略不变.压缩后,DS-Tree 的过滤效果会下降,因此得到的候选节点会增多.

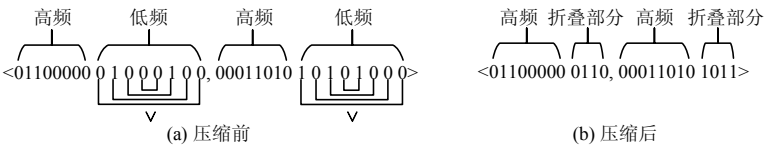


Fig.6 An example of compressing DS-tree

图 6 DS-Tree 压缩示例

因为低频元素对应的编码比较稀疏,因此我们只对这些编码进行压缩,压缩后对过滤效果影响不大.折叠压缩的原因是尽量让折叠后的编码比较均匀.由于在查询时无需进行解压操作,因此在查询效率影响不大的情况下,减少了 DS-Tree 占用的空间(见第 4 节).

3 查询处理

3.1 利用DS-Tree查询

根据 SMID 查询的定义,给定数据节点 v 和查询节点 u,若 u 与 v 的加权包含度小于 τ,那么 v 节点可以被过滤掉.与加权包含度对应,我们定义位向量之间的加权包含度.

定义 10(位向量加权包含度). 给定查询图节点 u 和数据图节点 v,位向量 $BV(u)$和 $BV(v)$的加权包含度为

$$VID(BV(v)/BV(u))=\frac{\sum_{a\in BV(v)\wedge BV(u)}W(a)}{\sum_{a\in BV(u)}W(a)} \tag{6}$$

其中,∧是按位与操作符,$a\in BV(u)\wedge BV(v)$表示位向量中对应的位为 1;$W(a)$是元素 a 对应的权重.

SMID 查询转化为:对于查询图每个节点的位向量 $BV(u_i)$,需要在数据图中找到一个节点的位向量 $BV(v_j)$,使得 $VID(BV(v_j)/BV(u_i))\geqslant\tau$.这里定义合并包含度上限.

定义 11(合并位向量加权包含度上限). 给定位向量 $BV(u_i)$和 $BV_{\cup}(v)$,其中,u_i 是目标查询图节点 u 的某个邻居节点,则它们的合并加权包含度上限为

$$UID(BV_{\cup}(v)/BV(u_i))=\frac{\sum_{a\in BV_{\cup}(v)\wedge BV(u_i)}W(a)}{\sum_{a\in BV(u_i)}W(a)} \tag{7}$$

基于定义 10 和定义 11,我们有如下聚集定理:

定理 1(聚合定理). 给定查询节点 u 和数据节点 v,u_i 是 u 的某个邻居节点.如果 $UID(BV_{\cup}(v)/BV(u_i))<\tau$,那么对于 v 的所有直接邻居 v_j,均有 $VID(BV(v_j)/BV(u_i))<\tau$.

证明: $VID(BV(v_j)/BV(u_i))=\dfrac{\sum_{a\in BV(v_j)\wedge BV(u_i)}W(a)}{\sum_{a\in BV(u_i)}W(a)}\leqslant\dfrac{\sum_{a\in BV_{\cup}(v)\wedge BV(u_i)}W(a)}{\sum_{a\in BV(u_i)}W(a)}=UID(BV_{\cup}(v)/BV(u_i))<\tau$.

基于聚合定理,有如下推论.

推论 1. 给定查询签名 $Sig(u)$和数据签名 $Sig(v)$,如果 $VID(BV(v)/BV(u))<\tau$,或者存在一个 u 的邻居节点 u_i,使得 $UID(BV_{\cup}(v)/BV(u_i))<\tau$,那么 v 不会与 u 匹配.

推论 2. 给定查询签名 $Sig(u)$和 DS-Tree 节点 $node$ 的签名 $Sig(node)$,如果 $VID(BV(node)/BV(u))<\tau$,或者存在 u 的一个邻居节点 u_i,使得 $UID(BV_{\cup}(node)/BV(u_i))<\tau$,那么 v 不会与以 $node$ 为根节点的子树内所有的数据节点

匹配.

因此,利用 DS-Tree 过滤方法如下:给定查询图 Q 和包含度阈值 τ,对于每一个 $u_i \in V(Q)$,从 DS-Tree 的根节点开始深度遍历(depth first search,简称 DFS).若 DS-Tree 的节点 *node* 不满足推论 2,则以 *node* 为根节点的子树无需继续遍历.逐一验证所有满足推论 2 的 DS-Tree 叶子节点对应的数据节点,那些满足包含度上限的数据节点构成的集合即为 u_i 的候选节点集,记为 $C(u_i)$.最后,通过算法 2 验证 u_i 对应的每个候选节点能否构成 Q 的同构图.

基于文献[13],本文提出了一种启发式子图同构验证算法(算法 2).其他的子图同构算法(如文献[12])也可直接应用于 SMID 查询.算法 2 首先将查询节点按照候选集合元素个数从小到大排序,这样能够减少递归调用次数,然后调用 *DFSCheckMatch* 函数.*DFSCheckMatch* 函数用哈希表记录所有已经匹配的结果,开始时循环遍历当前查询节点的所有候选节点,若当前候选节点未匹配其他查询节点,并且对应的边也匹配(第 10 行),那么将此候选节点放入结果集中(第 11 行).若所有的查询节点都找到了匹配节点,则输出结果,否则继续递归调用 *DFSCheckMatch* 函数.最后删除当前候选节点(第 17 行),以便验证当前查询节点的其他候选节点,找出所有匹配的子图.

