面向大规模图数据的分布式子图匹配算法

许 文 宋文爱 富丽贞 吕 伟
(中北大学软件学院 太原 030051)

摘 要 图数据规模的爆发式增长使在单机上的子图匹配变得较为困难。尽管现有的分布式算法可以在一定程度上解决大规模图数据的子图匹配问题,但分布式环境中的网络通信代价仍然影响着算法的性能。为此,文中提出了DSsearch分布式子图匹配算法,包含查询图拆分、数据图预处理、候选顶点过滤、中间结果合并4个步骤。其中,在数据图预处理步骤中使用图划分和完善邻居顶点策略来降低匹配过程中分布式计算节点之间的通信代价;在过滤候选顶点阶段设计DSgraph存储结构存储候选顶点,通过推迟笛卡尔积来减少冗余的中间结果。最后设计了对比实验并在具有7个计算节点的Spark分布式集群上使用真实数据集进行验证。实验结果表明,DSsearch算法能够在秒级时间内完成对百万规模顶点的数据图的子图匹配,尤其是在处理复杂查询图和稠密数据图方面更高效。数据图预处理策略的实验结果说明了通过顶点复制来降低分布式环境中网络通信代价这一策略的可行性。相比TwinTwigJoin、PSgL等算法,随着查询图顶点数量的增加,DSsearch算法的运行时间增长得更缓慢,当查询图顶点数量达到14时,其运行时间是TwinTwigJoin和PSgL算法的一半。实验数据充分说明,分布式环境中的网络通信代价和中间结果数量是影响分布式子图匹配算法的主要因素。实现数据图的预处理和推迟笛卡尔积解决了分布式子图匹配的性能瓶颈问题,有效地完成了大规模图数据的子图匹配。

关键词 子图匹配,子图查询,分布式,图数据,图划分
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.04.005

Distributed Subgraph Matching Algorithm for Large Scale Graph Data

XU Wen SONG Wen-ai FU Li-zhen LV Wei
(School of Software,North University of China,Taiyuan 030051,China)

Abstract The explosive growth of graph data size makes it difficult to process subgraph matching on single machine. Although the existing distributed algorithms could solve the problem to some extent, the network communication cost among the distributed computing nodes still affects the performance of algorithm. To solve this problem, this paper proposed a distributed subgraph matching algorithm, named DSsearch. It includes four steps: spliting query graph, preprocessing data graph, filtering candidate and combining intermediate results. In the data graph preprocessing, the strategy of graph partitioning and perfect neighbor vertex are used to reduce the communication cost among the distributed computing nodes. The DSgraph storage structure is designed to store candidate vertices during the filtering candidate vertex stage, and the redundant intermediate results are reduced by delaying the Cartesian product. Finally, a comparative experiment was designed and real data sets verification was performed on a Spark distributed cluster with 7 compute nodes. The experimental results show that the DSsearch algorithm can complete subgraph matching of data graphs of millions of vertices in seconds, especially in dealing with complex query graphs and dense data graphs. The experimental results of the data graph preprocessing strategy illustrate the feasibility of reducing network communication cost in a distributed environment through vertex replication. Compared with other algorithms such as TwinTwigJoin and PSgL, the increase of running time of DSsearch algorithm becomes slowly as the number of vertices in the query graph increases. When the number of query graph vertices reaches 14, the running time is only half of that of TwinTwigJoin and PSgL algorithms. The experimental data fully demonstrates that the data transmission cost and the number of intermediate results in the distributed system are the main factors affecting the efficiency of distributed subgraph matching algorithm. Realization of preprocessing the data graph and delaying the Cartesian product solves the bottleneck problem of the performance of distributed subgraph matching and effectively completes the subgraph matching of large-scale graph data.

Keywords Subgraph matching, Subgraph query, Distributed, Graph data, Graph partition

---

到稿日期:2018-09-27 返修日期:2018-12-03 本文受国家自然科学基金(61602427),山西省自然科学基金(201601D202037)资助。

许 文(1993—),男,硕士生,主要研究方向为云计算与大数据,E-mail:18735181961@163.com;**宋文爱**(1964—),女,博士后,教授,主要研究方向为云计算与大数据、图形图像处理,E-mail:188425686@qq.com(通信作者);**富丽贞**(1982—),女,博士生,主要研究方向为云计算与大数据、图数据管理;**吕 伟**(1992—),男,硕士生,主要研究方向为云计算与大数据。

1　引言

图作为一种重要的数据结构,非常适合表达具有内在关联性的数据。子图匹配作为实现图数据上高效查询的基本操作,也被广泛应用于各个领域的实际问题中。例如在社会安全领域,可以通过子图匹配技术在大规模人员信息图中对可疑行为和人员进行实时监控。给定一个数据图 G 和一个查询图 Q,子图查询的目标是从图 G 中找出与图 Q 同构的所有子图。由于子图匹配是一个 NP 完全问题,因此很难找到高效的子图查询处理算法。随着图数据规模的急剧扩增,大规模图数据上的子图匹配成为了学术界和工业界面临的新挑战。

目前存在的子图匹配算法主要分为基于索引算法和无索引算法两种。无索引算法是将查询图中顶点的属性以及顶点的局部结构等约束条件作为剪枝规则,对数据图中的数据顶点过滤得到匹配结果,代表性算法有 Ullamnn[1] 及其衍生算法[2]。这一类算法是通过构造一颗搜索树来得到最终的查询结果[3],只适用于小规模数据的子图匹配。$\text{Turbo}_{\text{ISO}}$[4] 算法提出候选区域探索,以解决子图匹配中匹配顺序的问题。文献[5]提出了 CPI 算法,CPI 算法是目前单机上最高效的子图匹配算法,该算法采用 CLF 分解查询图和 CPI 辅助数据结构避免了过早进行笛卡尔积而产生的无效中间结果,大大提高了算法的效率。

