第 28 卷第 3 期　　四川理工学院学报(自然科学版)　　Vol. 28　No. 3
2015 年 6 月　　Journal of Sichuan University of Science & Engineering(Natural Science Edition)　　Jun. 2015
文章编号:1673-1549(2015)03-0027-06　　DOI:10.11863/j.suse.2015.03.06

支持增量图数据的超图查询算法研究

孙勤红

(三江学院计算机科学与工程学院, 南京 210012)

摘　要:当前大部分图查询算法都是针对静态图数据,不适用于现实应用中不断更新的图数据。针对这一问题,提出支持增量图数据的超图查询算法。该算法将数据图分解成直至单个顶点的子图,然后从单个顶点的子图开始求它到查询图的子图同构,直到求出数据图到查询图的子图同构结果,算法在数据图增加时只需将新加入的数据图进行分解即可,不必重新计算。通过分析证明,所提算法时间和空间复杂度不随数据图的增加而呈线性增长,节省了大量时间和空间代价。

关键词:增量图数据;超图查询;算法;子图同构

中图分类号:TP311　　**文献标志码**:A

引　言

图作为一种复杂的数据结构被应用到各个领域中,因此图查询[1]作为图数据库管理的基本工具受到越来越多的关注。

图结构数据的复杂性决定了图查询的难度。图查询问题会最终转化到子图同构问题上来,所以子图同构是解决图查询问题的关键。子图同构是 NPC 问题[2],求同构子图的最通用的方法是基于搜索树的回溯。为了防止搜索树变得过大,研究人员已提出了很多不同的优化方法,如 Ullman 方法[3]、VF 方法[4]以及、VF2 算法[5],另外还有利用网格分割方法、神经网络方法和遗传算法等[6]。为了加快图查询处理过程,已有的方法大部分采用"过滤—验证"框架处理精确子图查询问题,索引的特征有基于路径的[7],基于树的[8],还有基于子图的[9-10]。

当前的算法基本采用"过滤—验证"框架结构,并需要对数据图与查询图逐个作子图同构验证。然而,这些方法都只适用于静态的图查询问题,而不适合在不断更新的图数据库上进行图查询。

动态图数据上的图查询问题面临的难点有两点:(1)图数据库是不断变化更新的;(2)数据库更新过程中用于回答查询的时间有限,不能让用户无限制地等待。正是基于动态图数据的这些特点,现有的方法不适用于动态图查询,除非每次更新都重新建立索引然后查询,这样的代价是很高的,而且更新后这些方法可能引起结果错误。

基于现有方法的不足以及动态图数据频繁更新的特点,本文提出了支持增量图数据的超图查询方法。支持增量图数据的超图查询方法的关键点在于当数据库增量更新后,只需将新加入数据图分解并将结果集合加入到原有分解集合中即可,并且在分解过程中可以用到原有结果,避免重新计算,提高了增量图数据的超图查询性能,节省时间和空间代价。

1　基本概念

给定图 $G=(V_G,E_G,l_{GV},l_{GE})$,$g=(V_g,E_g,l_{gV},l_{gE})$ 为 G 的子图。V_G 和 V_g 分别表示图 G 和子图 g 中所有顶点的集合,E_G 和 E_g 分别表示图 G 和子图 g 中所有边的集合,l_{GV} 和 l_{gV} 分别表示图 G 和子图 g 中所有顶点到点标签 V_c 的映射函数的集合,l_{GE} 和 l_{gE} 分别表示图 G 和子图 g 中所有边到边标签 E_c 的映射的集合。

定义 1　子图同构。假设 S 是 g 的子图,如果存在一

收稿日期:2015-04-21

作者简介:孙勤红(1979-),女,山东郯城人,讲师,硕士,主要从事数据挖掘方面的研究,(E-mail) 22113460@qq.com

个函数$f:V_G \to V_g$,且f是从G到S的同构,那么,称f是从G到g的子图同构。

定义2 图的差。如果$V_g = V_G$,两者的差$G_c = G - g$是边集$E_c = E_G - E_g$;如果V_g包含于V_G,两者的差$G_c = G - g$是G的子图$G_c = (V_c, E_c, l_{cV}, l_{cE})$,其中:

(1) $V_c = V_G - V_g$其中l_{cV}是V_c到点标签的映射函数;