算法 2. Check Isomorphic.

输入:数据图 G、查询图 Q、Q 的每个节点 u_i 的候选集 $C(u_i)$;

输出:Q 的所有同构子图.

方法:

```
(1)  function CheckIsomorphic(DataGraph G,QueryGraph Q,CandidatesMap C) {
(2)      Map matchedNode=∅;                                  //已经用作匹配的数据节点集
(3)      Sort C by candidates' number in ascending order;    //将 C 按候选节点数升序排序
(4)      DFSCheckMatch (G,Q,C.begin(),matchedNode);
(5)  }
(6)  function DFSCheckMatch(DataGraph G,QueryGraph Q,
         CandidatesMapIterator it,Map matchedNode) {
(7)      QueryNode node=it→key;
(8)      CandidatesSet candidates=it→value;
(9)      foreach candidate in candidates {
(10)         if (candidate not exist in matchedNode          //该候选点未被使用
             and all edges of node match) {                  //所有已匹配节点的边匹配
(11)             Add ⟨node,candidate⟩ to matchedNode;
(12)             if (matchedNode.size==Q.size) {
(13)                 output matchedNode as a matched graph;
(14)             } else {
(15)                 DFSCheckMatch (G,Q,it.next(),matchedNode);
(16)             }
(17)             Remove ⟨node,candidate⟩ from matchedNode;
(18)         }
(19)     }
(20) }
```

例如,利用图 5(e)的 DS-Tree 来查询图 1(b)中 u_4 节点的候选节点.若指定元素权重为:W("设计专业")=0.2,其他均为 0.9,τ=0.8.$Sig(u_4)$={⟨$BV(u_4)$,$BV(u_2)$⟩,⟨$BV(u_4)$,$BV(u_5)$⟩,⟨$BV(u_4)$,$BV(u_6)$⟩},分成 3 部分,验证每部分与 DS-Tree 节点的条目是否满足推论 2.首先与 DS-Tree 根节点签名比较,包含度为{⟨1,1⟩,⟨1,1⟩,⟨1,1⟩},满足条件,进一步验证根节点中每个条目.由于与第 1、第 2 个条目的包含度分别为{⟨0,1⟩,⟨0,1⟩,⟨0,0⟩}和{⟨0,1⟩,⟨0,1⟩,⟨0,1⟩},不满足推论

2,过滤其子树对应的所有数据节点;与第 3 个条目的包含度上限为{⟨1,1⟩,⟨1,1⟩,⟨1,1⟩},因此继续比较第 3 个条目对应的子树.用上述同样的方法验证该节点的每个条目.只有指向 v_6 节点的条目满足条件,因此 $C(u_4)=\{v_6\}$.注意,这里自动过滤了元素包含度大于 0.8 但结构不满足条件的 v_8,v_5.

3.2 支配子图

我们发现:对于查询节点 u_i,其候选节点集为 $C(u_i)$.设 u_j 为 u_i 的邻居节点,那么 u_j 的匹配节点一定包含在 $C(u_i)$ 的邻居节点集中.基于上述观察,我们提出基于支配子图(dominating subgraph)的 SMID 算法,当 DS-Tree 较大时,能减少遍历 DS-Tree 的次数,进一步提高查询效率.

支配子图的定义基于支配集[33−35],因此我们先看支配集的定义.

定义 12(支配集(dominating set)). 给定图 Q,称集合 $DS(Q)$是 $V(Q)$的支配集,当且仅当 $DS(Q)\subseteq V(Q)$,且 Q 的每个节点 u 要么 $u\in DS(Q)$,要么 u 与 $DS(Q)$的某些节点相邻.

定理 2(支配集定理). 给定节点 $u\in DS(Q)$,若$|DS(Q)|\geqslant 2$,则至少存在一个节点 $u'\in DS(Q)$,使得 $Hop(u,u')\leqslant 3$.其中,$Hop(\cdot,\cdot)$表示 Q 中两个节点之间最少边的数目.支配节点 u'称为 u 的相邻支配节点(neighboring dominating vertex).

证明:假定不存在 $u'\in DS(Q)$使得 $Hop(u,u')\leqslant 3$,那么 u 与任何其他支配节点 u'之间至少存在 3 个节点.此时至少有一个非支配节点与 u 或者 u'均不相邻,与定义 12 矛盾.

定义 13(一阶邻居和二阶邻居). 给定图 Q,$u\in V(Q)$,u 的一阶邻居集 $N_1(u)$和二阶邻居集 $N_2(u)$定义如下:

- $N_1(u)=\{u'|u'\in V(Q)$,且 u'和 u 之间至少存在一条长度为 1 的路径$\}$;
- $N_2(u)=\{u'|u'\in V(Q)$,且 u'和 u 之间至少存在一条长度为 2 的路径$\}$.

图 7 所示两个相邻支配节点 u_i 和 u_j 之间可能的拓扑结构.图 7(a)表示 u_i 和 u_j 直接相邻,图 7(b)表示 u_i 和 u_j 有共同的一阶邻居,图 7(c)表示 u_i 的二阶邻居中存在一个节点是 u_j 的一阶邻居,图 7(d)表示 u_i 的一阶邻居中存在一个节点是 u_j 的二阶邻居.实际上,图 7(c)和图 7(d)的情况等价.