基于索引的算法主要通过使用图的结构特征建立索引、过滤、验证来得到匹配结果,其中建立索引为关键步骤。通过建立索引可以快速过滤掉无效匹配结果,但是当数据量增大时,建立索引会带来很大的时间和空间代价。因此这一类算法也不适用于大规模图数据上的子图匹配。代表性的算法有 GraphGrep[6] 采用的基于路径特征的索引技术。Gcoding[7] 算法为每个顶点构建 LNPT 树来表示顶点的局部结构特征,并设计了 Gcode-Tree 索引结构来提高查询效率。Closure-tree[8] 算法通过构建 C-tree 结构,利用树的分层结构来建立索引。

尽管有许多高效的算法被提出,但随着图规模的增长,上述提到的单机子图匹配算法已经无法适用,为了更好地解决大规模图数据的子图匹配问题,许多分布式的子图匹配算法被提出。STwig[9] 是最早被提出的一种分布式的子图匹配方法,但只能用于 Trinity[10] 平台,不适用于其他分布式平台。SAHAD[11] 算法基于 color coding[12] 在 Hadoop 集群中使用 MapReduce[13] 计算框架对子图匹配进行并行处理。TwinTwigJoin[14] 算法将查询图拆分为一系列的星状结构的子查询,利用 MapReduce 的合并操作来合并子查询结果,迭代执行直到得到最终匹配结果。该算法为大规模图数据的子图匹配问题提供了解决思路,但是在多次迭代的过程中产生了大量的中间结果,中间结果的读取和存储增大了时间和空间代价,使得算法并不高效。文献[15]提出的 PSgL 算法利用广度优先搜索策略通过对数据图的遍历来完成子图匹配,每次获取已经匹配但并未完全展开的顶点 v,搜索其相邻顶点的匹配以生成更细粒度的结果。该算法避免了大量中间结果的产生,但在搜索过程中会带来大量的数据传输代价。文献[16]提出 Iterative Vertex Elimination(以下简称为 IVE),其迭代地从数据图中删除不符合查询约束的顶点和边来减少查询空间,尽管该算法可以较好地解决大规模图数据子图匹配的问题,但每次匹配都需要对整个数据图的所有顶点进行遍历,且并未考虑到分布式集群中各计算节点之间的通信代价。文献[17]提出了基于图划分的 k-hop 复制思想,为降低分布式集群计算节点之间的通信代价提供了解决思路。

经过分析,现有的分布式子图匹配算法主要存在两方面的问题:1)匹配过程中会产生大量的中间结果;2)匹配过程中给分布式环境各计算节点之间带来了巨大的数据传输代价。针对这两个问题,本文基于 k-hop 复制的思想提出了一个分布式子图匹配算法 DSsearch(Distributed Subgraph search),该算法包括查询图拆分、数据图预处理、候选顶点过滤、中间结果合并 4 个步骤。在数据图预处理阶段,利用 1-hop 复制降低数据传输代价,在过滤候选顶点阶段设计 DSgraph 存储结构减少中间结果的数量。

本文的主要贡献如下:1)将查询图拆分为一系列的子查询图,并针对拆分后的子查询图设计数据图预处理方案,保证了匹配过程中图的完整性,降低了各计算节点间数据传输的代价;2)设计了 DSgraph 存储结构,推迟了笛卡尔积的产生,避免产生大量的中间结果;3)设计并实现了 DSsearch 算法并与典型算法 PSgL、TwinTwigJoin 以及当前最新算法 IVE 进行了分析比较。实验结果表明,所提算法具有高效性和可扩展性,尤其是在稠密图上的子图匹配效率更高。

2　相关概念

2.1　子图匹配

定义 1(有向图)　$G=(V_G,E_G,L_G,f_G)$,其中 V_G 表示有向图 G 的顶点集合;E_G 表示 G 中所有的有向边的集合;L_G 表示 V_G 的标签映射集合;f_G 表示 $V_G\rightarrow L_G$ 的标签函数;对 $v\in V_G$ 赋予标签集合 L_G 中的一个标签。

定义 2(查询图)　$Q=(V_Q,E_Q,L_Q,f_Q)$,其中 V_Q 表示查询图 Q 的顶点集合;E_Q 表示查询图 Q 的边的集合;L_Q,f_Q 与 L_G,f_G 的表示方式相同。

定义 3(子图同构)　给定图 $g=(V_g,E_g,L_g,f_g)$ 和 $G=(V_G,E_G,L_G,f_G)$,当且仅当在 g 到 G 之间存在一个单射函数 M,其中 M 满足条件 $\forall u\in V_g,L_g(u)=L_G(M(u))$,$\forall(u,u')\in E_g,(M(u),M(u'))\in E_G$ 时,称 g 为 G 的子图同构,这里 $M(u)$ 指 u 的映射顶点。

定义 4(子图匹配)　对于数据图 G 和给定的查询图 Q,子图匹配就是查询出所有 Q 在 G 中的子图同构映射。图 1(b)所示的数据图 G 中有两个与图 1(a)所示查询图 Q 匹配的子图,分别为$\{(A,a_2),(B,b_2),(C,c_2),(D,d_1)\}$,$\{(A,a_2),(B,b_2),(C,c_2),(D,d_3)\}$。

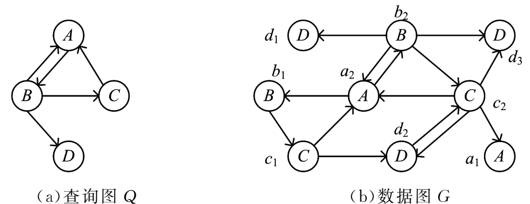


图 1　子图匹配

Fig. 1　Subgraph matching

2.2　图划分

定义 5(图划分)　图划分又称为图分区,给定图 $G=(V_G,E_G,L_G,f_G)$,正整数 k,将顶点 V_G 划分为互不相交的 k 个集合 $V_1,V_2,\cdots,V_k$,分别对应子图 $G_1,G_2,\cdots,G_k$。两个端

