基于分区过滤-增量验证的图编辑相似查询

王习特 白 梅 王朝金 马 茜 李冠宇
（大连海事大学信息科学技术学院 辽宁 大连 116000）

摘 要 图编辑相似查询问题是指从图集G中查询出所有与查询图q的图编辑距离（Graph Edit Distance，GED）在给定阈值τ内的数据图.由于GED计算是NP-Hard问题，现有的研究多采用过滤-验证框架进行查询，对未能过滤掉的图采用A*-GED算法验证.本文提出了分区过滤-增量验证框架PFIV来处理图相似查询问题，在增强过滤效果的同时，还能加快验证速度.首先，在过滤阶段提出了2种分区策略，用来加快分区速度.(1)映射顶点顺序策略：在分区过程中，基于图的特征信息和结构信息提出分区时顶点的映射顺序，尽快过滤掉不相似的图，减少计算量；(2)分区结束条件策略：在分区过程中，设置分区结束条件，加快不相似图的过滤速度.其次，在验证阶段提出了增量验证策略，利用过滤阶段保留的映射结果，设计状态空间树，进行增量验证，加快验证阶段的计算.最后，通过大量实验验证了PFIV能够高效地处理图编辑相似查询问题，对比原有算法，查询效率提高8%～17%，并证明了所提出策略的有效性.

关键词 图相似；GED；分区过滤；增量验证；图数据
中图法分类号 TP18 **DOI号** 10.11897/SP.J.1016.2024.00375

Graph Edit Similarity Query Based on Partitioned Filtering-Incremental Verification

WANG Xi-Te BAI Mei WANG Chao-Jin MA Qian LI Guan-Yu
(*Information Science and Technology College*, *Dalian Maritime University*, *Dalin*, *Liaoning* 116000)

Abstract The Graph Edit Similarity Query problem refers to querying all data graphs from the graph set G that have Graph Edit Distance (GED) within a given threshold τ from the query graph q. Since GED computation is an NP-Hard problem, most existing studies use a filtering-verification framework for querying, and the A*-GED algorithm is used to verify the graphs that fail to be filtered out. In this paper, we propose the Partitioned Filtering-Incremental Verification (PFIV) framework to deal with the graph similarity query problem, which enhances the filtering effect and speeds up the verification speed. First, two partitioning strategies are proposed in the filtering stage to speed up the partitioning. (1) Mapping vertex order strategy: during the partitioning process, the mapping order of vertices during partitioning is proposed based on the feature information and structure information of the graph to filter out dissimilar graphs as soon as possible and reduce the computation; (2) Partitioning end condition strategy: during the partitioning process, the partitioning end condition is set to speed up the filtering of dissimilar graphs. Secondly, an incremental verification strategy is proposed in the validation stage, using the mapping results retained in the filtering stage to design the state space tree for incremental

收稿日期：2023-04-06；在线发布日期：2023-11-28. 本课题得到国家自然科学基金(62002039, 61602076, 61702072, 61976032)、中央高校基本科研业务费专项资金(3132023259)资助. **王习特**(通信作者)，博士，副教授，主要研究领域为大数据管理、离群点检测，E-mail：xite-skywalker@163.com. **白 梅**，博士，副教授，主要研究领域为数据管理、云计算和查询处理. **王朝金**，硕士，主要研究领域为查询处理. **马 茜**，博士，副教授，主要研究领域为数据管理、数据清洗. **李冠宇**，博士，教授，中国计算机学会(CCF)会员，主要研究领域为智能信息处理、物联网、语义网.

verification and speed up the computation in the validation stage. Finally, it is verified through a large number of experiments that PFIV can efficiently handle the graph edit similar query problem, and the query efficiency is improved by 8%～17% compared with the original algorithm, and the effectiveness of the proposed strategy is demonstrated.

Keywords graph similar; GED; partitioned filtering; incremental verification; graph data

1 引 言

随着数据时代的到来,图数据模型在生物化学、社交网络、计算机视觉等领域的重要性不断凸显.图相似查询在不同领域有着广泛应用,包括计算机视觉中的对象识别,药物研发中的化合物分析等[1].目前,图相似查询已成为研究热点.

图相似经典的度量标准有最大公共子图(Maximum Common Subgraph,MCS)[2-7]和图编辑距离(Graph Edit Distance,GED)[8-9]两种.其中以图编辑距离为度量标准的方法最能够体现两个图的结构信息和特征信息差异,且允许噪声的存在.具体地,给定两个图q和g,将图q通过顶点和边的插入、删除、替换标签操作转换为图g所需要的最少操作的个数,即为两个图的编辑距离$ged(q,g)$.当两个图的编辑距离越小,认为两个图之间的差异越小,即越相似.当给定阈值为τ时,两个图之间的编辑距离不超过τ时认为两个图相似,称为图编辑相似,否则不相似.

在制药行业中,常存在着目标药物q有着很好的活性,但获取q需要耗费大量人力和物力的问题,因此想要找到与q具有相似活性的药物.然而手动检查并不现实.考虑到具有相似结构的药物具有相似的活性,因此,将药物表示为图,其中顶点表示原子,边表示化学键,将药物集合转化为一个图集.通过计算q与图集中每一个数据图的编辑距离判断是否相似,是否具有相似活性.如图1所示,想要在给定药物集合$G=\{g_1,g_2\}$找到与q相似的药物.将q转化为g_1需要将边(u_2,u_7)标签替换为b,边(u_7,u_8)标签替换为a,替换顶点u_4标签为B,替换顶点u_6标签为A,共需要4步,得到$ged(q,g_1)=4$,同样得到$ged(q,g_2)=6$,当给定阈值$\tau=5$时,认为q与g_1相似,与g_2不相似.

由于以图编辑距离为度量标准的相似查询具有很好的实用价值,因此本文采用以图编辑距离为相

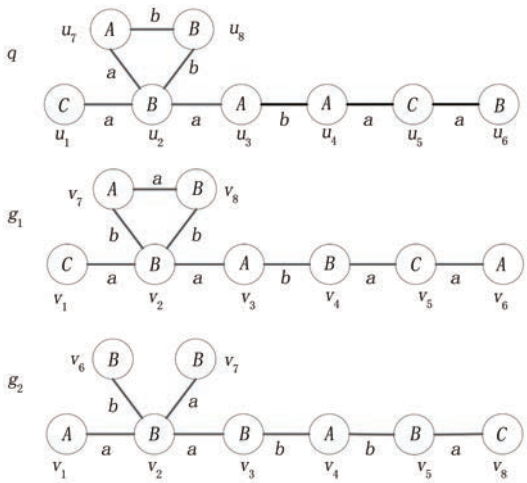


图1 目标药物q与药物集合$G=\{g_1,g_2\}$

似度量标准,简称为图编辑相似查询.然而,图编辑距离的计算已经被证明是一个NP-hard问题[10].目前,Inves算法[8]和AStart-LSa算法[9]是解决图编辑相似查询问题的最先进算法.Inves算法提出基于分区过滤的方法,减少需要计算图编辑距离的数据图数量.但由于分区时没有考虑顶点访问顺序,造成大量不必要映射.且对于不能通过分区过滤的图,直接使用A*-GED算法[11]计算图编辑距离,导致大量重复计算.AStar-LSa算法对A*-GED算法进行了改进,通过避免做虚拟顶点映射,减少映射数目.但由于直接计算图编辑距离,会比先分区过滤再计算图编辑距离代价高.因此,本文为加快查询速度,采用基于分区过滤-增量验证(PFIV)方法处理图编辑相似查询问题.概括起来,本文的主要贡献有:

(1) 在过滤阶段,基于分区的思想进行过滤,提出桥标签集的概念,提高过滤下界.同时提出两种分区策略,加快分区速度.

(2) 在验证阶段,对于未能过滤掉的数据图,利用过滤阶段保留的映射结果确定验证时顶点的映射顺序和匹配顺序,增量地验证,加快验证速度.

(3) 设计了详细的对比实验,选择三个真实数据集,对比先进的图编辑相似查询算法,实验结果表

明分区过滤-增量验证算法PFIV可以有效减少查询时间，提高查询速度.

本文第2节回顾了相关工作；第3节介绍了图相似查询的相关定义；第4节详细描述了本文提出的分区过滤-增量验证框架，针对分区阶段，提出分区过滤算法ParFil，针对验证阶段，提出增量验证算法IncVer；第5节给出了实验结果与分析；第6节对全文进行了总结.

2　相关工作

图相似查询按照不同的相似度量标准可以分为两种，分别是以最大公共子图和以图编辑距离为相似度量标准.

2.1　以最大公共子图为度量的相似查询 MCS

文献[2]最早提出以最大公共子图为度量标准，计算查询图与数据图的最大公共子图，根据查询图与最大公共子图的差异度量相似性，并证明该度量标准的合理性.

文献[3]提出基于图顶点覆盖的MCIS_VC算法，将查询图中必须匹配的顶点逐个标记为“待匹配”状态，减少剩余顶点的映射数量，加快查询速度. 文献[4]考虑公共子图中具有诱导性质的顶点，将逐个标记顶点改进为批处理标记. 此外，文献[5]采用并行多引擎的方法，利用线程池分而治之的思想，最小化子图重复部分，以加快最大公共子图的计算. 文献[6]根据查询图到数据图的子图之间的距离提出GrafD索引结构，利用该索引结构提出Grafil算法计算过滤下界，减少最大公共子图的计算数量. 文献[7]针对Top-k相似查询问题提出了四个具有不同紧密度和计算成本的过滤下界，并设计三个指标，在过滤能力和构建成本之间进行不同的权衡，支持高效剪枝.

2.2　以图编辑距离为度量的相似查询 GED

图编辑距离的概念最早由文献[12]提出，将以图编辑距离为度量的相似查询称为图编辑相似查询. 目前，针对图编辑相似查询的研究大多分为过滤和验证两个阶段.

文献[10,13-14]提出基于索引的方法过滤. 文献[10]和文献[13]从数据图中提取树的结构特征建立索引. 文献[10]将图映射为多组星形结构，对具有n个顶点的图构建为n个具有属性的单层有根树的星形结构，基于该星形结构给出过滤下界. 文献[13]基于字符串相似查询的Q-gram思想，对于图中一个顶点v进行k步深度搜索得到一个K-adjacent树，并利用K-at索引对分解结果进行索引，更好地保留图的结构信息. 文献[14]同样受到Q-gram思想启发，将文献[13]改进为从图中提取路径作为索引特征，针对稠密图可以得到更加严格的下界.

文献[15-16]提出基于子图结构计算图编辑距离下界过滤. 文献[15]将数据图按顶点数和边数之和划分为4种形式的子结构，提高下界值. 文献[16]通过有选择的算法将数据图划分为高选择性的子图，并整合为一个多层索引MLIndex，用于计算过滤下界. 然而形成的子结构有重叠部分，导致结构冗余，影响过滤效果.

文献[8,17]提出基于分区计算图编辑距离下界过滤. 文献[17]提出Pars算法将数据图划分为可变大小的非重叠子图，每个子图称为是一个分区，将图编辑距离计算问题转换为判断是否子图同构问题. 文献[8]基于分区的思想提出Inves算法逐步计算分区下界，对未能过滤的数据图，在验证阶段从过滤阶段得到的顶点数目最多的分区对应的顶点开始映射，利用A*-GED算法[11]精确计算数据图与查询图的图编辑距离.

A*-GED算法是经典的计算图编辑距离的方法，通过最佳优先搜索方式探索两个图顶点之间的所有映射，建立状态空间树，计算每个结点的状态值，再与阈值比较大小进行剪枝，最后得到满足阈值要求的路径. CSI-GED算法[18]将从顶点角度出发查找，改进为从数据图的部分有效边角度出发，并采用深度优先搜索计算图编辑距离. AStar-LSa算法[9]对A*-GED算法提出改进. 顶点映射时考虑从顶点数目少的图向顶点数多的图转化，避免虚拟顶点的设置，减少映射数量. 因此本文将顶点数目少的图默认为查询图，由查询图向数据图转换，避免虚拟顶点的设置. A*-CB算法[19]同样对A*-GED算法提出改进，利用位向量表示状态空间树，在不增加验证时间的情况下，减少了存储空间.

此外，还有一些学者使用编辑距离作为子图相似查询[20]和超图相似查询[21-22]的衡量标准.