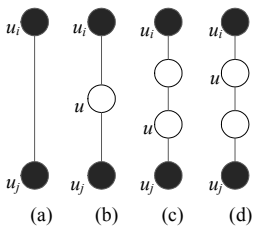


Fig.7 Possible topologies between two dominant vertexes

图 7 两个支配点之间可能的拓扑结构

基于支配集和支配集定理,我们定义支配子图:

定义 14(支配子图(dominating subgraph,简称 DS-graph)). 给定图 Q 的支配集 $DS(Q)$,图 Q 的支配子图 $Q^D=\langle V(Q^D),E(Q^D),\Theta\rangle$,其中,$V(Q^D)=DS(Q)$,$\Theta$是映射函数:$V(Q^D)\times V(Q^D)\to E(Q^D)$,边$(u_i,u_j)\in E(Q^D)$当且仅当下面至少一个条件满足:

(1) u_i 与 u_j 在 Q 中相邻(对应图 7(a));

(2) $|N_1(u_i)\cap N_1(u_j)|>0$(对应图 7(b));

(3) $|N_2(u_i)\cap N_1(u_j)|>0$(对应图 7(c));

(4) $|N_1(u_i)\cap N_2(u_j)|>0$(对应图 7(d)).

对于情形(1),边(u_i,u_j)的距离为 1(即 u_i~u_j 至少有一条长度为 1 的路径);对于情形(2),边(u_i,u_j)的距离为 2;情形(3)和情形(4),边(u_i,u_j)的距离均为 3.

将查询图 Q 转化成支配子图,首先找到 Q 的支配节点,然后对于 $DS(Q)$的每一对节点 u_i 和 u_j,确定它们之间

是否有边及距离.对于同一个查询图 Q,可能存在多个支配子图.图 8(a)~图 8(c)的灰色节点为例 1 查询图支配节点的 3 种情况.为减少遍历 DS-Tree 的次数、加快查询效率,我们选择最小支配子图.

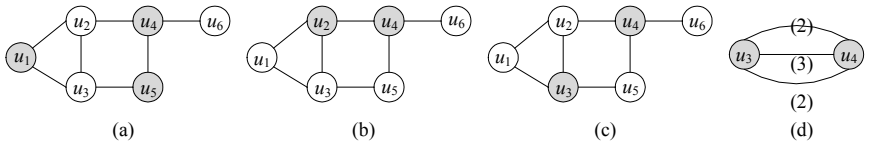


Fig.8 Dominant vertexes of example 1

图 8 例 1 查询图的支配节点

定义 15(**最小支配子图**(**minimum dominating subgraph**,**简称 Min DS-graph**)). 图 Q 的所有支配子图中,节点数目$|V(Q^D)|$最少的支配子图称为 Q 的最小支配子图,记为 $Q^D_{\min}$.

对于同一个查询图 Q,可能存在多个最小支配子图.例如,图 8(b)和图 8(c)对应的支配子图都是例 1 查询图的最小支配子图.文献[33]证明:对于含有 n 个节点的任意图 Q,其最小支配子图数目的下限和上限分别是 1.5704^n 和 1.7159^n.而找最小支配子图是一个 NP 难问题,当前已知最快的精确查找最小支配子图的算法时间复杂度为 $O(1.4957^n)$[35].综合考虑,本文提出一种贪心算法找出近似最小支配子图.

算法开始时,初始化集合 $DS=\varnothing$,不断加入节点到 DS 中,直到 DS 集合是支配集.为便于描述,我们将 Q 中的节点分成两类:一类是 DS 中的节点及其相邻节点,称为标记节点,记为集合 M;另一类称为未标记节点,与未标记节点 u 相邻的所有未标记节点的数目记为 $D(u)$.直观上看,为使最终$|DS|$最小,每次加入到 DS 中的未标记节点 u 的 $D(u)$应该最大.基于上述观察,贪心查找近似最小支配子图见算法 3.

算法 3. Find Min DS-graph.

输入:查询图 Q;

输出:近似最小支配子图 $Q^D_{\min}$ *.

方法:

```
(1)  function FindMinDSGraph(QueryGraph Q) {
(2)      Set DS=∅;
(3)      while (∃unmarked nodes){
(4)          Choose u∈{x|D(x)=max_{v∈(V(Q)−M)}{D(v)}};
(5)          DS=DS∪{u};
(6)          M=M∪{u and u's neighbors};
(7)      }
(8)      Find E( Q^D_min *) based on DS;
(9)      return  Q^D_min *;
(10) }
```

按照算法 3 的近似支配子图 $Q^D_{\min}$ *,任何两个支配节点都不可能相邻,因为每次都会标记当前支配节点 u 及其邻居(第 6 行).例 1 查找图的近似最小支配子图过程是:最开始,$DS=\varnothing$,$M=\varnothing$,$V(\bar{Q})M=\{u_1,u_2,u_3,u_4,u_5,u_6\}$.其中,$D(u_3)=D(u_4)=3$ 最大,我们取 u_3 放入 DS,并将 u_3 和 u_3 的邻居节点放入 M.此时,$DS=\{u_3\}$,$M=\{u_1,u_2,u_3,u_5\}$.$V(\bar{Q})M=\{u_4,u_6\}$中,$D(u_4)=D(u_6)=1$,我们取 u_4 放入 DS,u_4 和 u_6 放入 M.至此,$V(\bar{Q})M=\varnothing$.例 1 查询图的支配节点集 $DS=\{u_3,u_4\}$,对应支配子图为图 8(d),其中,边上的数字为 u_3,u_4 的距离.可见,支配子图可以有多条边.