点不在同一个集合中的边被称为交互边。

在图划分时需要满足如下两个原则：

1)子图与子图的规模相差不大，即对于每个 V_i，$|V_i| \approx \frac{|V|}{k}$，$i=1,\cdots,k$。

2)在负载均衡的条件下使切分代价最小化，即交互边的总数最少。

3 分布式子图匹配

为了在分布式环境下解决大规模图数据的子图匹配问题，本文提出了解决方案，主要分为 4 个处理过程：1)查询图拆分；2)数据图预处理；3)候选顶点过滤；4)中间结果合并。

3.1 查询图拆分

DSsearch 算法提出的查询图拆分策略是将给定的有向查询图 Q 分解为一系列子查询，每个子查询 Q_i 为一个两层树结构。从查询图 Q 中找出所有入度大于 0 且入度与出度之和大于 1 的查询顶点作为连接类型的顶点，记作 V_j，这些连接顶点可以把所有的子查询连接成一个整体。在拆分时把 Q 中任意一个出度不为 0 的查询顶点 v 作为树的根节点，以 v 为起始点的边指向的所有查询顶点作为树的叶子节点，共同组成一个子查询 Q_i。查询图拆分的伪代码如算法 1 所示。

算法 1 decompose

输入：一个查询图 Q

输出：子查询集合 S_Q

```
1. v←getNext(Q);
2. while v is not null
3.    SQ←SQ∪Qi//Qi 表示以 v 为根的子查询
4.    mark v is visited;
5.    v←getNext(Q);
6. return SQ
```

算法 1 中 getNext(Q)从给定查询图 Q 中找出一个出度不为 0 且未被访问过的顶点，如果没有则返回 null。第 3－5 行将以 v 为根节点的两层树结构作为子查询 Q_i 添加到子查询集合 S_Q 中，并标记 v 为访问状态，然后从 Q 中重新找到一个出度不为 0 且未被访问过的顶点赋值给 v。迭代执行直到 v 为 null，则说明查询图已经拆分完毕。拆分过程如图 2 所示，将图 2(a)中查询图 Q 拆分为图 2(b)中 Q_1，Q_2，Q_3 3 个子查询。

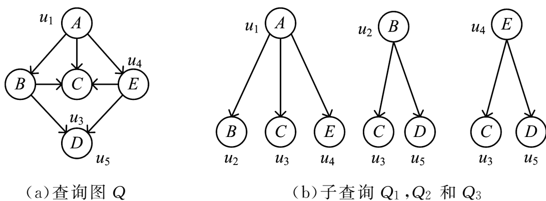


图 2 查询图的拆分过程

Fig. 2 Split process of query graph

由于在拆分查询图 Q 时，每次循环都从 Q 中找出一个出度大于 0 且未被访问过的查询顶点 u 作为子查询的根顶点，在最坏情况下，查询图 Q 中的所有查询顶点的出度都大于 0，因此，拆分后的子查询数量最大为 $|V(Q)|$。

3.2 数据图预处理

数据图的预处理是通过对数据图的预先处理来减少分布式查询过程中各个计算节点之间的通信代价。在数据图的预处理上，本文方法主要分为两个阶段：1)数据图划分[18]；2)完善邻居顶点[17]。数据图划分阶段是采用现有的多层图划分算法将给定的数据图 G 切分成规模相差不大的一系列数据子图 G_i 来保证分布式环境下的负载均衡；同时也要尽可能减少子图与子图间交互边的数量。完善邻居顶点阶段是将上一阶段切分后的每个子图中的所有顶点作为该子图的核心顶点，把每个核心顶点不在本子图中的邻接顶点复制过来作为子图的一部分，被复制过来的顶点称为辅助顶点。通过这种复制策略来保证所有核心顶点的邻接顶点完整性。整个预处理过程如图 3 所示。

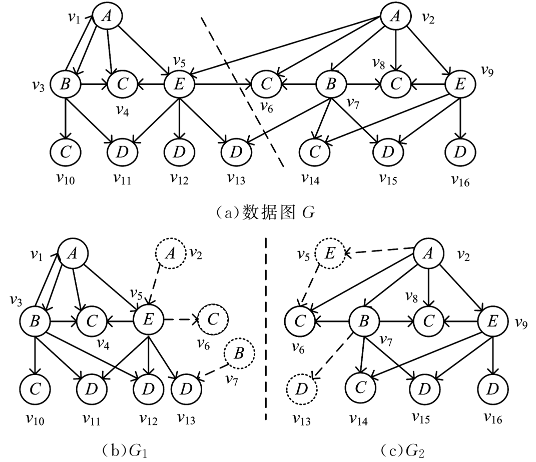


图 3 数据图预处理

Fig. 3 Data graph preprocessing

在图 3 所示的数据图预处理过程中，图划分阶段是通过使用图划分算法将图 3(a)中的数据图 G 沿着虚线切分为两个数据子图 G_1 和 G_2，切分后每个子图中的顶点被称为子图的核心顶点，分别表示为 $G_{1_C}=\{v_1,v_3,v_4,v_5,v_{10},v_{11},v_{12},v_{13}\}$ 和 $G_{2_C}=\{v_2,v_6,v_7,v_8,v_9,v_{14},v_{15},v_{16}\}$。在完善邻居顶点的阶段中，数据图 G_{1_C} 中顶点 v_5 的邻接顶点$\{v_2,v_6\}$、顶点 v_{13} 的邻接顶点$\{v_7\}$不在 G_1 中，因此复制顶点$\{v_2,v_6,v_7\}$到 G_1 中表示为 G_{1_A}，同样地，复制 v_5 和 v_{13} 两个顶点至 G_2 中，对 G_{2_C} 进行邻接顶点完善。预处理后两个数据子图如图 3(b)的 $G_1=\{v_1,v_3,v_4,v_5,v_{10},v_{11},v_{12},v_{13},v_2,v_6,v_7\}$ 和图 3(c)的 $G_2=\{v_2,v_6,v_7,v_8,v_9,v_{14},v_{15},v_{16},v_5,v_{13}\}$ 所示，虚线部分表示的顶点为被复制的顶点和边。