与本文最相近的文章是文献[8-9]，都是为加快图编辑相似查询问题提出的解决方案. 文献[8]基于分区的思想过滤减少需要计算图编辑距离的数据图数量. 但分区结果没有被利用到验证阶段图编辑距离的计算过程中，造成大量的冗余计算. 文献[9]中针对A*-GED算法进行了改进，但是由于直接进

行图编辑距离的计算，导致大量不相似的数据图需要计算图编辑距离．此外，这些算法都没有明确给出如何确定顶点的访问顺序，随机的顶点遍历造成空间和时间的浪费．

本文针对图编辑相似查询问题设计了一种更加高效的过滤和验证方法，有效提高了图编辑相似的查询效率．

3 问题描述

本文针对无向带标签图研究相似查询问题，但方法可以拓展到其它类型图的相似查询问题．为方便描述，表1给出本文所使用符号的定义．

表1 符号表示

符号	符号含义
q	无向带标签图
V_q, E_q	顶点集合，边集合
$l_q(u), l_q(u,v)$	图q顶点u的标签，图q边(u, v)的标签
$L_V(q), L_E(q)$	图q顶点的多重标签集，图q边的多重标签集
G	图集
τ	阈值
$ged(q, g)$	图q和图g的编辑距离
$Sim(q, G, \tau)$	在阈值为τ时，图q在图集G中的相似图集

给定一个无向带标签图$q=(V_q, E_q, l_q)$，其中，V_q表示图q的顶点集合，$E_q \subseteq V_q \times V_q$表示图$q$的边集合，其中$V_q \times V_q$指图$q$顶点集合$V_q$与$V_q$做笛卡尔积运算的结果，即任意两个顶点构成的顶点对集合，$E_q \subseteq V_q \times V_q$指顶点对集合的子集，即对应的边集合[7,10]．$|V_q|$表示顶点的个数，$|E_q|$表示边的个数．带标签图的顶点和边都带有标签，$l_q$表示顶点和边的标签函数，$l_q(u)$表示顶点$u$的标签，$l_q(u,v)$表示边$(u, v)$的标签．为方便表示，$L_V(q)$表示图$q$顶点的多重标签集．$L_E(q)$表示图$q$边的多重标签集．$N(u)$表示顶点$u$的邻居顶点集合，表示为$N(u)=\{u' | \forall u' \in V_q, \exists (u, u') \in E_q\}$.

在介绍图编辑距离的概念前，先介绍编辑操作的概念，两个图的编辑操作是指从一个图转换成另一个图的操作，具体包含以下6种操作：(1)删除一个带标签的顶点；(2)增加一个带标签的顶点；(3)替换一个顶点的标签；(4)删除一条带标签的边；(5)增加一条带标签的边；(6)替换一条边的标签．

定义1. 图编辑距离．给定两个图q和g，则q和g之间的图编辑距离是指将q转换为g所需要的最少编辑操作的个数，记作$ged(q, g)$.

得到$ged(q, g)$最直观的办法是将q全部顶点与g中全部顶点进行一一匹配，得到多组映射f．在不同映射的情况下，会得到不同数目的编辑操作．对于最小编辑操作的数目，即图编辑距离$ged(q, g)$.

如图1所示，给定查询图q和图集G，q转化为数据图g_1，当映射$f=[u_1 \to v_1, u_2 \to v_2, u_3 \to v_3, u_4 \to v_4, u_5 \to v_5, u_6 \to v_6, u_7 \to v_7, u_8 \to v_8]$时，需要经过以下4个编辑操作：(1)边$(u_2, u_7)$标签替换为$b$；(2)边$(u_7, u_8)$标签替换为$a$；(3)替换顶点$u_4$标签为$B$；(4)替换顶点$u_6$标签为$A$．4个编辑操作已经是$q$转化为$g_1$的最少数目，得到$ged(q, g_1)=4$．同理$q$转化为数据图$g_2$，当映射$f=[u_1 \to v_1, u_2 \to v_2, u_3 \to v_3, u_4 \to v_4, u_5 \to v_5, u_6 \to v_8, u_7 \to v_7, u_8 \to v_6]$时，需要经过以下6个编辑操作：(1)替换顶点$u_3$标签为$B$；(2)边$(u_4, u_5)$标签替换为$b$；(3)替换顶点$u_5$标签为$B$；(4)替换顶点$u_6$标签为$C$；(5)替换顶点$u_7$标签为$B$；(6)删除边$(u_7, u_8)$．6个编辑操作已经是$q$转化为$g_2$的最少数目，得到$ged(q, g_2)=6$.

定义2. 图编辑相似查询．给定一个图集G，查询图q和阈值τ，图编辑相似查询是指在G内查询出所有满足$ged(q, g) \leqslant \tau$的数据图，称为相似图集，记做$Sim(q, G, \tau)$．表示为$Sim(q, G, \tau)=\{g | g \in G, ged(q, g) \leqslant \tau\}$.

如图1所示，给定查询图q和图集G，给定阈值$\tau=4$，由定义2可以得到q与g_1相似，q与g_2不相似，因此$Sim(q, G, \tau)=\{g_1\}$.

4 分区过滤-增量验证算法

本节中针对图编辑相似查询问题的解决方案进行了详细的描述．首先介绍分区过滤-增量验证的框架，再分别对两部分算法详细描述．

4.1 分区过滤-增量验证框架

本文采用分区过滤-增量验证的框架处理图编辑相似查询．主要包括两个步骤：一是分区过滤阶段，记作ParFil阶段．二是利用分区过滤阶段保留的部分映射结果，增量验证，记作IncVer阶段．考虑到分区过滤阶段得到的映射结果可以加速验证阶段的计算，因此本文在分区过滤计算阶段，对部分映射

结果进行了保留，从而减少了大量的冗余计算，加速了算法整体的运算效率.

下面，算法1给出分区过滤-增量验证算法的框架.

算法1. 分区过滤-增量验证算法——基本框架（PFIV-Frame）

输入：查询图q，图集G和阈值τ.

输出：相似图集$Sim(q, G, \tau)$.

```
1. FOR (g∈G)
2.     初始化映射列表fList←∅;
3.     IF (ParFil(q, g, τ, fList)) //分区过滤阶段
4.         IF (IncVer(q, g, τ, fList)) //增量验证阶段
5.             把g加入到Sim(q, G, τ)中;
```

算法1介绍了整个分区过滤-增量验证算法框架. 在算法1中第2行，对图集中的每一个数据图g，初始化一个空的映射列表$fList$，用来存放分区映射的结果. 第3行为分区过滤算法，判断是否可以通过分区方法过滤掉g，并把分区过程中计算得到的部分映射结果存在映射列表$fList$中. 第4行，针对不能分区过滤掉的g，利用分区映射结果$fList$，进行增量验证，快速判定g是否是查询图q在阈值为τ时的相似图.

4.2 分区过滤阶段

本节针对分区过滤阶段进行介绍. 首先描述了分区过滤的定义定理等；接着提出具体的分区划分策略；最后给出分区过滤算法的具体过程.

4.2.1 分区的理论依据

由于直接采用A*-GED方法计算图编辑距离代价很高，本文采用分区的方法计算编辑距离下界进行过滤. 根据每个数据图g，将查询图q划分为多个不同的子图——分区. 每个分区计算得到一个编辑距离，各个分区的编辑距离总和为q与g的编辑距离下界——分区下界. 下面先介绍分区集的具体定义.

定义3. 分区集. 给出查询图q，将查询图q分为顶点互不相交的多个子图，每个子图称为查询图q的一个分区$part$，将所有分区$part$构成的集合称为查询图q的分区集$PartSet(q)$. 分区集可表示为$PartSet(q)=\{part|V_q=\bigcup_{part}V_{part}$且$\forall part, part', V_{part}\cap V_{part'}=\varnothing\}$.

如图2所示，给定查询图q，查询图q可以分为5个虚线框内的分区，表示为$PartSet(q)=\{part_1, part_2, part_3, part_4, part_5\}$.

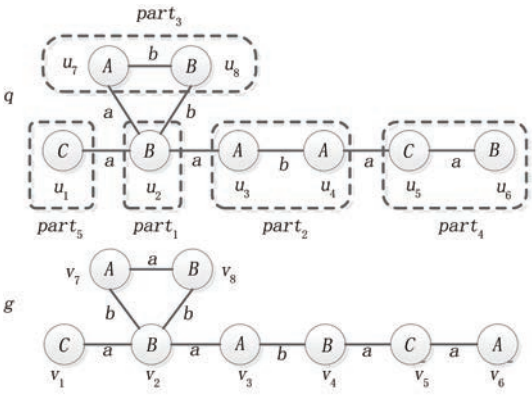


图2　查询图q和数据图g

由定义3可知，每一个分区$part$，可以视为由两部分构成：第一部分是分区内的顶点和边，第二部分是分区之间的边. 不难发现，第二部分是分区之间的边，被两个分区共享，称为桥. 下面给出桥标签集的定义.

定义4. 桥标签集. 给出查询图q的一个分区$part$，其中一条边$(u, u')\in E_q$，如果满足顶点$u\in V_{part}$且顶点$u'\notin V_{part}$，称边(u, u')是$part$中顶点u的一条桥. 顶点u的所有桥对应的标签集合称为顶点u的桥标签集$L_{Bri}(u)$，表示为$L_{Bri}(u)=\{l_q(u, u')|\forall u, u'\in V_q, u\in V_{part}$且$u'\notin V_{part}\}$.

如图2所示，给定查询图q和数据图g，对于分区$part$中的顶点u_2对应的桥有(u_2, u_1)、(u_2, u_3)、(u_2, u_7)、(u_2, u_8)，对应的桥标签集$L_{Bri}(u_2)=\{a, a, a, b\}$.

为增强过滤效果，提高每个分区贡献的编辑距离，每个分区贡献的编辑错误考虑两部分. 第一部分由分区内的顶点和边贡献，第二部分由分区的桥标签集贡献. 下面给出分区内的顶点和边在给定映射f时贡献的编辑距离定义.

定义5. 分区内映射编辑距离. 给定查询图q和数据图g，$part'$是q的一个分区，映射f'由$part'$和g子图$g_{f'}$的顶点对构成，$med_{f'}(part', g_{f'})$指的是将$part'$转化为$g_{f'}$时，分区内的顶点和边贡献的编辑操作数目. 当加入新顶点u，映射到顶点v，形成$f(u)=v$，得到新映射f，分区为$part$时，对应得到的分区内映射编辑距离$med_f(part, g_f)$的计算公式为：

$$med_f(part, g_f)=med_{f'}(part', g_{f'})+\begin{cases}1 & l_q(u)\neq l_g(v)\\ 0 & l_q(u)=l_g(v)\end{cases}$$

$$+\sum_{u'\to v'\in f'}\begin{cases}1 & l_q(u,u')\neq l_g(v,v')\\ 0 & l_q(u,u')=l_g(v,v')\end{cases} \quad (1)$$

其中，当$f=\varnothing$时，$med_f(part,g)=0$.

如图2所示，给定查询图q和数据图g，当加入新顶点u_2，映射到顶点v_2，得到映射$f=[u_2\to v_2]$，对应的分区$part_1$，由公式(1)得到对应的分区内映射编辑距离$med_f(part,g_f)=0$.

下面给出第二部分分区中桥标签集在给定映射f时，贡献的编辑距离定义.

定义6. 桥映射编辑距离. 给定查询图q和数据图g，$part$是q的一个分区，映射f由$part$和g子图g_f的顶点对构成，$bed_f(part,g_f)$指的是将$part$转化为子图g_f时，分区间共享的桥所贡献的编辑操作数目，计算公式为

$$bed_f(part,g_f)=\frac{1}{2}\times\sum_{u\to v\in f}\mathcal{T}\left(L_{Bri}(u),L_{Bri}(v)\right) \quad (2)$$

其中，$L_{EBri}(u)$表示顶点u的桥标签集. $\mathcal{T}(S_1,S_2)=\max\{|S_1|,|S_2|\}-|S_1\cap S_2|$，$S_1$和$S_2$均为多重标签集合，$|S_1|$表示集合$S_1$的大小.

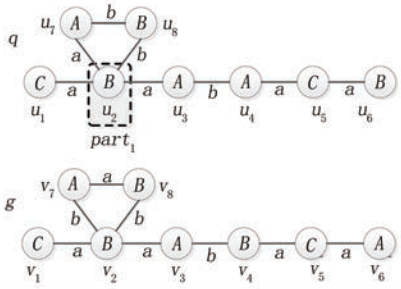
如图2所示，给定查询图q和数据图g，当映射$f=[u_2\to v_2]$时，形成分区$part_1$，有顶点u_2的桥标签集$L_{Bri}(u_2)=\{a,a,a,b\}$，顶点v_2的桥标签集$L_{Bri}(v_2)=\{a,a,b,b\}$. 可以计算得到$\mathcal{T}(L_{Bri}(u_2),L_{Bri}(v_2))=1$. $part_1$的桥映射编辑距离$bed_f(part_1,g_f)=1/2$.