定理 3(**距离保留定理**(**distance preservation principle**,**简称 DP-Principle**)). 假设 X 是 Q 在 G 中的一个子图匹配,X^D 是与 Q^D 对应的支配子图匹配.Q^D 和 X^D 分别有 n 个节点 $u_1,\ldots,u_n$ 和 $v_1,\ldots,v_n$.对于 Q^D 中任意边(u_i,u_j),均有:

(1) 如果距离是 1,那么 v_i 与 v_j 相邻;

(2) 如果距离是 2,那么$|N_1(v_i)\cap N_1(v_j)|>0$;

(3) 如果距离是 3,那么$|N_1(v_i)\cap N_2(v_j)|>0$ 或者$|N_2(v_i)\cap N_1(v_j)|>0$.

根据 SMID 和支配子图的定义,容易证明定理 3 的正确性.因此,对于那些非支配节点,我们可以通过支配节点的候选集找到非支配节点的候选集.例 1 中查询图利用 DS-Tree 找到支配节点 u_3,u_4 的候选集,$C(u_3)=\{v_3,v_4\}$,$C(u_4)=\{v_6\}$.查询图的非支配节点 u_5 的候选节点集 $C(u_5)\subseteq N_1(v_6)\cap(N_1(v_3)\cup N_1(v_4))=\{v_3,v_4,v_5\}$,因此,$C(u_5)=\{v_5\}$.同样办法确定其他非支配节点的候选集 $C(u_6)=\{v_7\}$,$C(u_2)=\{v_3,v_4\}$,$C(u_1)=\{v_2\}$.

3.3 SMID算法

首先,调用算法 1 建立数据图 G 的 DS-Tree,这一步是离线过程,只需执行一次,把结果存入数据库或文件,在查询时只需加载即可.然后,调用算法 3 建立查询图 Q 的近似最小支配子图 $Q^D_{\min}$ *.对于每个支配节点 u,利用 DS-tree 找到其候选集 $C(u)$,对于非支配节点 u',利用距离保留定理找到其候选集 $C(u')$.最后,调用算法 2 检查子图同构并输出结果.

算法 4. SMID.

输入:查询图 Q,数据图 G,包含度阈值τ,各元素权重集 W;

输出:Q 的所有同构子图.

方法:

(1) **Call** Algorithm 1 to build G's DS-Tree;

(2) **Call** Algorithm 3 to build $Q^D_{\min}$ *;

(3) **foreach** $u\in V(Q^D_{\min}$ *){

(4) Using DS-Tree to find $C(u)$;

(5) }

(6) **foreach** $u'\in V(Q)-V(Q^D_{\min}$ *){

(7) Using DP-Principle to find $C(u')$;

(8) }

(9) **Call** Algorithm 2 to check isomorphic and output result.

4 实 验

首先介绍本文实验用到的数据集和实验环境,然后介绍对比方法的基本思想,最后对本文提出的 SMID 方法与对比方法的离线性能和在线性能进行分析比较.

4.1 数据集和实验环境

本文采用表 1 描述的数据集验证基于 DS-Tree 索引的 SMID 算法性能,其中包括 Freebase 和 DBpedia 两个真实数据集以及 3 个人工数据集.实验代码采用 C++语言编写,运行在主频 2.40GHz、内存 16G 的 64 位 Windows 8.1 操作系统上.

Table 1 Details of datasets

表 1 数据集详细信息

数据集	FB	DBP	S1M	S5M	S10M
节点数	158 762	200 001	1 000 000	5 000 000	10 000 000
边数	2 764 450	411 408	2 500 000	6 375 186	12 729 739
不同元素数	243	200	100	100	100
单个节点平均元素数	6	3.8	4.6	4.6	4.6

(1) Freebase(http://www.freebase.com/)是一个知识数据库,其中包含很多实体(演员、电影等).可以将实体

看成图节点,每个实体拥有的特征描述构成节点的元素集,实体之间的关联关系看成边.因此,Freebase可以构成一个无向图,记为 FB.实验时,元素权重随机指定,并标准化在[0,1]范围之间;

(2) DBpedia(http://wiki.dbpedia.org/)抽取了维基百科(https://www.wikipedia.org/)的结构信息.在 DBpedia数据集(记为 DBP)中,每个节点对应一篇维基百科的文章,采用特征抽取方法[36]从文章中选取 TF/IDF最高的 200 个单词作为该节点的元素,元素权重即为对应的 TF/IDF,并标准化在[0,1]之间;

(3) 文献[37]的图产生器能够产生节点度满足幂律分布的连接无向图.实验中使用 3 个不同大小的图数据集 S1M,S5M 和 S10M,其节点数目分别为 1 000 000,5 000 000,10 000 000.每个节点随机赋予 2 个~20个元素,每个元素的权重随机赋值,并标准化在[0,1]之间.

4.2 对比方法

近年来,Liang 等人提出的 SMS^2 查询问题[27]考虑了节点元素集合.与 SMID 不同的是,SMS^2 要求图结构同构,且对应节点元素的集合相似度大于给定阈值.在离线处理过程中,SMS^2 为数据图创建了倒排模式格和签名桶索引.在线处理阶段,首先利用横向剪枝、纵向剪枝、反单调性剪枝等多种剪枝技术找出查询图每个节点的候选节点集,然后进行子图同构验证.本文以 SMS^2 算法作为其中一个对比方法,同时为公平起见,修改 SMS^2 算法的相似度为加权包含度.