经过预处理后的数据子图使核心顶点的所有邻接顶点都能够在本子图中找到。针对上一节中提出的两层树结构的子查询，预处理过程可以使所有的子查询在匹配过程中不需要访问其他子图就可以得到匹配结果，消除了子查询匹配过程中分布式计算节点之间的通信代价，提高了算法的执行效率。在最坏情况下，预处理后的每个子图 G_i 的大小为 $|V(G)|$。

3.3 候选顶点过滤

DSsearch 算法针对查询图 Q 为每个数据子图创建一个 DSgraph 存储结构，用来存储查询图 Q 每个查询顶点在数据子图 G_i 中对应的候选顶点。DSgraph 结构的设计能够推迟在构建查询匹配结果时对查询顶点的候选集笛卡尔积运算，避免产生大量的中间结果。这一部分也被分为初始化候选集与候选集剪枝两个阶段。在 DSgraph 结构中引入了候选集和辅助候选集两个概念，分别使用 C 和 A 表示。辅助候选集的设计一方面可以在初始化叶子候选集的过程中直接从根候

选顶点的邻接顶点中去过滤候选顶集并且不会错过有效的候选顶点，在初始化阶段提高了算法的效率；另一方面，辅助候选集可以在最后的合并阶段快速将所有数据子图的DSgraph连接起来并删除无效的候选顶点，进一步提高了算法的效率。

在分布式运算的过程中，需要为每个数据子图G_i创建一个对应的DSgraph用来存储该子图上的候选顶点。为了确保匹配结果的正确性，在最坏情况下，所有查询顶点的候选集合大小为$|V(G_i)|$，每一对查询顶点候选集之间的邻接关系为$|E(G_i)|$，对于查询图Q共有$|E(Q)|$对邻接顶点。为查询图Q构建一个DSgraph的空间复杂度为$O(|E(Q)|\times|E(G_i)|)$。在一般情况下，每个数据子图G_i具有$|L(G_i)|$个不同类型的标签，因而每个候选集中候选顶点的数量为$\frac{|V(G_i)|}{|L(G_i)|}$，DSgraph中所有候选顶点的大小为$\frac{|V(Q)|\times|V(G_i)|}{|L(G_i)|}$，$G_i$中的一条边为DSgraph中相邻候选集合之间连接关系的概率为$\frac{1}{|L(G_i)|^2}$，因此相邻候选集合之间邻接关系的大小为$\frac{|E(G_i)|}{|L(G_i)|^2}$。综上可得，在一般情况下DSgraph的空间复杂度为$O(\frac{|E(Q)\times|E(G_i)|}{|L(G_i)|^2}+\frac{|V(Q)|\times|V(G_i)|}{|L(G_i)|})$。

DSsearch算法首先为当前数据子图创建一个与查询图Q对应的DSgraph结构，并根据以下3个约束条件初始化查询顶点的候选集：

1)$L(u)=L(v)$；

2)$d_i(u)\leqslant d_i(v)$，$d_o(u)\leqslant d_o(v)$；

3)$\{L(u_i)\mid u_i\in N(u)\}\in\{L(v_i)\mid v_i\in N(v)\}$。

u表示子查询Q_i中的查询顶点，v表示数据图中的顶点，$d_i(u)$表示顶点u的入度，$d_o(u)$表示顶点u的出度，$N(u)$表示顶点u的邻接顶点。约束1)为标签约束；约束2)为出入度约束；约束3)为邻接顶点约束。

初始化查询顶点的候选集时，先初始化每个子查询的根顶点，然后围绕根顶点的候选集添加子查询叶子查询顶点的候选集。初始化根顶点候选集的算法伪代码如算法2所示。

算法2　initCandidate

输入：数据子图G_i，连接顶点集V_j，子查询集S_Q

输出：DSgraph

```
1. Set v.num ← 0 and v.flag ←false for all v in Gi
2. for each subquery Qi∈SQ do
3.   r ← Qi.root
4.   for each v∈Gi with L(r)do
5.     if v∈Gic and satisfy three constrain then
6.       r.C←r.C ∪ v
7.       v.num += 1;v.flag=true
8.       initLeafCandidate(N(v),Qi,DSgraph)
9.   if v∈GiA and LN(v)∩LVleaf≠∅ then
10.  r.A←r.A ∪ v
11.    initLeafCandidate(N(v),Qi,DSgraph)
12. return DSgraph
```

算法2的第1行对数据子图G_i中的每个核心数据顶点v设置一个计数器num和候选标志$flag$，并初始化num为0，初始化$flag$为false。其中num用于记录核心数据顶点v在所有子查询候选集中出现的次数，$flag$记录v是否存在于DSgraph中，如果为true则存在，否则不存在。第2－3行选出每个子查询Q_i的根查询顶点r。第4－8行将数据子图G_i中与根节点r满足约束1)－3)的数据顶点添加到DSgraph中查询顶点r对应的候选集$r.C$中。其中第4行过滤出与r满足约束1)的数据顶点，第5－8行判断如果$v\in G_{i_C}$并且满足约束条件2)和3)，则把v作为r的候选顶点添加到DSgraph中的$r.C$中(第6行)，计数器$v.num$加1并且设置v的候选标志$flag$为true，表示数据顶点v已经存在于DSgraph中(第7行)，第8行使用v的邻接顶点构建Q_i的叶子候选集，在算法3中将做详细介绍。第9－11行判断如果顶点$v\in G_{i_A}$并且v至少有一个邻接顶点v'与Q_i的叶子查询顶点满足约束1)，则说明v与v'可能为一个跨子图的候选结果的一部分，因此将v添加到DSgraph中r对应的辅助候选集$r.A$中(第10行)，第11行再调用算法3将v满足约束条件的邻接顶点添加到对应的叶子候选集中。