定义7. 分区映射编辑距离. 给定查询图q和数据图g，$part$是q的一个分区，映射f由$part$和g子图g_f的顶点对构成，$ped_f(part,g_f)$指的是将$part$转化为子图g_f时，分区$part$贡献的编辑操作数目，计算公式为

$$ped_f(part,g_f)=med_f(part,g_f)+bed_f(part,g_f) \quad (3)$$

如图3所示，给定查询图q和数据图g，对于查询图q的分区$part_1$，给出在所有可能的映射f下，计算得到的分区内映射编辑距离$med_f(part_1,g_f)$和桥映射编辑距离$bed_f(part_1,g_f)$，结果如图3表格第二列和第三列所示. 并由公式(3)计算得到分区映射编辑距离$ped_f(part_1,g_f)$，如图3表格中第四列所示.

定义8. 分区编辑距离下界. 给定查询图q和数据图g，对于查询图q的任意一个分区$part$，分区



f	$med_f(part_1,g_f)$	$bed_f(part_1,g_f)$	$ped_f(part_1,g_f)$
$u_2\to v_1$	1	3/2	5/2
$u_2\to v_2$	0	1/2	1/2
$u_2\to v_3$	1	1	2
$u_2\to v_4$	0	1	1
$u_2\to v_5$	1	1	2
$u_2\to v_6$	1	3/2	5/2
$u_2\to v_7$	1	1	2
$u_2\to v_8$	0	1	1

图3 分区$part_1$对应的分区映射编辑距离

$part$转化为图g的任意子图的编辑距离下界称为分区编辑距离下界$partlb(part,g)$，分区编辑距离下界公式为

$$partlb(part,g)=min_{f\in\mathcal{F}(part,g_f)}ped_f(part,g_f) \quad (4)$$

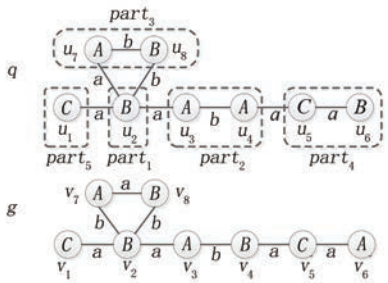
其中，$\mathcal{F}(part,g_f)$为分区$part$和子图g_f顶点映射的集合，映射f为$\mathcal{F}(part,g_f)$中的一个映射，$ped_f(part,g_f)$为分区$part$和图g_f在映射f下对应的分区映射编辑距离.

如图4所示，给定查询图q，当q划分为虚线框内的5个分区时，对应得到各个分区的分区编辑距离下界如图4表格所示.

定理1. 给定查询图q和数据图g，$part$是q的一个分区，对于$part$，考虑桥标签集贡献的编辑错误得到的分区编辑距离下界一定不小于不考虑桥标签集贡献的编辑错误得到的分区编辑距离下界.

证明：由定义7可知，考虑桥标签集贡献的编辑错误时，可以增大每一个分区映射编辑距离. 由定义8可知，分区编辑距离下界是所有可能的分区映射编辑距离中最小的一个，从而可以得到考虑桥边集贡献的编辑错误时，可以增大分区编辑距离下界. 证毕.

如图4所示，给定查询图q和数据图g，对于分区$part_1$，考虑桥标签集贡献的编辑错误时，得到的分区编辑距离下界为1/2，否则为0. 可知考虑桥标



part	f	$med_f(part_1, g_f)$	$bed_f(part_1, g_f)$	$ped_f(part_1, g_f)$
$part_1$	$u_2 \to v_2$	0	1/2	1/2
$part_2$	$u_3 \to v_3$, $u_4 \to v_4$	1	0	1
$part_3$	$u_7 \to v_3$, $u_8 \to v_4$	0	1/2	1/2
$part_4$	$u_5 \to v_5$, $u_6 \to v_4$	0	1/2	1/2
$part_5$	$u_1 \to v_1$	0	0	0

part	f	$partlb(part, g)$
$part_1$	$u_2 \to v_2$	1/2
$part_2$	$u_3 \to v_3$, $u_4 \to v_4$	1
$part_3$	$u_7 \to v_3$, $u_8 \to v_4$	1/2
$part_4$	$u_5 \to v_5$, $u_6 \to v_4$	1/2
$part_5$	$u_1 \to v_1$	0

图4　q各分区的分区编辑距离下界

签集贡献的编辑错误，可以提高分区编辑距离下界.

定义9.　图编辑距离下界. 给定查询图q和数据图g，当对q进行分区得到分区集$PartSet(q)$，查询图q和数据图g的距离下界$lb(q,g)$为

$$lb(q,g)=\sum_{part\in PartSet(q)} partlb(part,g) \quad (5)$$

其中，$partlb(part,g)$是$PartSet(q)$中任一分区$part$的分区编辑距离下界.

如图4所示，给定查询图q和数据图g，得到q和g的编辑距离下界$lb(q,g)=5/2$. 同时注意到，不考虑桥标签集贡献的编辑错误时，得到的图编辑距离下界$lb(q,g)=1$，考虑桥标签集贡献的编辑错误可以提高过滤下界.

定理2.　给定查询图q和数据图g，一定有查询图q和数据图g的编辑距离下界不大于编辑距离，即$lb(q,g)\leqslant ged(q,g)$.

证明：查询图q分区得到分区集$PartSet(q)$，对于其中每个分区$part$和g的一个子图做映射f都可以计算得到分区$part$的一个分区映射编辑距离$ped_f(part,g_f)$，在所有映射对应的映射编辑距离中找到最小值$partlb(part,g)$，即$part$和g子图最小的编辑距离. 又因为$PartSet(q)$中每个$part$都是顶点不相交，因此，有各个分区得到的最小映射编辑距离$lb(q,g)\leqslant ged(q,g)$.　　证毕.

如图4所示，给定查询图q和数据图g，按照图2分区时，得到$lb(q,g)=5/2$，当阈值$\tau=2$时，可以直接判定不相似. 同时注意到，当不考虑桥标签集贡献的编辑错误时，图编辑距离下界$lb(q,g)=1$，当阈值$\tau=2$时，不能判断是否相似. 因此考虑桥标签集的编辑错误可以增强过滤效果.

4.2.2　分区集划分策略

给定查询图q和数据图g，对q进行分区，得到的分区集$PartSet(q)$不同时，下界也不同. 根据定理2可知，当编辑距离下界$lb(q,g)$越大时，过滤效果越好. 下面，提出2条分区策略，快速得到具有编辑距离的分区，同时减少不必要的映射.

策略1：映射顶点顺序

分区时越快出现编辑错误，得到的分区越小，顶点映射数量越小，计算代价越小. 因此，规定顶点映射的先后顺序. 根据图的特征信息和结构信息，提出确定分区时，关于映射顶点顺序应考虑的两条原则：(1)特征信息——优先映射顶点度大的顶点. 顶点的度越大，出现编辑错误的概率也就越大. (2)结构信息——多个连通在一起的点要比多个孤立顶点更容易提供编辑错误，每次优先映射已映射顶点的邻居顶点.

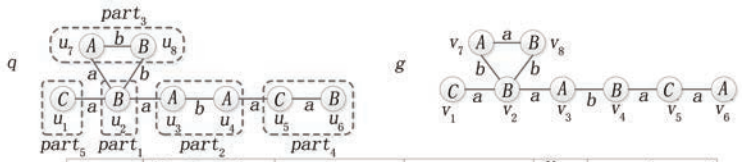
为方便表示，定义两个变量$MapVerOrder_1$和$MapVerOrder_2$. $MapVerOrder_1$存放当前分区已访问顶点的邻居顶点. $MapVerOrder_2$存放成区顶点的邻居顶点，并且这些顶点还未成区. 根据以上原则，首先，访问$MapVerOrder_2$中度最大的顶点u，并把顶点u的邻居节点放入$MapVerOrder_1$. 之后，依次从$MapVerOrder_1$中选取度最大的顶点u进行访问，并将u的邻居节点放入$MapVerOrder_1$. 最后，当形成一个分区后，将$MapVerOrder_1$赋给$MapVerOrder_2$. 再开始下一个分区的计算.

如图5所示，给定查询图q和数据图g，随着待映射顶点的不断加入，$MapVerOrder_1$和$MapVerOrder_2$的变化情况如图5表格所示.

策略2：分区结束条件

对查询图进行分区的目的是令分区$part$提供编辑距离，但希望分区可以在映射数量尽量少的情况

下，提供更大的编辑距离．因此，设计堆存放映射及映射对应的分区映射编辑距离，分区时按照策略1介绍的映射顶点顺序，逐一将待映射顶点加入到当前分区中，直到当前分区与数据图中的任意子图做映射都存在编辑错误，停止加入新顶点，当前分区结束．



part	$MapVerOrder_2$	$MapVerOrder_1$	待映射顶点	V_{part}	$partlb(part,g)$
初始	u_2	——	——	——	——
$part_1$	——	u_2	——	u_2	1/2
	——	u_2, u_3, u_7, u_8, u_1	u_2		
	u_3, u_7, u_8, u_1	——	——		
$part_2$	u_7, u_8, u_1	u_3	——	u_3, u_4	1
	u_7, u_8, u_1	u_3, u_4	u_3		
	u_7, u_8, u_1	u_3, u_4, u_5	u_4		
	u_7, u_8, u_1, u_5	——	——		
$part_3$	u_8, u_1, u_5	u_7	——	u_7, u_8	1/2
	u_8, u_1, u_5	u_7, u_8	u_7		
	u_8, u_1, u_5	u_7, u_8	u_8		
	u_1, u_5	——	——		
$part_4$	u_1	u_5	——	u_5, u_6	1/2
	u_1	u_5, u_6	u_5		
	u_1	u_5, u_6	u_6		
	u_1	——	——		
$part_5$	——	u_1	——	u_1	0
	——	u_1	u_1		
	——	——	——		

图5 变量$MapVerOrder_1$，$MapVerOrder_2$的变化过程

具体地，初始为每个分区$part$设置一个小根堆$\mathcal{H}$存放映射及贡献的编辑距离，并按照贡献的编辑距离排序．每次加入待映射顶点后，检查两个条件，分别是：分区结束条件1：堆顶的分区映射编辑距离不为0，并且堆顶的映射长度不短于堆中其他映射的长度．分区结束条件2：变量$MapVerOrder_1$中顶点都已经成区．当满足条件1或条件2时，分区结束．否则，继续加入待映射顶点，更新堆$\mathcal{H}$．

每一个待映射顶点u应与V_g中每一个待映射顶点v做映射，形成新的匹配对．下面，对于每一个待映射顶点u将V_g分为两部分：匹配顶点组和非匹配顶点组．下面给出具体定义．

定义10. 匹配顶点组．给定查询图q，数据图g，映射f和待映射顶点u，得到顶点u的邻居顶点u'在映射f下对应数据图g的顶点v'，顶点v'的邻居顶点集合$N(v')$称为顶点u在映射f下对应的匹配顶点组，表示为$N_f(u)=\{v|\forall u'\in N(u)且f(u')=v', v\in N(v')\}$．顶点集$V_g$中剩余顶点构成的集合称为非匹配顶点组$\overline{N_f}(u)$．

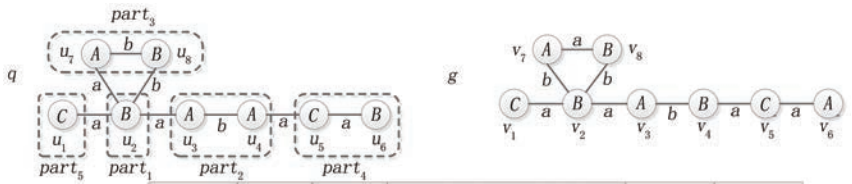
定理3. 给定查询图q和数据图g，当有映射f'和待映射顶点u，映射f'对应的分区为$part'$，顶点u与非匹配顶点组$\overline{N_f}(u)$中任意顶点v映射，形成的新映射f对应的分区$part$，映射编辑距离$ped_f(part, g_f)\geqslant ped_{f'}(part', g_{f'})+1$．

证明：根据映射顶点顺序策略可知，待映射顶点u至少与映射f'的一个顶点u'有一条边连接．根据定义10可知，非匹配顶点组$\overline{N_f}(u)$中的顶点必定和图g_f的顶点没有边连接．因此，形成的新映射f至少会贡献1个增加一条带标签边的编辑错误，即至少贡献1个编辑距离，对应分区映射编辑距离$ped_f(part, g_f)$至少比$ped_{f'}(part, g_{f'})$大1． 证毕．

当有映射f'和待映射顶点u，待映射顶点u应与数据图g中每一个顶点v做映射，得到新映射f，计算

新映射对应的分区映射编辑距离$ped_f(part, g_f)$. 为减少顶点对的数量，根据定理3，将数据图g的顶点集V_g分为匹配顶点组$N_f(u)$和非匹配顶点组$\overline{N_f}(u)$. 对于匹配顶点组$N_f(u)$，待映射顶点u与匹配顶点组$N_f(u)$中每一个顶点v做映射，计算分区映射编辑距离. 对于非匹配顶点组$\overline{N_f}(u)$，待映射顶点u与非匹配顶点组$\overline{N_f}(u)$做映射，分区映射编辑距离加1.