(2) $E_c = E_G - E_g$其中l_{cE}是E_c到边标签的映射函数;

定义3 超图查询。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$q$,找出所有的$G_i$满足$G_i$子图同构于$q$。

定义4 增量超图查询。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$q$,完成超图查询后数据库$D$增量更新(增加数据图)变为$D_u$,再次找出所有的$G_i$满足$G_i$子图同构于$q$。

2 图分解算法

在离线阶段,将每个数据库图分解成子图的集合和边的集合。在本文中,将每个数据库图分解为两个子图,然后分别进一步递归地分解两个子图,直到子图是单个顶点的图。

将一组数据库图$D = \{G_0, G_1, G_2, \cdots, G_n\}$分解($n$为有穷自然数),这里所指的分解是将每个$G_i$分解,得到一组由四元组$< G, G', G'', E >$构成的有限集合$DB$称为图数据库$D$的数据图分解集合。

如果图数据库中的几个图$G_i, G_j, \cdots$有公共子图g',或者在一个数据图G_i中出现多次子图g',那么子图g'成为公共子图。在DB中用四元组$< G, G', G'', E >$表示G的分解就可以省去重新分解和计算的工作。

在已分解的DB中,最大的数据库图的公共子图S_{max}称为最大公共子图。

具体的分解算法如算法1所示,算法Decomposition(D)的输入是要被分解的数据库图集合$D = \{G_0, G_1, G_2, \cdots\cdots, G_n\}$,分解后输出元组集合用$DB$表示,其初值为空。

算法首先计算出数据库D中的每个数据图G_i的顶点个数,按照顶点个数从小到大对数据图进行排序,再对排序后的数据图调用Decompose(G)将其分解。

算法用四元组集合记录分解得到的所有子图。子图的个数与子图的平均顶点个数成反比。如果子图的个数增加,那么子图的平均顶点个数就会减少。反之,子图的平均顶点个数就会增加。为了保证子图的平均顶点个数增加,算法对待处理的数据图按顶点个数进行排序,并按照顶点个数从小到大的顺序进行分解。当数据图中存在"包含"关系时,小图首先被分解,大图的分解也会节省代价[11]。

算法1 数据图分解算法

输入:图数据库$D = \{G_1, G_2, \cdots, G_n\}$;

输出:数据图分解集合DB;

1:Decomposition(D)

2:数据图库$D = \{G_1, G_2, \cdots, G_{n+1}\}$,$DB = \Phi$;

3:按顶点个数从小到大的顺序将D中的数据图进行排序,得到$D = \{G_{t1}, G_{t2}, \cdots, G_{tn+1}\}$;

4:FOR $i = t1$ *to* $tn + 1$

5:Decompose(G_i);

6:END FOR;

7:Decompose(G_i)

8:设DB是已有的分解结果,$S_{max} = \Phi$;

9:IF(G中只有一个顶点)THEN

10:返回结果$< G$, NULL, NULL$>$;

11:ELSE 调用子图匹配算法 Matchgraph(G)在已分解的$tuple \in DB$中求每个$parent$

12:到G的子图同构,并找出其中的最大子图S_{max},即

13:$S_{max} = max\{parent \mid \exists < parent, sub1, sub2, E > \in DB \wedge parent$是$G$的子图$\}$;

14:END IF

15:IF(S_{max}与G同构)THEN EXIT;//同构是指S_{max}与G的维数相同;

16:ELSE IF(S_{max} =NULL) THEN

17:随机选择G的一个子图S_{max};

18:求G中S_{max}与$G - S_{max}$之间的边集E;

19:将$< G, S_{max}, G - S_{max}, E >$加入到$DB$中;

20:Decompose(S_{max});

21:Decompose($G - S_{max}$);

22:ELSE IF(S_{max}的顶点数 = G的顶点数 && S_{max}边数 < G边数)THEN

23:求属于G而不属于S_{max}的边集E;

24:从G中删除边集E;

25:分解G,将$tuple1 < G, S_{max}, Null, E >$加入$DB$中;

26:ELSE 求G中S_{max}与$G - S_{max}$之间的边集 E;

27:将$< G, S_{max}, G - S_{max}, E >$加入到$DB$中;

28:Decompose($G - S_{max}$);

29:END IF

30: END IF

31: END IF

32: RETURN DB;