初始化子查询的叶子候选集时，从根候选顶点的邻接顶点中选取数据顶点来初始化叶子查询顶点的候选集，并且在DSgraph中建立根候选顶点与叶子候选顶点之间的邻接关系，使用$C(v,u)$来表示，其中v表示一个候选数据顶点，u表示查询顶点。$C(v,u)$表示查询顶点u对应的候选集中所有与v具有连接关系的候选数据顶点集合。初始化叶子候选集的伪代码如算法3所示。

算法3　initLeafCandidate

输入：N(v)，子查询Q_i，DSgraph

输出：DSgraph

```
1. for each u∈Qi.leaves do
2.   for each v′∈N(v) do
3.   if v′∈Gic and satisfy three constrain then
4.     C(v,u)←C(v,u)∪v′
5.     C(v′,L(v))←C(v′,L(v))∪v
6.     v′.num+=1
7.     if !v′.flag then
8.       u.C←u.C∪v′;v′.flag=true
9.   else if v′∈GiA and L(v′)=L(u) then
10.    C(v,u)←C(v,u)∪v′
11.    C(v′,L(v))←C(v′,L(v))∪v
12.    u.A←u.A∪v′
13. return DSgraph
```

算法3第3－7行指对于v的邻接顶点v'，如果$v'\in G_{i_C}$且与叶子查询顶点u满足约束1)－3)，则在DSgraph中建立v与v'之间的邻接关系(第4－5行)，并在第6行更新v'的计数器$v'.num$。第7行判断v'是否存在于DSgraph中，如果不存在，则在第8行将v'添加到$u.C$中，同时设置v'的候选标志$flag$为true。第9－12行表示如果$v'\in G_{i_A}$且与u满足约束1)，则将v'添加到$u.A$中同时构建v与v'之间的邻接关系。由于当$v'\in G_{i_A}$时，当前子图中v'不具有邻接点完整性，只用来作为辅助候选顶点，因此不需要满足约束2)和3)。图4是图2中的查询图Q在图3预处理后数据子图G_1和G_2上构建的DSgraph存储结构，为了便于理解，这里按照图2中拆分后的所有子查询结构将图4中的DSgraph拆分成如图5所示的一系列DStree。但在算法的执行过程中实际的存储结构仍为DSgraph。其中实线框中代表的是核心候选集，虚线框

中代表的是辅助候选集。

最坏情况下,数据子图 G_i 中的每个数据顶点 v 都可以作为查询图 Q 中查询顶点 u 的候选顶点。算法 2 与算法 3 的时间复杂度均为 $O(|V(G)|\times|V(Q)|)$。

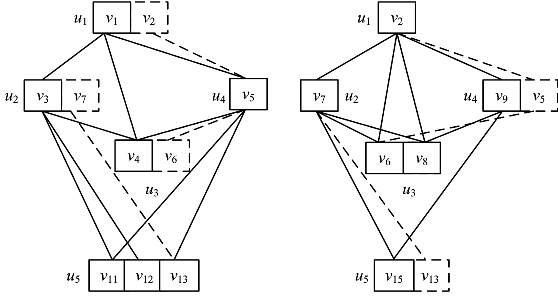
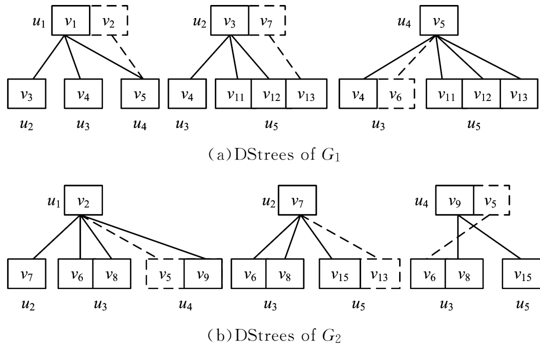


图 4 DSgraph

Fig. 4 DSgraph



(a) DStrees of G_1

(b) DStrees of G_2

图 5 DStrees of DSgraph

Fig. 5 DStrees of DSgraph

最后将局部约束和连接约束作为剪枝条件从 DSgraph 候选集中剪枝无效的候选顶点。剪枝无效候选顶点算法的伪代码如算法 4 所示。

算法 4 filter

```
输入:子查询集合 SQ,连接顶点集 Vj,DSgraph
输出:DSgraph
1. for each query vertex u∈Vj do
2.    for each v∈u.C do
3.       if v.num≠u.NUM then u.C←u.C\v
4. for each Qi∈SQ do
5.    r←Qi.root
6.    for each v∈r.C do
7.       for each u'∈Qi.leaves do
8.       for each v'∈C(v,u') do
9.          if v'∉u'.C and v'∉u'.A then
10.         C(v,u')←C(v,u')\v'
11.         if C(v,u')=∅ then
12.            r.C←r.C\v;break
13.         if v∉r.C then break
14.   for each u∈Qi.leaves do
15.      for each v'∈u.C do
16.         for each v∈C(v',r) do
17.            if v∉r.C and v∉r.A then
18.               C(v',r)←C(v',r)\v
19.         ifC(v',r)=∅ then u.C←u.C\v'
20. return DSgraph
```