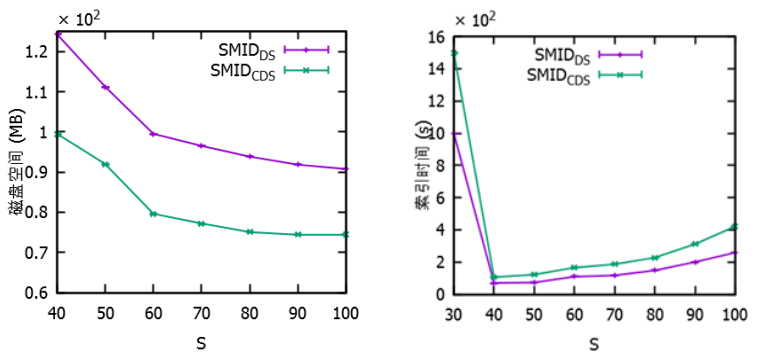
DS-Tree 是 S-Tree 的变形,不同的是,DS-Tree 的节点容量随着层数的增加而减少,而 S-Tree 中每层节点容量相同.为证明 DS-Tree 的有效性,我们比较基于 S-Tree 索引的 SMID 查询性能,记为 $SMID_S$.为便于区分,基于 DS-Tree 索引的 SMID 查询记为 $SMID_{DS}$.

为验证本文提出的 DS-Tree 压缩算法的有效性,我们比较 DS-Tree 压缩前后的索引大小及查询时间.基于压缩算法的 SMID 查询记为 $SMID_{CDS}$.$SMID_{CDS}$ 将数据图中的元素按频次倒序排序后,取前 50%作为高频,后 50%作为低频,然后对低频元素对应的签名进行折叠压缩.在实际应用中,可以根据实际情况调整高频和低频的范围,高频所占比例越少,压缩率越高.

同时,为验证支配子图查询算法能够加快查询效率,我们还比较基于 DS-Tree 索引并使用支配子图的查询方法,记为 $SMID_{DS}D$.

4.3 离线处理性能

离线处理主要开销为 DS-Tree 索引的建立,而 DS-Tree 索引大小和索引时间与公式(5)中的 s,r 参数有关.图 9 所示在 S1M 数据集上,使用压缩的方法 $SMID_{CDS}$ 与不使用压缩的方法 $SMID_{DS}$ 的索引大小和索引时间随 s 的变化情况.



(a) r=ln10,索引空间随 s 变化 (b) r=ln10,索引时间随 s 变化

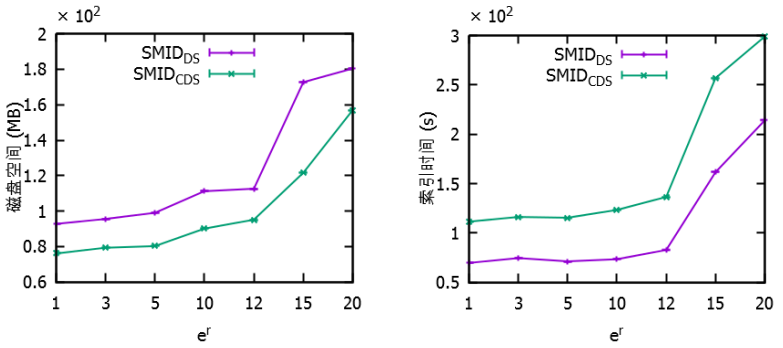
Fig.9 Impact of s on offline performance in dataset S1M

图 9 参数 s 对 S1M 数据集离线性能的影响

由图 9 可知:随着 s 的增长,$SMID_{CDS}$ 与 $SMID_{DS}$ 索引文件均变小,但索引时间均变大.因为当 s 增大时,表明每一层的相对容量变大,DS-Tree 的层数就越少,产生的 DS-Tree 就越小.但由于节点分裂采用原生聚类算法,节

点容量越大时聚类时间越长,因此索引时间也在增长.整体而言,$SMID_{CDS}$ 的索引空间约为 $SMID_{DS}$ 索引空间的76%~80%,因为压缩的 DS-Tree 对应的节点签名长度是未压缩的 75%,因此整体所占用的空间更小.$SMID_{CDS}$ 在索引建立的过程中需要进行折叠压缩操作,因此索引所需要的时间更长.一个有趣的现象是:当 s 过小(等于 30),索引文件大小和索引时间急剧上升,因为此时 DS-Tree 节点不断溢出和分裂.当 s=30 时,$SMID_{DS}$ 索引时间为1 000s,索引文件达到 2G(为了使图更清楚,当 s=30 时的索引空间情况未在图中体现).因此,s 不能太小.

图 10 所示 S1M 数据集的索引大小和索引时间随 r 的变化情况.由图可知:随着 r 的增长,$SMID_{DS}$ 与 $SMID_{CSD}$ 索引空间和索引时间都呈上升趋势.因为 r 决定每一层容量的递减速度,r 越大,高层的节点容量就越小,产生的 DS-Tree 索引越大.当 r 大于一定值时,索引空间和索引时间上升得更快,因为此时高层节点不断分裂.同样,$SMID_{CDS}$ 索引空间约为 $SMID_{DS}$ 的 80%左右,因为对应的签名长度为 75%.由于压缩需要时间,因此 $SMID_{CDS}$ 需要更多的时间.



(a) s=50,索引空间随 r 变化 (b) s=50,索引时间随 r 变化

Fig.10 Impact of r on offline performance in dataset S1M

图 10 参数 r 对 S1M 数据集离线性能的影响

根据多组实验,我们对各数据集的 r,s 选择见表 2.图 11 显示了各数据集的索引空间和索引时间情况.