如图6所示，给定查询图q和数据图g，给出建立堆$\mathcal{H}_2$的过程，待映射顶点为u_3时，得到8个映射，如第一部分. 拓展堆顶映射$[u_3 \to v_3]$时，匹配顶点组内顶点有v_2和v_4，得到映射$[u_3 \to v_3, u_4 \to v_2]$和$[u_3 \to v_3, u_4 \to v_4]$. 待映射顶点$u_4$与非匹配顶点组中整体做映射. 由于堆内映射过多，仅在图6中给出3组映射. 此时，堆$\mathcal{H}_2$中有10个映射. 而不对顶点分组时，堆中有14个映射，发现对顶点分组可以有效减少映射数量.



待映射顶点u	$N_f(u)$	$\overline{N_f}(u)$	$\mathcal{H}_2$		条件1	条件2
			f	$ped_f(part_2, g_f)$		
——	——	——	——	——	——	——
u_3	——	——	$u_3 \to v_3$	0	不满足	不满足
			$u_3 \to v_7$	0		
			$u_3 \to v_4$	1		
			……	……		
u_4	v_2, v_4	$v_1, v_3, v_5, v_6, v_7, v_8$	$u_3 \to v_7$	0	不满足	不满足
			$u_3 \to v_3$, $u_4 \to v_4$	1		
			$u_3 \to v_4$	1		
			……	……		

……

图6　堆$\mathcal{H}_2$的建立过程

4.2.3　分区过滤算法描述

利用4.2.2节中介绍的分区策略，下面介绍逐步分区，计算每个分区编辑距离下界算法EstPar(q, g, $MapVerOrder_2$, $fList$).

下面，给出具体的伪代码，如算法2所示.

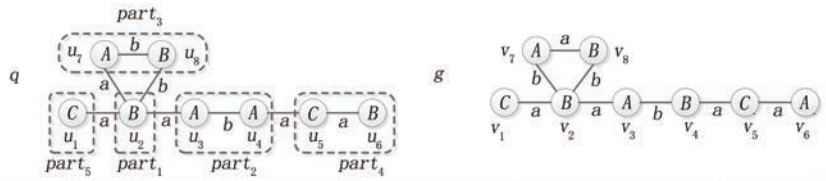
算法2.　计算分区编辑距离下界算法EstPar(q, g, $MapVerOrder_2$, $fList$)

输入：查询图q，数据图g，变量$MapVerOrder_2$和映射列表$fList$.

输出：$\mathcal{H}$堆顶的$ped_f(part, g_f)$.

1. 根据$MapVerOrder_2$初始化变量$MapVerOrder_1$；//策略1
2. 初始化堆$\mathcal{H} \leftarrow \varnothing$；
3. WHILE（不满足条件1且不满足条件2）//策略2
4. 　从$MapVerOrder_1$选择待映射顶点u，从$\mathcal{H}$堆顶映射f；
5. 　FOR ($v \in N_f(u)$)
6. 　　f中加入$f(u)=v$，计算$ped_f(part, g_f)$，更新$\mathcal{H}$；
7. 　u与u的非匹配顶点组$\overline{N_f}(u)$做映射，更新$\mathcal{H}$；
8. 　更新$MapVerOrder_1$；　//策略1
9. 　$\mathcal{H}$堆顶映射加入$fList$；
10. 更新$MapVerOrder_2$；　//策略1
11. RETURN $\mathcal{H}$堆顶的$ped_f(part, g_f)$；

第1-2行，用变量$MapVerOrder_2$的第一个顶点初始化变量$MapVerOrder_1$，初始化小根堆$\mathcal{H}$为空. 第3行，利用策略2判断是否满足分区结束条件. 若不满足，执行第4-8行，利用策略2中提出的对顶点分组的方法，拓展堆顶映射，再判断是否满足分区结束条件. 若满足，执行第9-11行，分区结束，更新变量$MapVerOrder_2$，返回堆顶对应的映射编辑距离，即该分区的分区编辑距离下界. 如图7所示，给定查询图q和数据图g，给出建立分区$part_2$的过程. 需要拓展3次堆顶映射，最后得到分区$part_2$的分区编辑距离下界为1，对应堆顶映射$[u_3 \to v_3, u_4 \to v_4]$.



$MapVerOrder_2$	$MapVerOrder_1$	待映射顶点u	$N_f(u)$	$\overline{N_f(u)}$	$\mathcal{H}_2$		条件1	条件2
					f	$ped_f(part_2, g_f)$		
u_3, u_7, u_8, u_1	u_3	——	——	——	——	——	——	——
u_7, u_8, u_1	u_3, u_4	u_3	——	——	$u_3 \to v_3$	0	不满足	不满足
					$u_3 \to v_7$	0		
					$u_3 \to v_4$	1		
					……	……		
u_7, u_8, u_1	u_3, u_4, u_5	u_4	v_2, v_4	$v_1, v_3, v_5, v_6, v_7, v_8$	$u_3 \to v_7$	0	不满足	不满足
					$u_3 \to v_3$, $u_4 \to v_4$	1		
					$u_3 \to v_4$	1		
					……	……		
u_7, u_8, u_1	u_3, u_4, u_5	u_4	v_2, v_8	$v_1, v_3, v_4, v_5, v_6, v_7$	$u_3 \to v_3$, $u_4 \to v_4$	1	满足	——
					$u_3 \to v_4$	1		
					$u_3 \to v_8$	1		
					……	……		
u_7, u_8, u_1, u_5	——	——	——	——	——	——	——	——

图7 分区$part_2$的建立过程

利用上述计算分区编辑距离下界算法，下面介绍逐步对查询图分区的分区过滤算法 ParFil(q, g, τ, $fList$).

下面，给出具体的伪代码，如算法3所示.

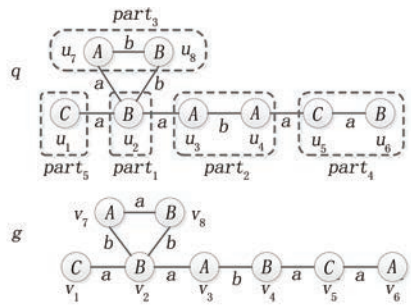
算法3. 分区过滤算法 ParFil(q, g, τ, $fList$)

输入：查询图q，数据图g，阈值τ和映射列表$fList$.

输出：不相似输出 false，否则输出 true.

1. 初始化$lb(q,g)=0$；
2. 初始化$MapVerOrder_2 \leftarrow q$中度最大的顶点；
3. WHILE ($MapVerOrder_2$不为空)
4. $lb(q,g)+=$EstPar(q, g, $MapVerOrder_2$, $fList$)；
   //算法2
5. IF ($lb(q,g)>\tau$)
6. RETURN false； //判定q和g不相似
7. RETURN true； //无法判定q和g是否相似

第1行，初始化图编辑距离下界$lb(q,g)$为0. 第3-6行，根据算法2在每次计算得到新分区的分区编辑距离下界后，都将得到的分区编辑距离下界加入$lb(q,g)$，再比较$lb(q,g)$与阈值τ的大小，若$lb(q,g)$大于阈值τ，说明一定不相似，直接过滤，否则继续分区. 如图8所示，给定查询图q和数据图g，当阈值$\tau=2$时，第4个分区结束后，得到$lb(q,g)=5/2$，可以直接判定不相似.



part	$MapVerOrder_2$	V_{part}	$partlb(part, g)$	堆顶映射
初始	u_2	——	——	——
$part_1$	u_3, u_7, u_8, u_1	u_2	1/2	$u_2 \to v_2$
$part_2$	u_7, u_8, u_1, u_5	u_3, u_4	1	$u_3 \to v_3, u_4 \to v_4$
$part_3$	u_1, u_5	u_7, u_8	1/2	$u_7 \to v_3, u_8 \to v_4$
$part_4$	u_1	u_5, u_6	1/2	$u_5 \to v_5, u_6 \to v_4$
$part_5$	——	u_1	0	$u_1 \to v_1$

$lb(q,g)$	$fList$
5/2	$u_2 \to v_2, u_3 \to v_3, u_4 \to v_4, u_7 \to v_3, u_8 \to v_4, u_5 \to v_5, u_6 \to v_4, u_1 \to v_1$

图8 分区过滤过程

值得注意，当阈值$\tau=4$时，按照上述介绍的分区过滤过程，直至第5个分区结束，得到$lb(q,g)=5/2$，此时$lb(q,g)<\tau$，且变量$MapVerOrder_2$为空，分区已经结束，不能通过分区过滤的方法判断查询图q和数据图g是否相似，则需要进一步验证.

4.3　增量验证阶段

本节针对未能在分区过滤阶段过滤掉的数据图g，提出采用增量的方法快速验证查询图q和数据图g是否相似. 首先，介绍了通过建立查询图q和数据图g状态空间树的方法计算图编辑距离，验证查询图q和数据图g是否相似的理论依据. 之后，提出使用增量的策略快速构建状态空间树，并给出具体增量验证算法.

4.3.1　验证的理论依据

给定查询图q和数据图g，顶点集V_q中所有顶点映射到顶点集V_g中所有顶点构成的映射集合可以表示在一棵状态空间树中. 树共有$|V_q|$层，每一层中都有一个固定的映射顶点u，与顶点集V_g中所有未匹配顶点做映射，形成顶点对，拓展上一层父结点的部分映射，形成多个新的部分映射. 从根结点到某个非叶子结点的所有顶点对表示查询图q和数据图g的一个部分顶点映射，从根结点到某个叶子结点的所有顶点对表示查询图q和数据图g的一个完整顶点映射. 状态空间树的每个结点中除了包含映射顶点u和匹配顶点v，表示一个匹配对外，还包含该结点表示的部分顶点映射对应的状态编辑距离，下面给出状态编辑距离的定义.

定义11　状态编辑距离，给定查询图q和数据图g，查询图q部分顶点对应的子图q_f和数据图g部分顶点对应的子图g_f匹配得到映射f，则查询图q和数据图g在映射f下的状态编辑距离$sed_f(q,g)$为：

$$sed_f(q,g)=fed_f(q_f,g_f)+\mathcal{T}\left(L_V(q_{\bar{f}}),L_V(g_{\bar{f}})\right)+\mathcal{T}\left(L_E(q_{\bar{f}}),L_E(g_{\bar{f}})\right)+\sum_{u\to v\in f}\mathcal{T}\left(L_{Bri}(u),L_{Bri}(v)\right)\quad(6)$$

其中公式(6)由三部分构成，第一部分$fed_f(q_f,g_f)$是子图q_f和子图g_f的映射编辑距离，在下面算法4中介绍增量计算映射编辑距离方法. 第二部分中$q_{\bar{f}}$和$g_{\bar{f}}$分别表示不在映射f下的查询图q的子图和数据图g的子图，$\mathcal{T}\left(L_V(q_{\bar{f}}),L_V(g_{\bar{f}})\right)+\mathcal{T}\left(L_E(q_{\bar{f}}),L_E(g_{\bar{f}})\right)$是子图$q_{\bar{f}}$和$g_{\bar{f}}$顶点和边标签集的差. 第三部分$\sum_{u\to v\in f}\mathcal{T}\left(L_{Bri}(u),L_{Bri}(v)\right)$是映射$f$下对应桥标签集的差.

文献[9]中已证明当$|V_q|=|V_g|$时，由查询图q转换为数据图g时，既不会发生删除带标签顶点的操作，也不会发生增加带标签顶点的操作. 当$|V_q|<|V_g|$时，如果部分映射f对应着查询图q映射到数据图g的部分子图g_f，得到的映射编辑距离为$fed_f(q,g_f)$，则得到的$fed_f(q,g)=fed_f(q,g_f)+(|V_g|+|E_g|-|V_{g_f}|-|E_{g_f}|)$. 由此，在本文中，将默认查询图$q$与数据图$g$顶点数目设为相等.

下面，给出具体增量计算映射编辑距离算法的伪代码，如算法4所示.