剪枝策略 1:连接约束。对于一个标签为 $f(v)$ 的数据顶点 v(这里使用 u 来表示 $f(v)$),当且仅当 $v.num=u.NUM=n$,并且 v 能同时作为所有包含查询顶点 u 的 n 个子查询的候选顶点时,v 才可能为查询图 Q 的候选顶点。在算法 4 的剪枝过程中,当 $v.num\neq u.NUM$ 时,将 v 从对应的候选集 $u.C$ 中剪枝。

剪枝策略 2:局部约束。在查询图 Q 中,查询顶点 u 的邻接查询顶点为 u',数据顶点 $v\in u.C$,当且仅当 v 有一个邻接顶点 $v'\in u'.C$ 时,v 才能满足局部约束。使用 $C(v,u')$ 来表示 v 在 $u'.C$ 中的所有邻接顶点,算法 4 利用 $C(v,u')$ 作为局部约束剪枝无效候选集。当 $C(v,u')=\emptyset$ 时,v 不满足局部约束,此时将 v 从对应的候选集 $u.C$ 中剪枝。

算法 4 首先利用连接约束删除无效候选顶点,然后在此基础上利用局部约束从候选集中删除无效候选顶点,并更新候选集中候选顶点之间的邻接关系。在第 1-3 行利用所有子查询之间的连接约束从连接类型顶点的候选集中剪掉无效的候选顶点。第 1 行中 V_j 表示所有连接类型顶点的集合,由于非连接类型顶点在所有的子查询中只出现了一次,无须判断,因此这里只判断连接类型顶点的连接约束。第 2 行遍历每个连接类型的查询顶点在 DSgraph 中的候选顶点。第 3 行中的 $u.NUM$ 表示所有包含查询顶点 u 的子查询个数,候选顶点 $v.num$ 表示数据顶点 v 在所有包含查询顶点 u 的子查询候选集中出现的次数,如果 $v.num\neq u.NUM$,则说明数据顶点 v 不能作为所有包含查询顶点 u 的子查询的候选顶点,因此不满足连接约束,将 v 从 $u.C$ 中删除。不满足连接约束被删除的候选顶点会使得子查询中的其他一些候选顶点因此而不满足局部约束。第 4-20 行更新 DSgraph 中所有候选顶点之间的邻接关系,并利用局部约束删除候选集中的无效候选顶点。第 6-13 行更新所有子查询中根候选顶点到叶子候选顶点之间的邻接关系并将更新后不满足局部约束的根候选顶点删除。第 10 行从 $C(v,u')$ 中将既不是 u' 候选集中的候选顶点,又不是 u' 辅助候选集中的候选顶点的数据顶点 v' 删除,即更新 v 与 u' 候选顶点之间的邻接关系。第 11 行通过判断 $C(v,u')$ 是否为 $\emptyset$ 来判断更新后的顶点 v 是否满足局部约束,如果 $C(v,u')=\emptyset$,则说明候选顶点 v 没有 u' 类型的邻接候选顶点,则在第 12-13 行将 v 从对应的候选集中删除,由于 v 已经从对应的候选集中删除,因此无须再判断 v 与其他邻接顶点之间的约束,直接跳出第 6 行 for 循环的本次循环,开始判断 $r.C$ 中的下一个候选顶点。第 14-19 行更新所有子查询中叶子候选顶点与根候选顶点之间的邻接关系,并在更新后将不满足局部约束的叶子候选顶点从对应的候选集中删除。第 18 行从 $C(v',r)$ 中删除既不是 r 候选集中的候选顶点也不是 r 辅助候选集中的候选顶点的数据顶点 v,删除完所有这样的 v 后,在第 19 行判断 $C(v',r)$ 是否为 $\emptyset$,如果为 $\emptyset$,则说明 v' 没有 r 类型的邻接候选顶点,不满足局部约束,因此将 v' 从对应的候选集中删除。

在剪枝过程中,filter 算法的时间开销主要分为两部分:利用连接约束剪枝无效候选顶点的代价,更新候选顶点之间的邻接关系并利用局部约束剪枝无效候选顶点的代价。在最坏情况下,查询图 Q 中的所有查询顶点都为连接类型顶点,并且每个查询顶点的候选集大小都为 $|V(G_i)|$,DSgraph 中每一对候选集之间的邻接关系为 $|E(G_i)|$。因此,算法 4 的时间复杂度为 $O(|V(G_i)|\times(|V(Q)|+|E(Q)|\times|E(G_i)|))$。

3.4 中间结果合并

DSsearch 算法将拆分后的子查询集合 S_Q 发布到集群中的各个计算节点上,分别对各个数据子图 G_i 执行分布式子图匹配算法,最后在每个计算节点上都会生成一个 DSgraph 来存储该数据子图上所有过滤后的候选顶点。将这些候选顶点收集到一台计算节点上合并。由于 DSsearch 特定的数据图切分方法,每个子图中的辅助候选顶点是该子图与其他子图的连接点,因此,在合并阶段判断每个子图 DSgraph 中查询顶点 u 对应的 $u.A$ 中的辅助候选顶点 v 是否存在于其他子图的 DSgraph 的 $u.C$ 中,如果存在,则将 $u.A$ 中 v 的邻接顶点合并到 $u.C$ 中 v 的邻接顶点中,否则,直接删除 $u.A$ 中的候选顶点 v,并且将因删除 v 而不满足局部约束的候选顶点也一并删除。最后,连接所有的候选顶点得到查询图 Q 的最终匹配结果,例如,图4中 G_1 和 G_2 的 DSgraph 合并,其中 G_1 的 DSgraph 中 u_1 的辅助候选顶点 v_2 存在于 G_2 对应的核心候选集中,所以对 G_1 中辅助候选顶点 v_2 的邻接顶点与 G_2 中核心顶点 v_2 构建连接关系,合并后的所有查询顶点候选集为 $u_1.C=\{v_1,v_2\}$, $u_2.C=\{v_3,v_7\}$, $u_3.C=\{v_4,v_6,v_8\}$, $u_4.C=\{v_5,v_9\}$, $u5.C=\{v_{11},v_{12},v_{13},v_{15}\}$。利用算法4对合并后的 DSgraph 剪枝,再将所有子查询的候选顶点连接起来并进行组合,得到查询图 $Q=\{u_1,u_2,u_3,u_4,u_5\}$ 对应的最终匹配结果为 $\{\{v_1,v_3,v_4,v_5,v_{11}\},\{v_1,v_3,v_4,v_5,v_{12}\},\{v_2,v_7,v_6,v_5,v_{13}\},\{v_2,v_7,v_8,v_9,v_{15}\}\}$。