Table 2 Default value of r/s in each dataset

表 2 各数据集的默认参数设置

数据集	FB	DBP	S1M	S5M	S10M
s	20	20	50	50	50
e^r	3	3	10	10	10

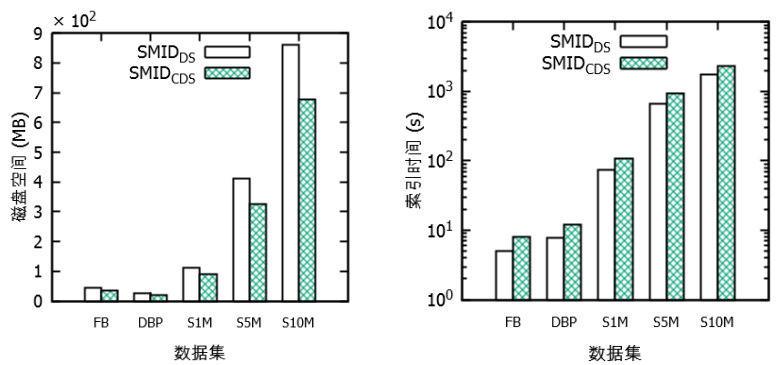


Fig.11 Index space and index time of different datasets

图 11 不同数据集的索引空间和索引时间

由图可知,索引空间和索引时间随着数据集的大小呈指数级增长.但即使对于 1000 000 000 节点的数据集,

$SMID_{DS}$ 方法的索引空间约为 800M,索引时间不到 30 分钟,单机性能仍可接受.对于每个数据集,基于以上同样的原因,$SMID_{CDS}$ 方法的索引空间要小,而索引时间稍高.

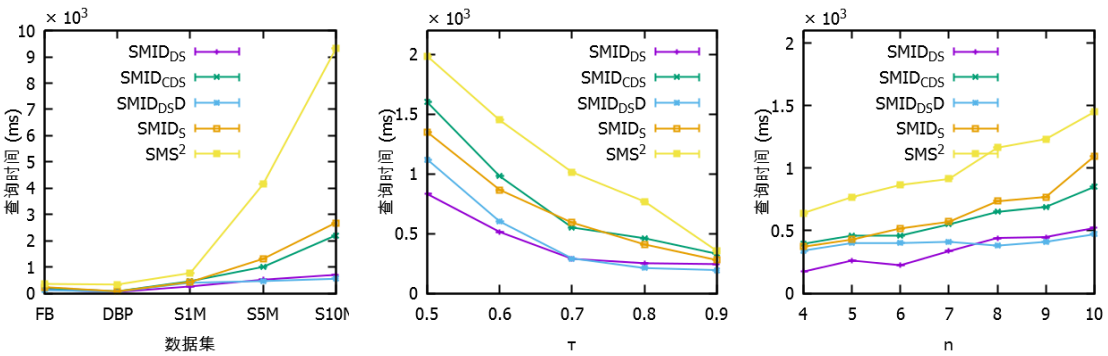
4.4 在线处理性能

在线处理阶段,我们比较 $SMID_{DS}$,$SMID_{S}$,$SMID_{CDS}$ 和 SMS^2 的查询响应时间.对于每组实验,从数据图中随机抽取 100 个特定大小的子图作为查询图.

图 12(a)表示包含度上限 τ=0.8、查询图大小 n=5 时,不同数据集上的平均查询时间.当数据量较小时,$SMID_{CDS}$,$SMID_{DS}$,$SMID_{S}$ 和 SMS^2 算法性能相差不大,但 $SMID_{CDS}$ 和 $SMID_{DS}$ 算法最佳,$SMID_{S}$ 次之,SMS^2 较差.随着数据量增大,$SMID_{DS}$ 优势愈发明显,表明 DS-Tree 索引的扩展性较好.$SMID_{CDS}$ 比 $SMID_{DS}$ 所需查询时间稍长,因为 $SMID_{CDS}$ 过滤的数据节点较少,产生的候选集较多,在验证同构时所需的时间更长.在小数据集上,$SMID_{DS}D$ 查询时间比 $SMID_{DS}$ 稍长,当数据集较大时,$SMID_{DS}D$ 算法略优.因为在小数据集上,验证子图同构的时间占主导地位.而大数据集对应的 DS-Tree 大,查询 DS-Tree 所需的时间更长.

图 12(b)显示的是在 S1M 数据集上,当 n=5 时,不同方法随着包含度阈值 τ的变化情况.τ越小,每个节点的候选节点就越多,验证子图同构所需的时间就越多,因此查询时间越久.随着 τ的增大,过滤的节点数就越多,因此总查询时间越少.对于 $SMID_{DS}D$ 方法,当 $\tau \geqslant 0.8$ 时,所需时间比 $SMID_{DS}$ 方法少,因为当包含度阈值越大,查到的候选节点少,由支配节点扩展找到非支配节点的候选节点也越少.此时,查找 DS-Tree 的时间占主要部分.随着包含度阈值增大,$SMID_{DS}D$ 查找时间少于 $SMID_{DS}$ 时间.

图 12(c)比较了不同方法的查询时间随查询图大小的变化情况,此时选择数据集为 S1M,τ=0.8.对于各方法,容易理解:当查询图越小时,验证子图同构的时间越少,查询就越快.可以发现一个有趣的现象,$SMID_{DS}D$ 方法随着查询图增大所需的查询时间变化不大.因为查询图变大时,支配子图的大小变化不大.因此,$SMID_{DS}D$ 适合查询图较大的情况.