算法4.　增量计算映射编辑距离算法 IncCal MapEditDis$(q,g,u,v,f',fed_{f'}(q_{f'},g_{f'}))$

输入：查询图q，数据图g，映射顶点u，匹配顶点v，映射f'，映射编辑距离$fed_{f'}(q_{f'},g_{f'})$.

输出：映射编辑距离$fed_f(q_f,g_f)$.

1. 用映射顶点u和匹配顶点v以及f'初始化映射f；
2. 初始化映射编辑距离$fed_f(q,g)=fed_{f'}(q_{f'},g_{f'})$；
3. IF $(l_q(u)\neq l_g(v))$ //替换顶点标签
4. $fed_f(q_f,g_f)\leftarrow fed_f(q_f,g_f)+1$；
5. FOR $((u,u')\in E_{q_f})$//删除带标签的边或替换边标签
6. IF$((v,f(u'))\notin E_{g_f}$或$l_q(u,u')\neq l_g(v,f(u')))$
7. $fed_f(q_f,g_f)\leftarrow fed_f(q_f,g_f)+1$；
8. FOR$((v,v')\in E_{g_f})$ //增加带标签的边
9. IF$((u,f^-(v'))\notin E_{q_f})$
10. $fed_f(q_f,g_f)\leftarrow fed_f(q_f,g_f)+1$；
11. RETURN $fed_f(q_f,g_f)$；

如图9所示，给定查询图q和数据图g，给出增量计算映射编辑距离的方法，以及得到的状态空间树. 查询图q有8个顶点，因此查询图q和数据图g的状态空间树共有8层. 对于第1层的映射$f_1=[u_1\to v_1]$，根据算法4可以得到映射f_1对应的映射编辑距离$fed_{f_1}(q_{f_1},g_{f_1})=0$，由公式(6)得到状态编辑距离$sed_{f_1}(q,g)=0$. 在第2层继续拓展$f_1$，得到了7个新的部分映射，对于$f_2=[u_1\to v_1,u_2\to v_2]$，根据算法4可以增量得到映射$f_2$对应的映射编辑距离$fed_{f_2}(q_{f_2},g_{f_2})=0$，由公式(6)得到状态编辑距离$sed_{f_2}(q,g)=2$. 重复上述过程，遍历到查询图$q$和数据图$g$的完整顶点映射，可以得到如图9所示$q$和$g$的一棵状态空间树.

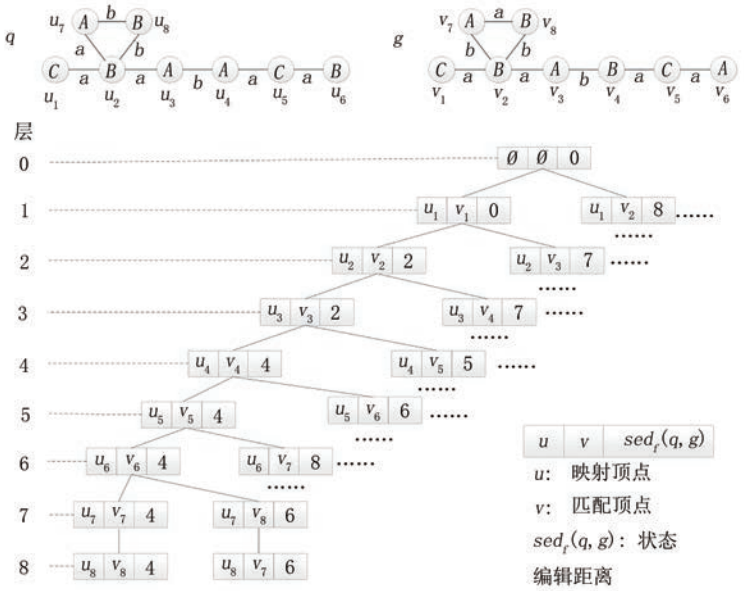


图9 q和g的状态空间树

4.3.2 增量验证策略

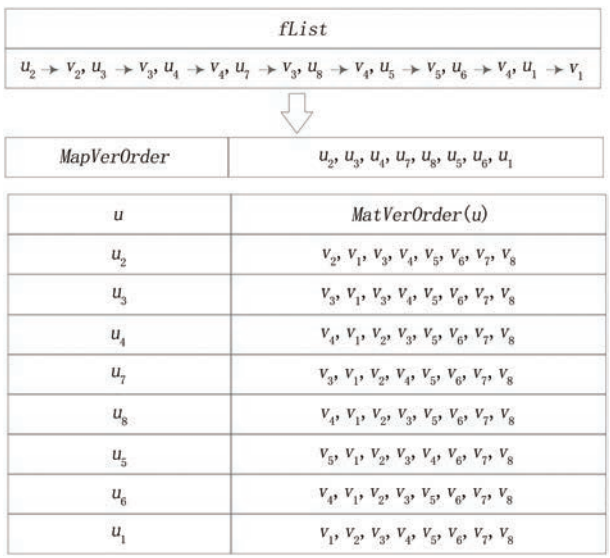
给定一个查询图q和数据图g，遍历q和g之间所有可能的顶点映射，构建状态空间树，计算所有完整映射对应的映射编辑距离，判断与阈值的大小关系，当某个状态编辑距离大于阈值，则不需要继续拓展，可以直接进行剪枝操作. 由于图集中大多数据图与查询图都不相似，建立状态空间树的过程中，越早出现编辑错误，先拓展的结点对应的状态编辑距离越大，越早出现剪枝操作，可以减少映射的数量. 因此，本小节提出增量策略，利用分区过滤阶段保留的映射结果，快速遍历到图编辑距离对应的映射. 并给出增量验证算法的具体过程.

增量策略：根据分区过滤阶段得到的映射列表*fList*，确定验证时顶点的映射顺序与*fList*中映射顺序相同，用变量*MapVerOrder*表示. 变量*MapVerOrder*中每个映射顶点u的匹配顶点顺序也与*fList*中顶点u的匹配顶点顺序保持相同，用变量*MatVerOrder*(u)表示.

如图10所示，给定查询图q和数据图g，当阈值为4时，利用分区过滤阶段保留的部分映射结果*fList*，确定查询图q映射顶点的顺序$MapVerOrder=[u_2, u_3, u_4, u_7, u_8, u_5, u_6, u_1]$，以及对于$q$中每一个顶点$u$对应的匹配顶点顺序*MatVerOrder*($u$).

4.3.3 增量验证算法描述

给定查询图q和数据图g，根据4.3.2节中给出的增量验证策略，提出增量验证算法. 下面介绍增量验证算法IncVer(q, g, τ, *fList*).



u	*MatVerOrder*(u)
u_2	$v_2, v_1, v_3, v_4, v_5, v_6, v_7, v_8$
u_3	$v_3, v_1, v_3, v_4, v_5, v_6, v_7, v_8$
u_4	$v_4, v_1, v_2, v_3, v_5, v_6, v_7, v_8$
u_7	$v_3, v_1, v_2, v_4, v_5, v_6, v_7, v_8$
u_8	$v_4, v_1, v_2, v_3, v_5, v_6, v_7, v_8$
u_5	$v_5, v_1, v_2, v_3, v_4, v_6, v_7, v_8$
u_6	$v_4, v_1, v_2, v_3, v_5, v_6, v_7, v_8$
u_1	$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$

图10 映射顶点和匹配顶点顺序

下面，给出具体增量验证算法的伪代码，如算法5所示.

算法5. 增量验证算法 IncVer(q, g, τ, *fList*)

输入：查询图q，数据图g，阈值τ和映射列表*fList*.

输出：相似返回true，否则返回false.

1. 初始化映射$f \leftarrow \varnothing$;
2. 根据*fList*确定变量*MapVerOrder*; //增量策略
3. RETURN Bac(q,g,τ,*fList*,*MapVerOrder*,f);//算法6

首先初始化映射f为空，即初始化状态空间树根节点为空，接着利用增量验证策略确定查询图q顶点的访问顺序*MapVerOrder*，最后使用回溯算法，拓展映射f进行验证.

下面，介绍回溯算法，建立状态空间树的过程中，当映射f对应的状态编辑距离满足阈值时，需要继续拓展f. 按照上述增量策略确定映射顶点顺序，使用回溯方法确定拓展映射f的算法Bac(q, g, τ, *fList*, *MapVerOrder*, f).

下面，给出具体的伪代码，如算法6所示.

算法 6.　回溯算法 Bac($q, g, \tau, fList, MapVerOrder, f$)

输入：查询图q，数据图g，阈值τ，映射列表*fList*，映射顶点顺序*MapVerOrder*和映射f.

输出：映射f是满映射返回true，否则false.

```
1. IF (f是q到g的完整映射)
2.     RETURN true;  //判定相似
3. ELSE
4.     从MapVerOrder得到映射顶点u，确定变量MatVerOrder(u);  //增量策略
5.     FOR (取MatVerOrder(u)中顶点v)
6.         f中加入f(u)=v，计算状态编辑距离sed_f(q,g);  //公式(6)
7.         IF(sed_f(q,g)≤τ)
8.             RETURN Bac(q,g,τ,fList,MapVerOrder,f);
9. RETURN false
```

第1行，判断f是否是一个完整映射，若是，执行第2行，若不是，继续拓展f，执行3-8行. 第2行，f是完整映射，表示已经建立到状态空间树的叶子结点，可以直接判定相似. 第4-8行，需要进行深度搜索，判定是否相似. 其中，第4行，从变量*MapVerOrder*中得到待映射顶点u，根据增量策略得到u的待匹配顶点顺序$MatVerOrder(u)$. 第5-8行，依次从$MatVerOrder(u)$中取顶点v，拓展映射f，得到新映射f，判断f对应的状态编辑距离是否满足阈值要求，若满足，则执行第8行，继续拓展f.

如图11所示，给定查询图q和数据图g，当阈值$\tau=4$时，未被过滤掉，需要增量验证. 初始化映射$f_1=\varnothing$，由图10所示映射列表，得到$MapVerOrder=[u_2, u_3, u_4, u_7, u_8, u_1, u_5, u_6]$，由于$f_1$不是完整映射，从*MapVerOrder*中得到待映射顶点u_2，由映射列表得到$MatVerOrder(u_2)=[v_2, v_1, v_3, v_4, v_5, v_6, v_7, v_8]$. 顶点$u_2$依次与$MatVerOrder(u_2)$中待匹配顶点做映射，首先得到映射$f_2=[u_2 \to v_2]$，对应的状态编辑距离$sed_{f_2}(q,g)=2$，由于$sed_{f_2}(q,g)\leq\tau$，继续拓展

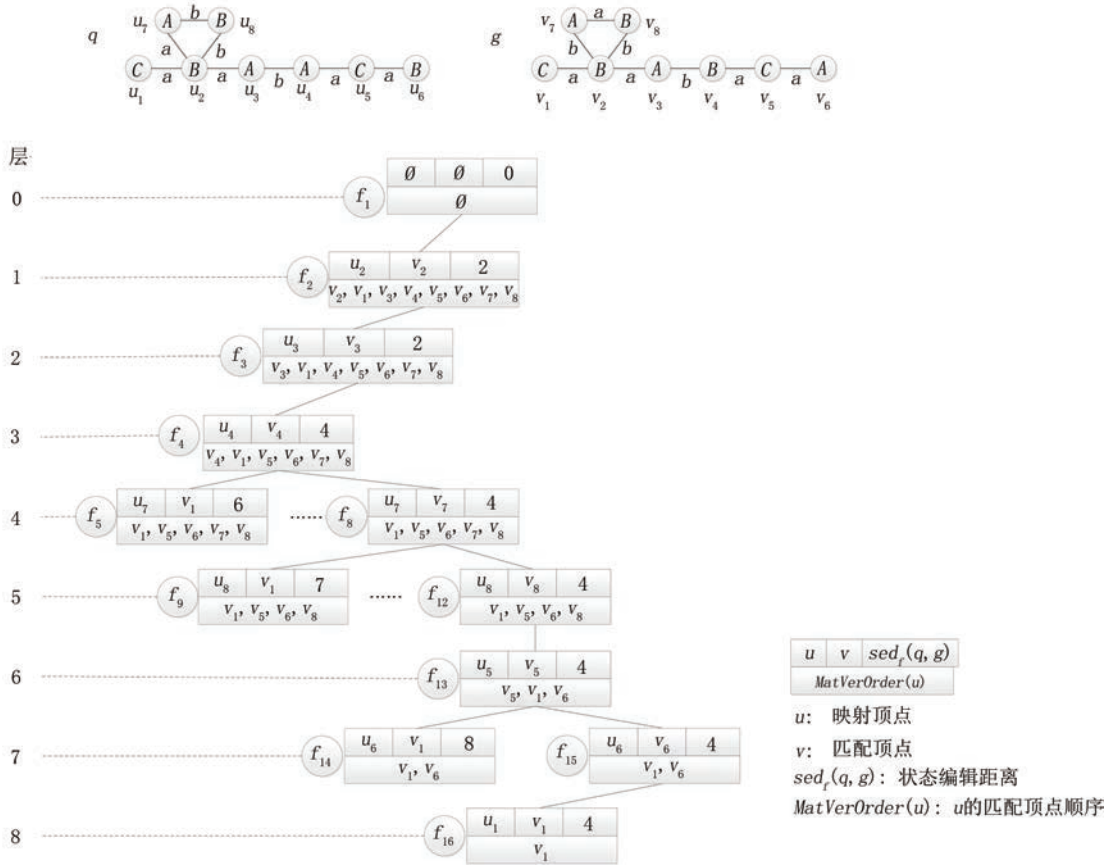


图11　增量建立状态空间树

映射f_2. 重复上述操作，直至映射$f_{16}=[u_2\rightarrow v_2, u_3\rightarrow v_3, u_4\rightarrow v_4, u_7\rightarrow v_7, u_8\rightarrow v_8, u_5\rightarrow v_5, u_6\rightarrow v_6, u_1\rightarrow v_1]$，对应的$sed_{f_{16}}(q,g)=4$，继续拓展，判断是完整映射，返回true，说明查询图q和数据图g在阈值$\tau=4$时相似.