4 实验与结果分析

4.1 实验设计

实验使用的集群是由7台华为服务器搭建成的 Spark[19] 高可用分布式集群。每台服务器上有两个6核12线程的 Intel(R) Xeon(R) E5-2620 CPU,64 GB 内存,安装了 4.2.0-27-generic 内核和64位 Ubuntu 操作系统、Java 1.8、Scala 2.11.12、Hadoop 2.9.1、Spark 2.3.0。HDFS 块大小设置为 128 M,副本数设置为3。Spark 的持久化级别为 MEMORY_ONLY。实验时使用 scala 语言实现了4种算法:TwinTwigJoin,PSgL,IVE 和 DSsearch,并在 Spark 分布式集群上运行。

实验中将 soc-LiveJournal,wiki-topcats,cit-Patnets,wiki-talk,web-google,web-stanford 6个真实数据集作为数据图,这6个数据集的顶点规模在十万至百万级别,边规模在百万至千万级别。其中顶点数表示对应数据集中的所有顶点数量,边数表示对应数据集中所有边的数量。平均度为数据图中每个顶点的出度与入度的平均数之和,表示图的稠密度,平均度越大说明数据图越稠密,反之则表示数据图越稀疏。为了更好地进行实验,在数据图预处理的过程中从准备好的100个标签中随机地选取一个标签为每个数据顶点添加标签。实验过程中将所有的数据集都存放在 Hadoop 集群的 HDFS 分布式文件系统中。

表1　实验数据集

Table 1　Experimental data set

数据集	顶点数	边数	平均度
soc-LiveJournal	4847571	68993773	28.47
wiki-topcats	1791489	28511807	31.83
cit-Patents	3774768	16518948	8.75
wiki-talk	2394385	5021410	4.19
web-google	875713	5105039	11.66
web-stanford	281903	2312497	16.41

通过在数据图上进行随机游走,生成数据图的连接子图作为实验使用的查询图。为了使实验结果更具有代表性,设计了分别包含2,4,6,8,10,12,14个查询顶点的7种类别的查询图,分别使用 Q_1-Q_7 表示,且每个类别 i 中分别包含6组不同稠密度的查询图,分别使用 $Q_i^1-Q_i^6$ 表示,这6组的平均度数分别为1,2,3,4,5,6。针对每个稠密度 j 分别设计5个查询图,实验时以5个查询图运行时间的平均值作为 Q_i^j 的运行时间,然后以每个类别 i 中 $Q_i^1-Q_i^6$ 的运行时间均值作为第 i 类查询图 Q_i 的运行时间。

4.2 数据图划分策略对比实验

在预处理过程中使用不同划分策略划分数据图,并在其基础上完善邻居顶点。

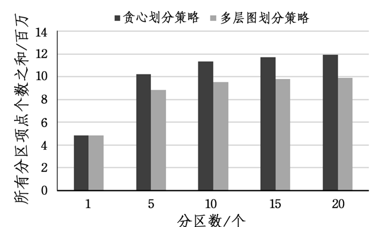
1)划分策略1:贪心划分策略[20]

从数据图的所有数据顶点中选择 N 个互不连通的顶点作为 N 个子图的初始顶点,然后从数据图中剩余的数据顶点中选择一个顶点作为待分配顶点 v,从所有的子图中选择一个顶点数量最少并且与待分配顶点 v 之间联系最紧密的子图作为目标子图 G,将 v 添加到目标子图 G 中。

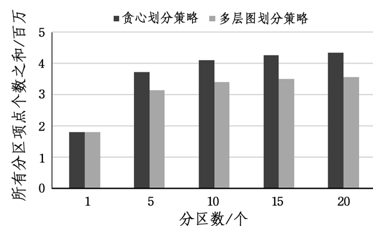
2)划分策略2:多层图划分策略

采用现有的多层次图划分框架[21],并将 KL 算法[22]作为该框架的细化算法来划分数据图。

实验一设计了一组对比实验,分别采用划分策略1和划分策略2对 soc-LiveJournal 和 wiki-topcats 两个稠密图数据集进行分区。从实验结果中可以看出,采用多层图划分策略分区后的子图在完善邻居顶点阶段复制的顶点数量最少,并且分区后随着分区数量的增加,图规模增速较为平缓,且预处理后的图规模基本控制在原图的2倍左右。这说明通过复制顶点来降低分布式集群中计算节点之间数据传输代价这一策略具有可行性。图6为预处理后数据顶点规模随分区数量的变化情况。