在分解一个数据图 G 时,首先在已经得到的子图中查找最大的子图 S_{max},然后将图 G 分解为 S_{max} 和 $G - S_{max}$ 两部分。只有当找不到这样的 S_{max} 时,才将 G 随机分解为两部分。当 S_{max} 与图数据库中的一个数据图 G_i 的顶点数相同,但是 S_{max} 的边数小于 G_i 的边数时,求属于 G_i 而不属于 S_{max} 的边集 E,从 G_i 中删除这些边,并分解 G_i,增加四元组 $< Gi, S_{max}, NULL, E >$。这样既能保证子图顶点个数的最大化,又能节省计算代价。

Decomposition (D) 算法最重要的特点是对增量图数据的分解效果明显,图数据库 D 已经应用 Decomposition (D) 算法分解后,在 D 中又新增加了一个数据图 G_{n+1}, 传统的超图查询方法需要重新构造索引进行更新, 而应用 Decomposition (D) 算法只需执行 Decompose (G_{n+1}) 将新加入的数据图 G_{n+1} 分解即可,达到只对增量更新,而无需全部重新计算的特点。Decomposition (D) 尤其适用于图数据较大以及需要在运行时向数据库中新增数据图的情况。

3　子图的映射集合

首先采用标准测试问题比较 chaoticMOPSO 与经典的 NSGA2 以及当前最新提出的 BB - MOPSO 两种算法以验证本文所提算法的性能。

将分解得到的分解结果集 DB 中的单个顶点的子图先与查询图进行匹配,匹配成功的这些单个顶点的图合并成大的子图再与查询图进行匹配,这一过程递归地进行直至合并成的最后图为图数据库中的数据图本身。在此过程中会遇到两个问题:

(1)分解出的最小子结构是单个顶点的子图,需要有方法能够求得单个顶点子图到查询图的子图同构过程;

(2)得出分解结果集合 DB 且 $< G, G', G'', E > \in DB$, 所有这样的元组中已求出 G' 和 G'' 到查询图的子图同构,那么需要有方法将 G' 和 G'' 合并成 G, 进而求子图同构。

本文采用算法 2 给出的单点同构检测算法来解决问题(1)。

算法 2　单点同构检测算法

//单点同构检测 Vertex_Test ($v, lable, G_I$)

输入:分解结果 DB, G_I;

输出:同构映射集合 F。

1: $G_I = (V_{GI}, E_{GI}, l_{GIV}, l_{GIE})$; $F = \Phi$; $lable = l_{GIV}(v)$;

2: Vertex_Test ($v, lable, G_I$)

3: FOR EACH $v_I \in V_I$

4: IF ($lable = l_{GIV}(v_I)$) THEN

5: $f(v) = v_I$;

6: $F = F \cup \{f\}$;

7: END IF

8: END FOR

9: RETURN F;

采用算法 3 的子图映射组合算法 MergeTuple(Tuple $tuple$, $Graph\ target$) 可获得分解 DB。

算法 3　图映射组合算法

输入: $sub1, sub2, G_I, E, F_1, F_2$;

输出:同构映射集合 F。

1: MergeTuple(Tuple $tuple$, $Graph\ target$)

2: $tuple = < parent, sub1, sub2, E > \in DB$;

3: IF ($sub2 ==$ NULL) //对于子图 $sub1$ 到 $target$ 的每种可能的映射组合 $f_1 \in F_1$ 进行如下检查:

4: FOR EACH $e \in Eparent$ && $e \notin E_{sub1}$, $from\ parent\ V(e) = v_1$, $to\ parent\ V(e) = v_2$

5: $IF(e_1 \in E_{target}$ && from

$V_{target}(e_1) = f_1(v_1)$ && to

$target\ V(e_1) = f_1(v_2)$

6: && arg

$type$

$t\ et\ V(e_1) =$ type

$parent\ V(e)$)

7: $f_1 \in F_1$, 得到新的映射 $f: V_{sub1} \rightarrow V_{target}$, 且 $f(n) = f_1(n)$;

8: 将 f 加入到 $parent$ 的同构映射表 F 中,即 $F = F \cup \{f\}$;

9: END IF

10: END FOR

11: ELSE // 对于每个 $f_1 \in F_1, f_2 \in F_2$ 进行如下检查:

12: IF ($f_1(V_{sub1}) \cap f_2(V_{sub2}) = \Phi$ && // 检查两个子图到 $target$ 中的映射是不相交的

13: // 对于两个子图到 $target$ 的每种可能的映射组合,判断边的约束:

14: FOR EACH $e \in E_{parent}$ && $from$

$parent\ V(e) = v_1 \in V_{sub1}$ && to

$parent\ V(e) = v_2 \in V_{sub2}$

15: IF (($e_1 \in E_{target}$ && $from$

$V_{target}(e_1) = f_1(v_1)$ && *to*

target $V(e_1) = f_1(v_2)$

16: && arg

type

t et $V(e_1) = type$

parent $V(e))$ OR

17: FOR EACH $e \in E_{parent}$ && to

parent $V(e) = v_1 \in V_{sub1}$ && from

parent $V(e) = v_2 \in V_{sub2}$

18: IF $((e_1 \in E_{target}$ && arg

to

V_t *et* $(e_1) = f_1(v_1)$ && from

parent $V(e_1) = f_1(v_2)$

19: && arg

type

t et $V(e_1) = type$

parent $V(e))$

20: $f: V_{sub1} \cup V_{sub2} \rightarrow V_{target}$// 对于满足 a 和 b 的每种组合合成新的映射

21: 将 f 加入到 *parent* 的同构映射表 F 中,即 $F = F \cup \{f\}$;

22: END IF

23: END FOR

24: END IF

25: END IF

26: RETURN F;

对于 *DB* 中一个四元组 $< parent, sub1, sub2, E > \in DB$ 已经找到了所有从 *sub*1 和 *sub*2 到查询图的子图同构 f_1 和 f_2, 那么需要将他们组合成从 *parent* 到查询图的子图同构 f, 即需要判断由 f_1 和 f_2 能否组合成 *parent* 到 *target* 的子图同构 f, 这一工作由 MergeTuple (Tuple *tuple*, Graph *target*) 完成。该首先对 *sub*2 为空的子图的情况进行了分析,若 *sub*2 为空,那么只需要对属于 *parent* 而不属于 *sub*1 的边进行检测即可。检测包括两部分:分别是边的连接顶点类型和边类型,对于无向图且边上无标签的图可以将边的类型检测忽略;其次对 *sub*1 和 *sub*2 都不为空的情况进行检测,此时需要对属于 *parent* 而不属于 *sub*1 也不属于 *sub*2 的边进行检测,检测的步骤同前面。

4 实验仿真

为了验证本文算法的有效性和可靠性,采用 MIT 地质图像测试数据库为研究对象,选择测试图像 1 和测试图像 2 验证本文算法的可扩展性和运行效率。对比本文算法和 cIndex 算法,评价指标包括算法可扩展性、执行效率、离线构造时间、索引特征的过滤能力。对比结果如图 1 ~ 图 6 所示。