5 实验分析

5.1 实验设置

在本文中，使用Visual C++语言实现了分区过滤-增量验证算法PFIV. 实验环境为Intel i7-12700H CPU@2.30 GHz；16 GB内存；512 GB硬盘和Windows 11操作系统.

在本文中，将算法PFIV与算法Inves[8]、算法AStar-LSa[9]和算法A*-CB[19]进行了验证比较，主要的衡量标准为查询时间. 算法Inves：采用分区的方法进行过滤，当每个分区出现编辑错误时，就停止分区，记录一个编辑错误. 对于未能过滤掉的数据图，采用文献中的A*-GED算法进行验证，其中从过滤阶段得到的顶点数目最多的分区对应的顶点开始映射. 算法AStar-LSa和算法A*-CB：由于AStar-LSa方法和A*-CB方法仅对验证阶段进行了改进，因此在过滤阶段采用算法Inves中提到的过滤方法进行过滤，在验证阶段分别使用AStar-LSa方法和A*-CB方法进行验证.

本文采用3个真实数据集进行验证. AIDS是一个在研究艾滋病过程中发布的抗病毒化合物数据集，包含42689种化合物(AIDS数据集的获取网址为https://cactus.nci.nih.gov/). PubChem是一个获取化合物信息的网站(网址为https://ftp.ncbi.nlm.nih.gov/pubchem/)，从众多化合物集合中抽取两个，PubChem_1和PubChem_2分别包含10 000个化合物集合. 3个数据集的统计数据如表2所示，给出了数据集中数据图数量$|D|$，平均顶点数Avg$|V|$、平均边数Avg$|E|$、顶点标签的种类数#vlabels和边标签的种类数#elabels.

下面，给出实验过程中的相关参数，参数的默认值和变化范围.

表2 实验数据集描述

数据集	$\|D\|$	$Avg\|V\|$	$Avg\|E\|$	#vlabels	#elabels
AIDS	42 689	25.60	27.52	68	3
PubChem_1	10 000	45.51	47.79	10	3
PubChem_2	10 000	62.87	66.10	47	3

表3 参数设置

参数	默认值	变化范围
阈值τ	3	1,2,3,4
查询图数目n	50	30,40,50,60

5.2 阈值变化对算法性能的影响

本节评估了阈值大小对算法性能的影响.

首先，为分析在不同阈值时算法的性能，给出数据集AIDS、数据集PubChem_1和数据集PubChem_2在查询图数目为50时，相似结果随阈值τ的变化情况，如表4所示. 随着阈值的增大，数据集AIDS和数据集PubChem_1对应的相似图数目明显增加，数据集PubChem_2对应的相似图数目变化不明显.

表4 不同阈值对应相似图数目

阈值τ	1	2	3	4
AIDS	55	84	135	211
PubChem_1	111	117	124	152
PubChem_2	52	54	54	54

接着，为展示不同数据集的相似情况，将相似图数目占数据图数目的比率称为相似比率. 给出数据集AIDS、数据集PubChem_1和数据集PubChem_2在查询图数目为50时，相似比率随阈值τ的变化情况，如表5所示. 随着阈值的增大，相似比率逐渐增大. 从表5可以发现当阈值相同时，数据集PubChem_1的相似比率要高于其它两个数据集.

表5 不同阈值对应的相似比率

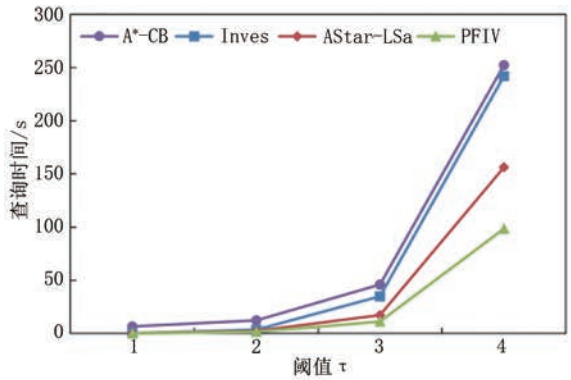
阈值τ	1	2	3	4
AIDS	0.0013	0.0020	0.0032	0.0049
PubChem_1	0.0111	0.0117	0.0124	0.0152
PubChem_2	0.0052	0.0054	0.0054	0.0054

其次，给出了阈值大小对过滤数目的影响. 将过滤结束后，需要继续在验证阶段判断的数据图称为待验证的数据图. 下面，给出数据集AIDS、数据集PubChem_1和数据集PubChem_2在查询图数目n为50时，不同阈值对应待验证的数据图数目，如表6所示. 随着阈值的增大，待验证的数据图数目在增大，相似图集总数目占待验证的数据图数目的比率在降低，过滤效果变差. 但本文所提出过滤算法使得待验证的数据图数目一直最小，过滤效果一直最优. 这是因为本文算法考虑到了桥标签集贡献的编辑距离，使得过滤下界变大，过滤效果更好.

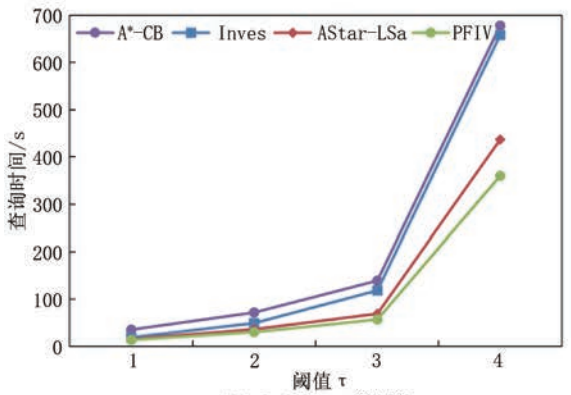
表6　不同阈值对应待验证的数据图数目

	阈值	1	2	3	4
AIDS	Inves	105	318	3235	16 744
	PFIV	81	246	1808	12 252
PubChem_1	Inves	122	203	575	985
	PFIV	122	163	249	700
PubChem_2	Inves	56	98	157	413
	PFIV	52	59	80	165

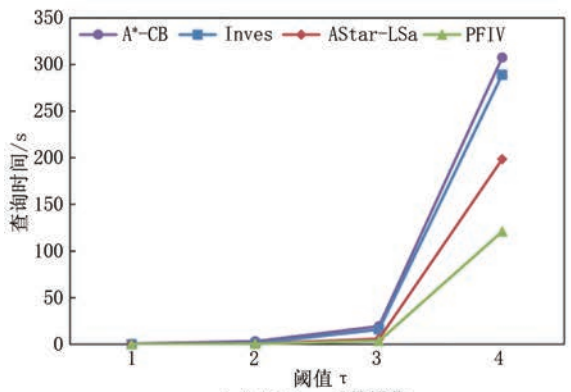
接着，对比了查询图数目n为50时，阈值的变化对查询时间的影响，如图12所示．图12中(a)、(b)和(c)分别是数据集AIDS、数据集PubChem_1和数据集PubChem_2对应的查询时间．从图12可以观察到随着阈值τ的增大，三个数据集的查询时间都在变长．这是因为阈值变大，导致过滤下界的计算时间变长．且阈值越大，相似图数目越多，未能过滤掉的数据图数目也越多，验证时间越长．算法A*-CB查询时间最长，算法PFIV的查询时间最短，这是因为算法A*-CB在算法A*-GED的基础上，仅减少了内存占用，未缩减查询时间．算法PFIV相较于算法Inves，考虑到了分区时顶点的映射顺序和桥标签集可能提供的编辑错误，使得过滤下界增大，分区速度加快，且采用了增量验证的方法，使得验证时间变短．算法PFIV相较于算法AStar-LSa，保留了过滤阶段的部分映射结果，采用增量方法进行验证，更快找到满足阈值要求的编辑路径．对于不同数据集，在阈值为1时，算法PFIV比算法AStar-LSa的查询时间缩短了8%，在阈值为2时，算法PFIV比算法AStar-LSa的查询时间缩短了15%，在阈值为3和4时，算法PFIV比AStar-LSa的查询时间缩短了17%．同时，对比图12中三个数据集，可以观察到数据集PubChem_1对应的查询时间较长，这是因为数据集PubChem_1的相似比率较高，说明对于较相似的图集，出现剪枝操作较晚，较难发现满足阈值要求的编辑距离，导致查询效果较差．实验表明，本文所提出的算法PFIV能够更加高效处理图编辑相似查询问题．



(a) AIDS数据集



(b) PubChem_1数据集



(c) PubChem_2数据集

图12　阈值对查询时间的影响

最后，对比了查询图数目n为50时，阈值的变化对内存大小的影响，如图13所示．图13中(a)、(b)和(c)分别是数据集AIDS、数据集PubChem_1和数据集PubChem_2对应的内存大小．由图13可以观察到算法PFIV比算法Inves和A*-CB占用内存少2%-11%，但比算法AStar-LSA占用内存多4%-9%．这是因为建立状态空间树时，算法Inves和A*-CB都直接建立，而算法PFIV和AStar-LSA都考虑了通过避免虚拟顶点的设置减少映射数量，减少内存占用．但算法PFIV比AStar-LSA多保留了映射结果，内存占用增加．考虑到内存差异不大，但本文所提出的算法PFIV在时间优于其它算法，说明算法PFIV优于其它算法．

5.3　查询图数目变化对算法性能的影响

本节评估了查询图数目对算法性能的影响．

首先，为分析在不同查询图数目时算法的性能，给出数据集AIDS、数据集PubChem_1和数据集PubChem_2在阈值τ为3时，相似图的总数目随查询图数目的变化情况，如表7所示．从表7可以观察到随着查询图数目的增大，三个数据集对应的相似

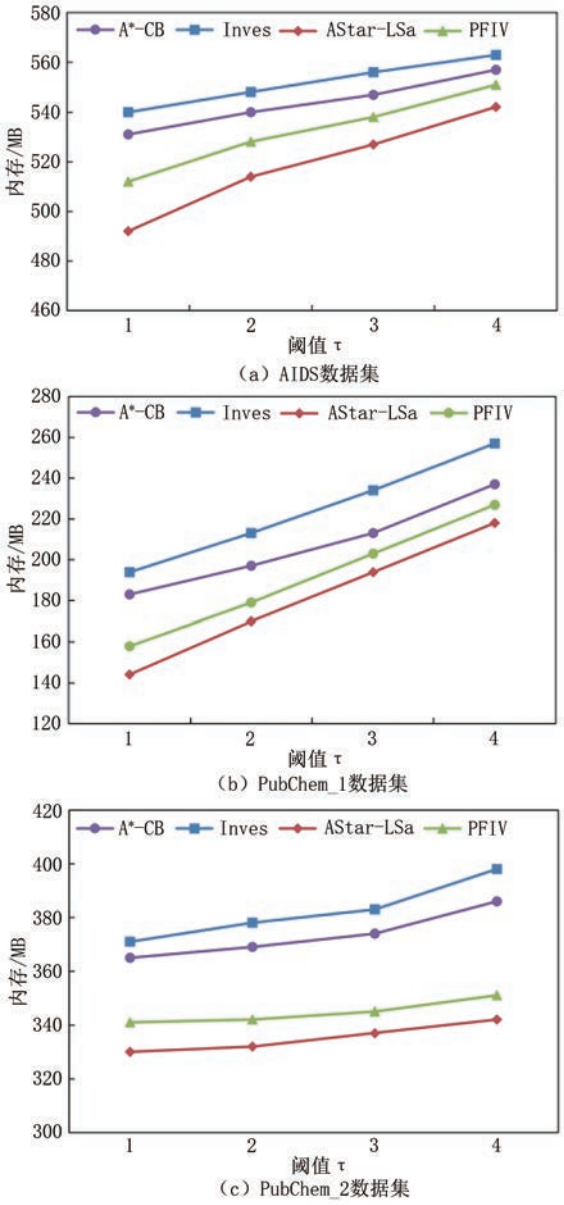


图13 阈值对内存的影响

表7 不同查询图数目对应相似图的总数目

查询图数目n	30	40	50	60
AIDS	94	111	135	173
PubChem_1	60	84	124	144
PubChem_2	32	44	54	67

图的总数目都在逐渐增多.