(a) 查询时间 VS.数据集 (b) 查询时间 VS.包含度阈值 (c) 查询时间 VS.查询图大小

Fig.12 Query time of different methods

图 12 不同方法的查询时间

上述实验结果表明:在大数据图、高包含度阈值、查询图节点较多的情况下,基于支撑节点的 $SMID_{DS}D$ 方法要优于 $SMID_{DS}$ 方法.此外,本文提出的 DS-Tree 压缩方法能够在对查询时间影响不大的情况下,缩小 DS-Tree 的内存空间,这对数据图较大的情况非常有用.最后,$SMID_{DS}$ 优于基于 S-Tree 索引的查询方法 $SMID_{S}$.原因是:对于 DS-Tree,每层节点的容量是不同的,越往高层的容量越小.这符合 DS-Tree 的增长规律,即,越往高层的节点越难装满.同时,$SMID_{DS}$ 方法优于 SMS^2 方法,因为 DS-Tree 相对于 SMS^2 中的签名桶具有以下优势.

(1) 更好的适应性.签名桶基于局部敏感哈希(locality sensitive hashing,简称 LSH)[38],但 LSH 与特定应用相关,找到一个合适的哈希函数并不容易.而 DS-Tree 无需特定其他函数,能够适用不同应用的需求;

(2) 更大的灵活性.签名桶的层数需要事先指定,而层数的多少取决于数据图的大小,在创建之前难以预

估.而 DS-Tree 动态增长,无需实现指定层数;

(3) 更好的扩展性.同一个数据节点可能存在于多个签名桶中,这会造成存储空间浪费.当数据量大的时候,造成索引文件非常大.而对于 DS-Tree,一个数据节点只会存在于一个 DS-Tree 的叶节点中.

5 结 论

本文提出了一种应用广泛的子图查询问题,即基于包含度的子图匹配 SMID.在 SMID 查询中,图结构必须同构,且对应节点的加权包含度大于用户给定阈值.此外,给出了一种基于 DS-Tree 和最小支配子图的 SMID 查询算法.最小支配子图算法在数据图较大、包含度阈值较高和查询图较大的情况下,能够加快查询效率.针对 DS-Tree 占用内存空间高的问题,提出了一种 DS-Tree 的压缩算法,在对查询效率影响不大的情况下,缩小了内存空间.通过实验证明,本文给出的 SMID 算法在单机上具有较高的查询效率和较好的扩展性.下一步工作,将集中精力在计算机集群中实现 SMID 查询.

References:

[1] Yu X, Sun Y, Zhao P, Han J. Query-Driven discovery of semantically similar substructures in heterogeneous networks. In: Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2012. 1500−1503. [doi: 10.1145/2339530.2339765]

[2] Zou L, Mo J, Chen L, Özsu MT, Zhao D. gStore: Answering SPARQL queries via subgraph matching. Proc. of the VLDB Endowment, 2011,4(8):482−493. [doi: 10.14778/2002974.2002976]

[3] Tian Y, Patel JM. TALE: A tool for approximate large graph matching. In: Proc. of the 2008 IEEE 24th Int'l Conf. on Data Engineering. IEEE Computer Society, 2008. 963−972. [doi: 10.1109/ICDE.2008.4497505]

[4] Zhao P, Han J. On graph query optimization in large networks. Proc. of the VLDB Endowment, 2010,3(1-2):340−351. [doi: 10.14778/1920841.1920887]

[5] Zhang S, Yang J, Jin W. Sapper: Subgraph indexing and approximate matching in large graphs. Proc. of the VLDB Endowment, 2010,3(1-2):1185−1194. [doi: 10.14778/1920841.1920988]

[6] Sun Z, Wang H, Wang H, Shao B, Li J. Efficient subgraph matching on billion node graphs. Proc. of the VLDB Endowment, 2012, 5(9):788−799. [doi: 10.14778/2311906.2311907]

[7] Qu KS, Zhai YH. Posets, inclusion degree theory and FCA. Chinese Journal of Computers, 2006,29(2):219−226 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2006.02.004]

[8] Ren X, Liu J, Yu X, Khandelwal U, Wang L, Han J. Cluscite: Effective citation recommendation by information network-based clustering. In: Proc. of the 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2014. 821−830. [doi: 10.1145/2623330.2623630]

[9] Duan L, Street WN, Liu Y, Lu H. Community detection in graphs through correlation. In: Proc. of the 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2014. 1376−1385. [doi: 10.1145/2623330.2623629]

[10] Cordella LP, Foggia P, Sansone C, Vento M. A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2004,26(10):1367−1372. [doi: 10.1109/TPAMI.2004.75]

[11] Zhao X, Xiao C, Lin X, Wang W, Ishikawa Y. Efficient processing of graph similarity queries with edit distance constraints. VLDB Journal, 2013,22(6):727−752. [doi: 10.1007/s00778-013-0306-1]

[12] Ullmann JR. An algorithm for subgraph isomorphism. Journal of the ACM, 1976,23(23):31−42. [doi: 10.1145/321921.321925]

[13] Shang H, Zhang Y, Lin X, Yu J. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. Proc. of the VLDB Endowment, 2008,1(1):364−375. [doi: 10.14778/1453856.1453899]

[14] Han WS, Lee J, Lee JH. Turbo iso: Towards ultrafast and robust subgraph isomorphism search in large graph databases. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. 2013. 337−348. [doi: 10.1145/2463676.2465300]

[15] Ma H, Shao B, Xiao Y, Chen LJ, Wang H. G-SQL: Fast query processing via graph exploration. Proc. of the VLDB Endowment, 2016,9(12):900−911. [doi: 10.14778/2994509.2994510]