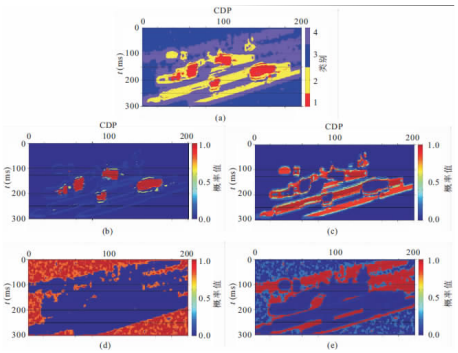

图 1 测试图像 1 及其分解结果

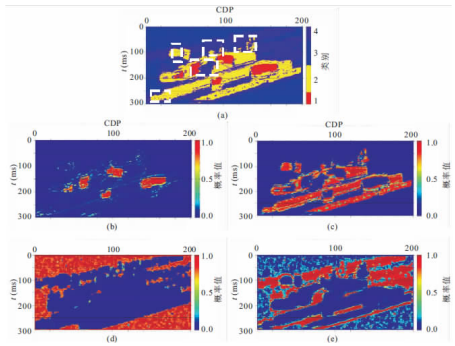

图 2 测试图像 2 及其分解结果

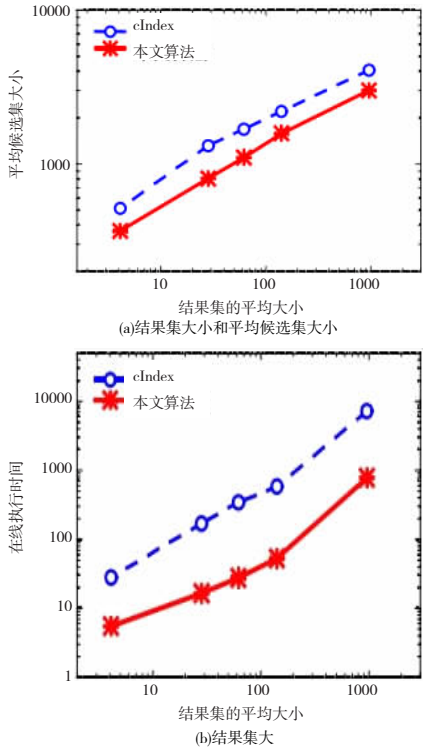

图 3 候选集大小和执行时间

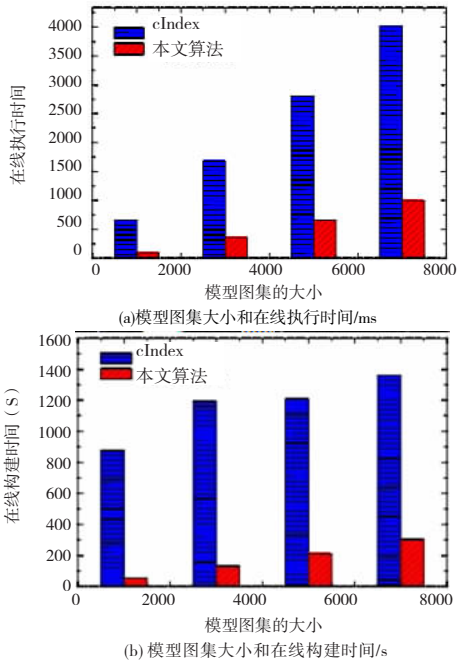

图 4　可扩展性

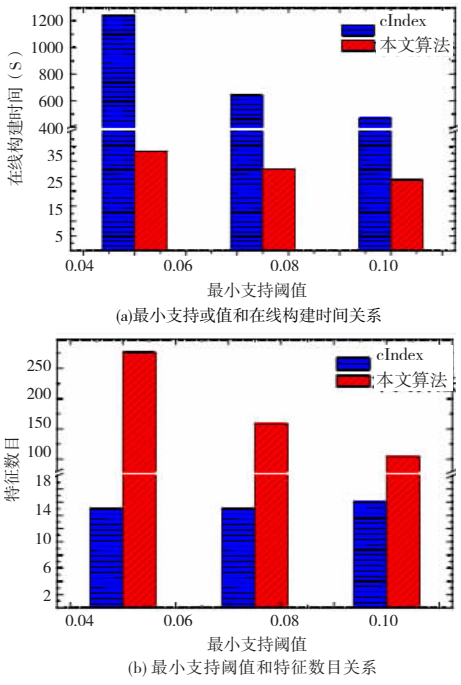

图 5　离线构造性能

由图 3 可知,本文算法所需响应时间比 cIndex 方法小很多,运行效率得到了大约 1 个数量级的提升,主要是因为本文算法可以实现数据集的压缩,从而达到大大降低运行计算的代价。

由图 4 可知,本文算法的可扩展性优于 cIndex 方法效果较好。由图 5 离线构造性能实验结果可知,本文算法在执行效率和预处理结果两个方面均优于 cIndex 方法。