其次,给出了查询图数目大小对过滤数目的影响.下面,给出数据集AIDS、数据集PubChem_1和数据集PubChem_2在阈值τ为3时,不同查询图数目对应待验证的数据图数目,如表8所示.随着查询图数目的增大,未被过滤掉的总数据图数目在增大,验证阶段需要进一步验证的数据图数目增大.但本文所提出分区过滤算法使得待验证的数据图数目一直最少,过滤效果一直最优.这是因为虽然查询图数目增大,但是本文所提出算法对每个查询图的过滤都是有效的,因此总待验证的数据图数目最少.由此可知,本文所提出基于分区的过滤算法优于Inves的过滤算法,从而有效证明了本文所提出分区过滤算法的有效性.

表8 不同查询图数目对应待验证的数据图数目

	查询图数目n	30	40	50	60
AIDS	Inves	1942	2573	3235	3705
	PFIV	1088	1356	1808	2707
PubChem_1	Inves	369	476	575	734
	PFIV	134	174	249	306
PubChem_2	Inves	88	116	157	198
	PFIV	57	69	80	98

接着,对比了数据集AIDS、数据集PubChem_1和数据集PubChem_2在阈值为3时,查询图数目的变化对查询时间的影响,如图14所示.从图14可以观察到随着查询图数目的增大,三种算法对应的查询时间都在增大.但本文所提出的算法PFIV对应的查询时间一直是最短的,这得益于提出的分区的思想,提高了过滤效果和验证效果,整体的查询时间变短,由此说明算法PFIV在查询效率上优于其他三种算法.

最后,对比了数据集AIDS、数据集PubChem_1和数据集PubChem_2在阈值为3时,查询图数目对内存的影响,如图15所示.图15中(a)、(b)和(c)分别是数据集AIDS、数据集PubChem_1和数据集PubChem_2对应的内存大小.由图15可以观察到随着查询数目的增加,内存占用随之增大.算法PFIV的内存小于算法Inves和A*-CB,大于算法AStar-LSA的内存.考虑到算法PFIV和算法AStar-LSA的内存差异不大,但本文所提出的算法PFIV在时间优于其它算法,说明算法PFIV优于其它算法.

5.4 策略对算法性能的影响

5.4.1 桥标签集对查询时间的影响

在本节评估了桥标签集对算法性能的影响,设置PFIV桥标签集为分区过滤和增量验证阶段不考虑桥边贡献的编辑错误,如图16所示.随着阈值的增大,查询时间都在增长,但当不考虑桥边贡献的编辑错误时,查询时间更长.这是由于分区

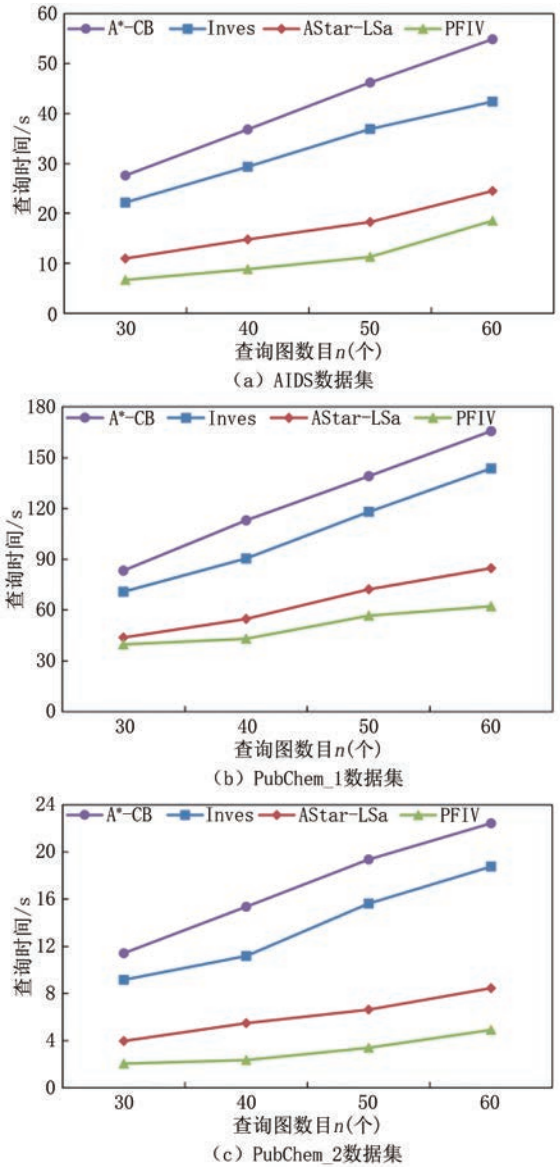


图14　查询图数目对查询时间的影响

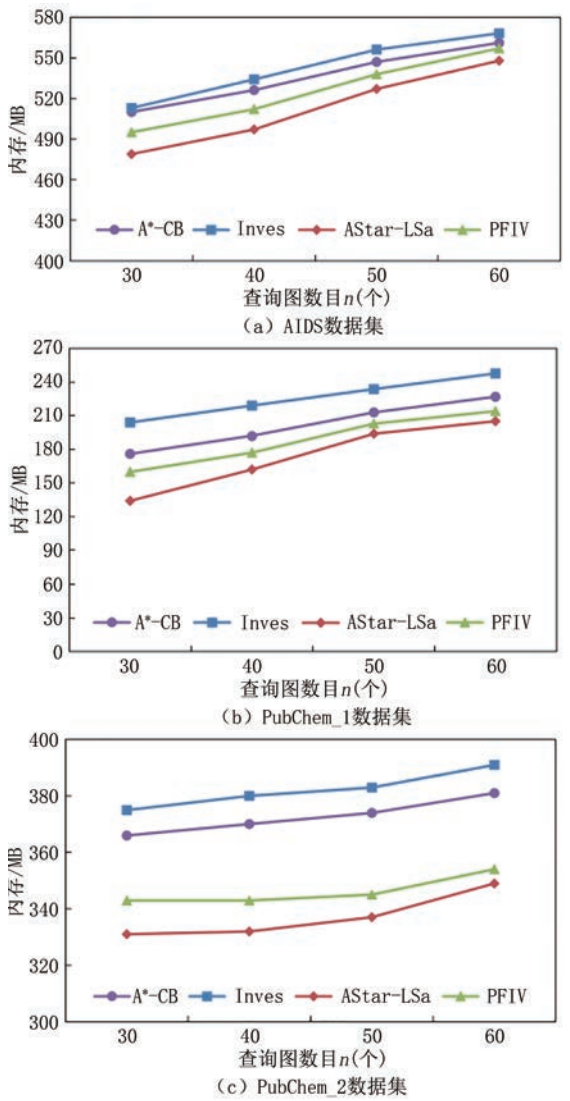


图15　查询图数目对内存的影响

编辑距离下界和状态编辑距离都考虑了桥的标签集可能贡献的编辑距离，提高了分区编辑距离下界和状态编辑距离的值，更快过滤和剪枝掉不相似的数据图.

5.4.2　映射顶点顺序策略对查询时间的影响

本节评估了映射顶点顺序策略对算法性能的影响. 设置PFIV映射顶点顺序算法为分区过滤阶段顶点的映射顺序不采用本文提出的映射顶点顺序策略，而是按照顶点下标排序，两种算法的查询时间如图17所示.

随着阈值的增大，无论是否使用映射顶点顺序策略，查询时间都在增长，但使用映射顶点顺序策略比不使用映射顶点顺序策略查询时间短. 因为按照映射顶点顺序策略分区，可以优先映射边数较多的顶点，这些顶点发生编辑错误的概率更大，过滤速度更快. 如果未被过滤且不相似，也可以在验证阶段尽早回溯，更快判断不相似.

5.4.3　分区结束条件策略对查询时间的影响

本节评估了分区结束条件策略对算法性能的影响. 设置PFIV分区结束条件为不使用分区结束条件策略，即直接验证，两种算法的查询时间如图18所示.

随着阈值的增大，无论是否使用分区结束条件策略，查询时间都在增长，但是使用分区结束条件策略明显比不使用分区结束条件策略查询时间短. 因为不使用分区结束条件策略，相当于没有过滤阶段，直接使用A*-GED算法进行验证，验证数目增加，使得查询时间增长.

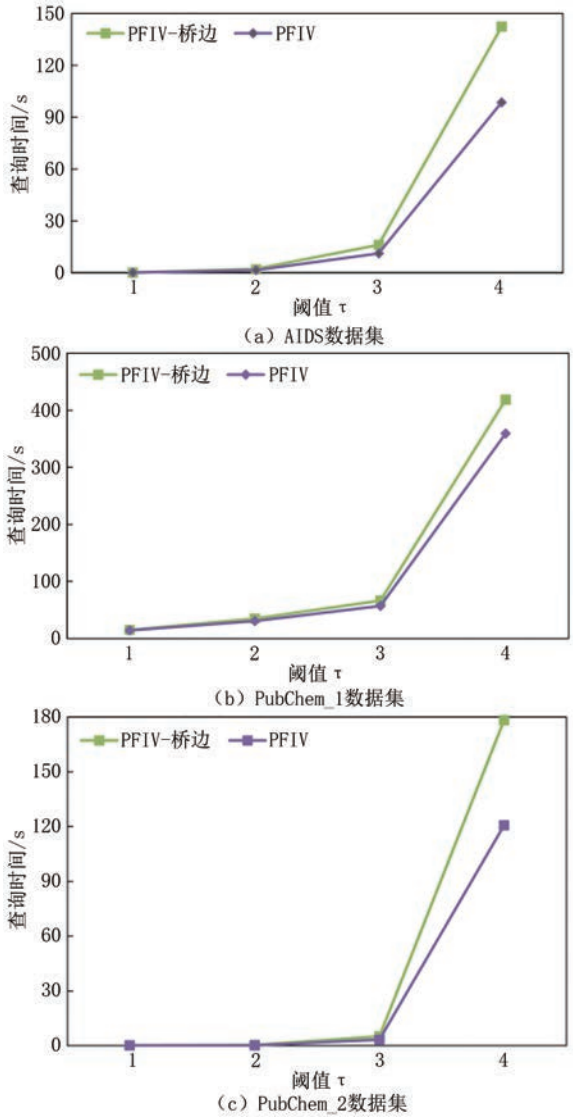


图16 桥边对查询时间的影响

5.4.4 增量策略对查询时间的影响

在本节中测试了增量策略对算法性能的影响，设置PFIV增量算法为在验证过程中不使用增量的策略，在验证阶段顶点的映射顺序和匹配顺序按照顶点下标排列，查询时间如图19所示. 随着阈值的增大，使用增量策略会使查询时间相对较小. 增量策略对AIDS数据集和PubChem_1数据集的查询时间影响较大，这是因为在分区过滤阶段有较多数据图没有被过滤掉，仍需要在验证阶段进行判断.