[16] He H, Singh AK. Closure-Tree: An index structure for graph queries. In: Proc. of the 22nd Int'l Conf. on Data Engineering (ICDE 2006). IEEE, 2006. 38−38. [doi: 10.1109/ICDE.2006.37]

[17] Zhang XC, Yu H, Gong XJ. A random walk based iterative weighted algorithm for sub-graph query. Journal of Computer Research and Development, 2015,52(12): 2824−2833 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2015.20140801]

[18] Yuan Y, Wang G, Xu JY, Chen L. Efficient distributed subgraph similarity matching. The VLDB Journal, 2015,24(3):369−394. [doi: 10.1007/s00778-015-0381-6]

[19] Gupta M, Gao J, Yan X, Cam H, Han J. Top-*k* interesting subgraph discovery in information networks. In: Proc. of the 2014 IEEE 30th Int'l Conf. on Data Engineering. IEEE, 2014. 820−831. [doi: 10.1109/ICDE.2014.6816703]

[20] Rangapuram SS, Bühler T, Hein M. Towards realistic team formation in social networks based on densest subgraphs. In: Proc. of the 22nd Int'l Conf. on World Wide Web. ACM Press, 2013. 1077−1088. [doi: 10.1145/2488388.2488482]

[21] Khan A, Wu Y, Aggarwal CC, Yan X. Nema: Fast graph search with label similarity. Proc. of the VLDB Endowment, 2013,6(3): 181−192. [doi: 10.14778/2535569.2448952]

[22] Zou L, Chen L, Lu Y. Top-*k* subgraph matching query in a large graph. In: Proc. of the ACM First Ph. D. Workshop in CIKM. ACM Press, 2007. 139−146. [doi: 10.1145/1316874.1316897]

[23] Peng P, Zou L, Özsu MT, Chen L. Processing SPARQL queries over distributed RDF graphs. Computer Science, 2016,25(2):1−26. [doi: 10.1007/s00778-015-0415-0]

[24] Zheng W, Zou L, Lian X, Wang D, Zhao D. Efficient graph similarity search over large graph databases. IEEE Trans. on Knowledge and Data Engineering, 2015,27(4):964−978. [doi: 10.1109/TKDE.2014.2349924]

[25] Zheng W, Zou L, Peng W, Yan X, Song S, Zhao D. Semantic SPARQL similarity search over RDF knowledge graphs. Proc. of the VLDB Endowment, 2016,9(11):840−851. [doi: 10.14778/2983200.2983201]

[26] Lian X, Chen L, Wang G. Quality-Aware subgraph matching over inconsistent probabilistic graph databases. IEEE Trans. on Knowledge and Data Engineering, 2016,28(6):1560−1574. [doi: 10.1109/TKDE.2016.2518683]

[27] Hong L, Zou L, Lian X, Philip SY. Subgraph matching with set similarity in a large graph database. IEEE Trans. on Knowledge & Data Engineering, 2015,27(9):2507−2521. [doi: 10.1109/TKDE.2015.2391125]

[28] Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[29] Deppisch U. S-Tree: A dynamic balanced signature index for office retrieval. In: Proc. of the 9th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM Press, 1986. 77−87. [doi: 10.1145/253168.253189]

[30] Mamoulis N, Cheung DW, Lian W. Similarity search in sets and categorical data using the signature tree. In: Proc. of the 19th Int'l Conf. on Data Engineering. IEEE, 2003. 75−86. [doi: 10.1109/ICDE.2003.1260783]

[31] Chen Y, Chen Y. On the signature tree construction and analysis. IEEE Trans. on Knowledge and Data Engineering, 2006,18(9): 1207−1224. [doi: 10.1109/TKDE.2006.146]

[32] Kontaki M, Manolopoulos Y, Nanopoulos A. Compressing large signature trees. Lecture Notes in Computer Science, 2003,2798: 163−177. [doi: 10.1007/978-3-540-39403-7_14]

[33] Fomin FV, Grandoni F, Pyatkin AV, Stepanov AA. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. on Algorithms, 2008,5(1):596−600. [doi: 10.1145/1435375.1435384]

[34] Couturier JF, Heggernes P, van't Hof P, Krastsch D. Minimal dominating sets in graph classes: Combinatorial bounds and enumeration. Theoretical Computer Science, 2013,487:82−94. [doi: 10.1016/j.tcs.2013.03.026]

[35] Rooij JMMV. Exact Exponential-Time Algorithms for Domination Problems in Graphs. Boxpress, 2011.

[36] Yang Y, Pedersen JO. A comparative study on feature selection in text categorization. In: Proc. of the 14th Int'l Conf. on Machine Learning. Morgan Kaufmann Publishers Inc., 1998. 412−420.

[37] Viger F, Latapy M. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. 2005, 3595:440−449. [doi: 10.1007/11533719_45]

[38] Chakrabarti A, Parthasarathy S. Sequential hypothesis tests for adaptive locality sensitive hashing. Computer Science, 2014. [doi: 10.1145/2736277.2741665]

附中文参考文献:

[7] 曲开社,翟岩慧.偏序集、包含度与形式概念分析.计算机学报,2006,29(2):219−226. [doi: 10.3321/j.issn:0254-4164.2006.02.004]

[17] 张小驰,于华,宫秀军.一种基于随机游走的迭代加权子图查询算法.计算机研究与发展,2015,52(12):2824−2833. [doi: 10.7544/issn1000-1239.2015.20140801]

李瑞远(1990−),男,湖南郴州人,博士生,主要研究领域为时空数据管理,城市计算,云计算.

洪亮(1982−),男,博士,副教授,主要研究领域为图数据库,社会网络,时空数据管理.