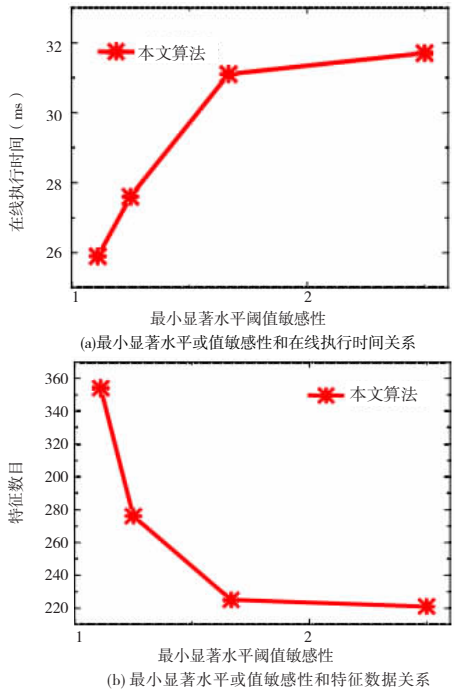

图 6　阈值敏感性

由图 6 可知,随着阈值敏感性的增大,数据图集的索引特征呈现下降趋势,而执行时间却变长,从而说明索引特征的大小和执行效率上存在矛盾的关系。

通过对可扩展性、执行效率、离线构造时间、索引特征的过滤能力四个性能评价指标的仿真实验可知,本文算法不但执行效率高,同时也具有较低复杂度,效果较好。

5　结束语

本文提出了一种新的超图查询算法—支持增量图数据的超图查询算法,该算法将数据图分解成直至单个顶点的子图,然后从单个顶点的子图开始求它到查询图的子图同构,直到求出数据图到查询图的子图同构结果。通过文中的分析表明,相较 cIndex 算法,本文提出的算法对超图的查询更加地简单省时,同时,所提算法空间复杂度不随数据图的增加而呈线性增长,节省了大量空间代价。

参 考 文 献:

[1] Wang Xiaoli,Ding Xiaofeng,Tung A,et al. An efficient graph indexing method[C]//Proceeding of IEEE 28th International Conference on Data Engineering(ICDE), DC,Washington,April 1-5,2012:210-221.

[2] France R,Gabriel V. Chemical graphs,chemical reaction

graphs,and chemical graph transformation[J].Theoretical Computer Science,2005,127(1):157-166.

[3] Moustafa W,Deshpande A,Getoor L.Ego-centric graph pattern census[C]//Proceeding of IEEE 28th International Conference on Data Engineering (ICDE), DC, Washington,April 1-5,2012:234-245.

[4] Hu Y,Wu W,Zhang B.A fast method to identify the order of frequency-dependent network equivalent[J]. IEEE Transactions on Power Systems,2015,PP(99):1-9.

[5] Shang Huiliang, Tao Yudong, Gao Yuan, et al. An improved invariant for matching molecular graphs based on VF2 algorithm[J]. IEEE Transactions on Systems, Man,and Cybernetics:Systems,2015,45(1):122-128.

[6] Perez-Ortiz M,Gutierrez P A,Hervas-Martinez C,et al. Graph-based approaches for over-sampling in the context of ordinal regression[J].IEEE Transactions on Knowledge and Data Engineering, 2015, 27 (5): 1233-1245.

[7] Llados J, Marti E, Villanueva J. Symbol recognition by error-tolerant sub-graph matching between region adjacency graphs[J].IEEE Transactions on Pattern Analysis and Machine Intelligence,2001,23(10):1137-1143.

[8] Akrivi V,Michalis V,Klaus B.Algorithms and models for the Web-Graph[M]. Berlin Heidelberg:Springer-Verlag, 2008.

[9] Jin R,Ruan N,Dey S,et al.Scaling reachability computation on large graphs[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data,Arizona,May 20-24,2012:169-180.

[10] Thomsen J,Yiu M,Jensen C.Effective caching of shortest paths for location-based services[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data,Arizona,May 20-24,2012:313-324.

[11] 张硕,李建中,高宏,等.一种多到一子图同构检测方法[J].软件学报,2010,21(3):401-414.

Research on Supergraph Query Algorithm of Support Incremental Graph Data

SUN Qinhong

(School of Computer Science and Engineering, Sanjiang University, Nanjing 210012, China)

Abstract: Most current graph query algorithms are applicable for static graph data, but cannot be used in constantly updated map data in real world applications. Aiming at this problem, the supergraph query algorithm of support incremental map data is put forward. The data graph is divided into subgraphs with a single vertex by using the proposed algorithm, and then from the single vertex subgraph to find it's subgraph isomorphic to query graph, until the subgraph isomorphism results of data graph to query graph are found. When data graph increases, algorithm only needs to decomposition the newly added data graphs, and do not need to compute. The analysis shows that, the time and space complexity of proposed algorithm does not increase linearly with the increase of data graph, which saves much time and space costs.

Key words: increment map data; supergraph query; algorithm; subgraph isomorphism