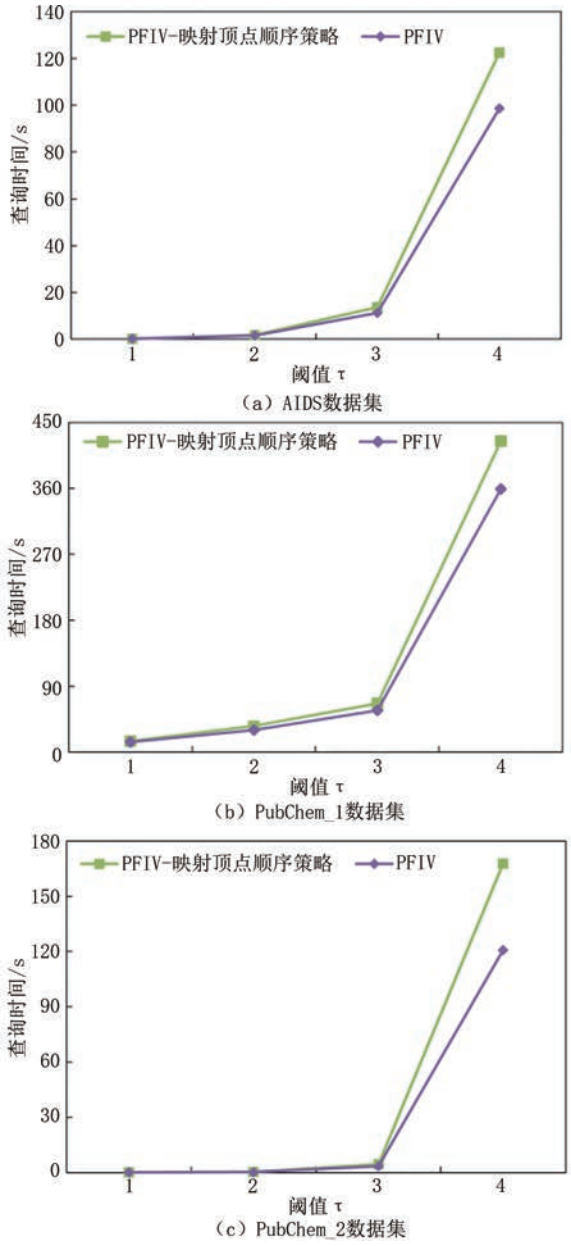
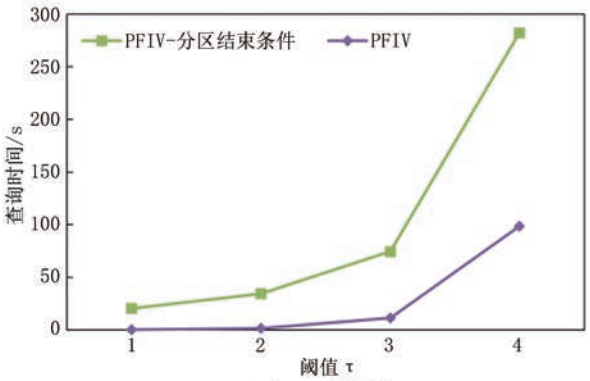


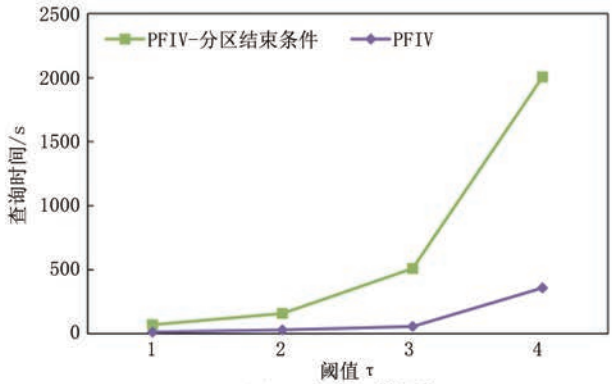
图17 映射顶点顺序策略对查询时间的影响

6 结 论

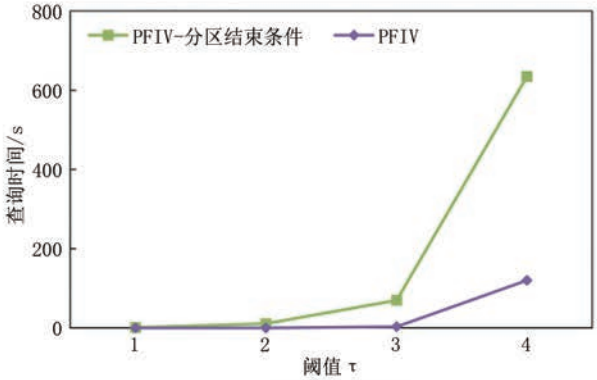
本文采用分区过滤-增量验证算法PFIV对图编辑相似查询问题进行了深入研究. 首先，针对过滤阶段，提出分区的思想计算过滤下界，并提出了两种策略：提高过滤速度，增强过滤效果. 其次，对未能过滤掉的数据图，利用分区过滤阶段保留的映射结果，加快验证速度，提高计算效率. 最后，在三个真实数据集上，对比算法PFIV和目前先进的图编辑相似查询算法如AStar-LSa在不同阈值时的查询时间. 在阈值为1时，查询时间减少8%，阈值在2到4时，查询时间减少15%～17%. 说明算法PFIV能够有效处理图编辑相似查询问题，提升查询效率.



（a）AIDS数据集

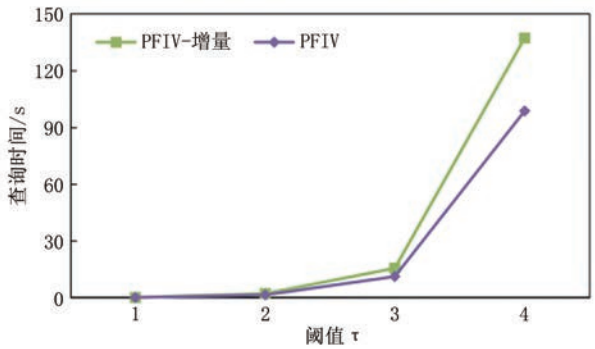


（b）PubChem_1数据集

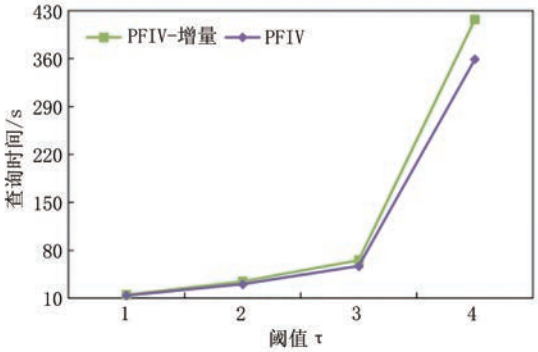


（c）PubChem_2数据集

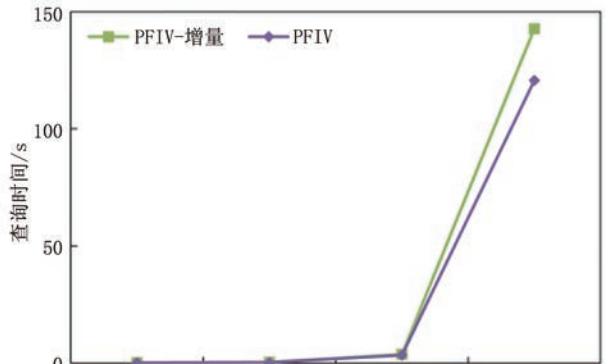
图18　分区结束条件策略对查询时间的影响



（a）AIDS数据集



（b）PubChem_1数据集



（c）PubChem_2数据集

图19　增量策略对查询时间的影响

参 考 文 献

[1] Marin R M, Aguirre N F, Daza E E. Graph theoretical similarity approach to compare molecular electrostatic potentials. Cheminform, 2008, 39(18): 109-118

[2] Bunke H, Shearer K. A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters, 1998, 19(3-4): 255-259

[3] Abu-khzam F N. Maximum common induced subgraph parameterized by vertex cover. Information Processing Letters. 2014, 114(3): 99-103

[4] Nirmala P, Lekshmi R.S., Nadarajan R.. Vertex cover-based binary tree algorithm to detect all maximum common induced subgraphs in large communication networks. Knowledge and Information Systems. 2016, 48(1): 229-252

[5] Quer S, Marcelli A, Squillero G. The maximum common subgraph problem: a parallel and multi-engine approach. Computation, 2020, 8(2): 48

[6] Shang H, Lin X, Zhang Y, et al. Connected substructure similarity search//Proceedings of the ACM SIGMOD International Conference on Management of Data. Indiana, USA, 2010: 903-914

[7] Zhu Y, Qin L, Yu J X, et al. Answering top-*k* graph similarity queries in graph databases. IEEE Transactions on Knowledge and Data Engineering, 2020, 32(8): 1459-1474

[8] Kim J, Choi D, Li C. Inves: Incremental partitioning-based verification for graph similarity search//Proceedings of the 22nd International Conference on Extending Database Technology. Lisbon, Portugal, 2019: 229-240

[9] Chang L, Feng X, Lin X, et al. Speeding up GED verification for graph similarity search//Proceedings of the IEEE International Conference on Data Engineering. Dallas, USA, 2020: 793-804

[10] Zeng Z, Tung A K H, Wang J, et al. Comparing stars: On approximating graph edit distance. Proceedings of the VLDB Endowment, 2009, 2(1): 25-36

[11] Riesen K, Fankhauser S, Bunke H. Speeding up graph edit distance computation with a bipartite heuristic//Proceedings of the Mining and Learning with Graphs. Firence, Italy, 2007: 1-4

[12] Sanfeliu A, Fu K-S. A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics, 1983, SMC-13(3): 353-362

[13] Zhao X, Xiao C, Lin X, et al. Efficient graph similarity joins with edit distance constraints//Proceedings of the IEEE International Conference on Data Engineering. Washington, USA, 2012: 834-845

[14] Wang G, Wang B, Yang X, et al. Efficiently indexing large sparse graphs for similarity search. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(3): 440-451

[15] Zheng W, Zou L, Lian X, et al. Efficient graph similarity search over large graph databases. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(4): 964-978

[16] Liang Y, Zhao P. Similarity search in graph databases: A multi-layered indexing approach//Proceedings of the IEEE International Conference on Data Engineering. San Diego, USA, 2017: 783-794

[17] Zhao X, Xiao C, Lin X, et al. A partition-based approach to structure similarity search. Proceedings of the VLDB Endowment, 2013, 7(3): 169-180

[18] Gouda K, Hassaan M. CSI_GED: An efficient approach for graph edit similarity computation//Proceedings of the International Conference on Data Engineering. Helsinki, Finland, 2016: 265-276

[19] Chegrane I., Hocine I., Yahiaoui S., et al. Graph edit distance compacted search tree//Proceedings of the International Conference on Similarity Search and Applications. Bologna, Italy, 2022: 181-189

[20] Zhang S, Yang J, Jin W. SAPPER: Subgraph indexing and approximate matching in large graphs. Proceedings of the VLDB Endowment, 2010, 3(1): 1185-1194

[21] Zhang W, Lin X, Zhang Y, et al. Efficient probabilistic supergraph search. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(4): 965-978

[22] Yamada M, Inokuchi A. similar supergraph search based on graph edit distance. Algorithms, 2021, 14(8): 1-25

WANG Xi-Te, Ph. D., associate professor. His main research areas include big data management and outlier detection.

BAI Mei, Ph.D., associate professor. Her main research areas include data management, cloud computing and query processing.

WANG Chao-Jin, M. S. Her research area is query processing.

MA Qian, Ph.D., associate professor. Her main research areas include data management, data cleaning.

LI Guan-Yu, Ph.D., professor. His main research areas include intelligent information processing, Internet of Things, semantic web.

Background

Not only the vertices and edges in a graph can carry feature information, but also the connection relationship between the vertices can represent structural information. The graph query problem refers to finding out the data graph that is the same as the specified query graph, but the existence of data inconsistency and natural noise often exists in the graph query process, so the graph query problem is often converted into the graph similar query problem. The graph similar query problem refers to querying all data graphs in the graph database that are similar to the given query graph. The graph edit distance is the measure that best captures the structural and feature differences between graphs, not only for graphs with vertices and edges without labels, but also for graphs with vertices and edges with labels. Therefore, graph edit distance becomes the most commonly used measure for graph similarity queries, referred to as graph edit similarity queries.

However, graph edit distance calculation is an NP-Hard problem. In order to reduce the computation time, currently, most of the studies for graph edit similar queries use the filtering-validation framework. In the filtering phase, different methods are designed to get the filtering lower bound and filter out some of the data graphs that must not be similar. In the validation phase, most of them directly use the A*-GED algorithm to construct a state space tree, search all possible mappings of vertices between the query graph and the data graph, and calculate the graph edit distance. However, due to the low filtering lower bound, there are still a lot of data graphs to be verified, resulting in a long query time. The current filtering stage uses only the filtering lower bound for filtering, which has the problem of poor scalability and does not effectively use the intermediate results generated by filtering.

Compared with the previous algorithms, the algorithm in this paper calculates the filtering lower bound by a partitioning method to enhance the filtering effect. At the same time, some of the mapping results of the filtering stage are retained and an incremental approach is taken to quickly verify whether they are similar. Therefore, the algorithm proposed in this paper can solve the graph editor similarity query problem more effectively.

This research was partially supported by the National Natural Science Foundation of China under Grant No. 62002039, 61602076, 61702072, 6197603 and the Fundamental Research Funds for the Central Universities No. 3132023259.