(a) soc-LiveJournal 不同图划分策略对比



(b) wiki-topcats 不同图划分策略对比

图6　图划分策略效果对比

Fig. 6　Comparison of graph division strategy effects

4.3 算法的高效性和可扩展性实验

实验二为了验证算法的高效性,设计了不同算法之间的对比实验,以测试算法执行时间与查询图中顶点数量之间的关系,并分别在6个真实数据集上运行,实验结果如图7所示。

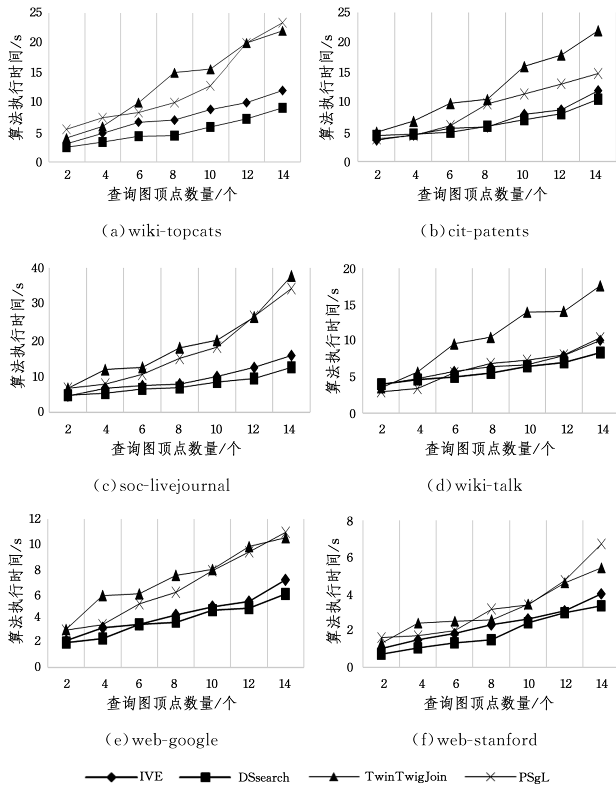


图7 实验结果对比

Fig. 7 Comparison of experimental results

从6个实验对比图中可以明显看出,4个算法的执行时间都与查询图顶点数量成正比,算法运行时间均随着查询图顶点数量的增加而上升,在最开始阶段当查询图顶点数量很少时,4个算法并没有明显的差异,但随着查询图顶点数量的增加,DSsearch算法要略优于IVE算法,明显优于TwinTwigJoin和PSgL算法,且查询图顶点数量越多,优势越明显。这是由于TwinTwigJoin算法将查询图拆分为一系列的星状子查询,并基于MapReduce框架合并子查询结果,MapReduce任务的迭代次数取决于拆分后的子查询数量,当查询图顶点数量增多时,会使子查询数量增多,迭代的次数也会增多,从而产生大量的中间结果,中间结果的读取和存储会使得算法效率降低,因此出现实验中TwinTwigJoin算法增长幅度很大的现象。PSgL算法通过向外探索的方式来查询匹配结果,避免了算法执行过程中产生大量的中间结果,在查询顶点数量较少的情况下要明显优于TwinTwigJoin算法,但随着查询图顶点数量的增多,需要探索的区域扩大,会给分布式系统的计算节点带来巨大的数据传输开销,这是该算法执行时间随查询顶点数量增多的原因。在稠密图中,由于图中数据顶点之间联系过于紧密,数据传输开销也会增大,因此PSgL算法在稠密图中的效率低于TwinTwigJoin算法,而在稀疏图中的效率很高。IVE算法迭代地删除数据图中不满足匹配条件的候选顶点和边,不会产生大量的中间结果,且因删除而引起的连锁反应会使该算法的剪枝效率大幅提升,尽管各计算节点之间有数据传输开销,但还是明显优于前两个算法。DSsearch算法通过设计特定的查询图拆分策略和邻接顶点完善的方法很好地降低了匹配时分布式系统计算节点之间的巨大的网络通信开销,缩短了算法的执行时间。因此可以由实验结果可知,当数据图的稠密度越高时,DSsearch算法效率越高。除此之外,DSgraph存储结构的设计减少了大量中间结果的产生,且DSgraph中辅助候选顶点的设计也加快了中间结果合并的效率,对算法的性能提升有很大的帮助。

从实验二的结果对比图中还可以看出,当数据集越稠密,DSsearch算法比IVE算法越高效,这说明了DSsearch算法使用查询图拆分和对应的数据图预处理方案来降低数据传输代价的方案具有可行性和高效性。

实验三验证DSsearch算法的执行时间和查询图的稠密度之间的关系,每次查询分别使用稠密度相同但查询顶点个数不同的查询图的运行时间的平均值作为该稠密度下的运行时间,实验结果如图8所示。其中,横坐标表示查询图中查询顶点的平均度数,平均度数越大说明查询图越稠密。从图中可以看出,6个数据集上的运行时间均随着查询图稠密度的增加呈现略微的下降趋势,且当数据集的稠密度越小时,下降的趋势越明显。这是因为越稠密的查询图结构越复杂,在匹配过程中的剪枝能力就越强,在稀疏的数据集中就可以通过查询顶点的局部结构约束在算法初期剪枝大量的无效候选结果,避免了在算法后期对无效候选集的判断,从而提升算法的效率。因此,wiki-talk和cit-Patents数据集的下降幅度要超过其他几个数据集。

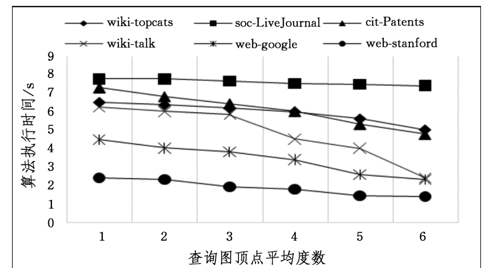


图8 算法执行时间随查询图稠密度的变化

Fig. 8 Chang of running time of algorithms with query graph denisty

实验4通过扩增集群的规模来测试DSsearch算法的可扩展性,实验结果如图9所示,在初始阶段除web-stanford数据集外的其他几个数据集的执行时间都随集群规模的扩增迅速减少,然后趋于平缓。而web-stanford数据集的算法执行时间随着集群规模的扩增并没有下降,反而呈现略微的上升趋势,这是由于当数据集很小而集群规模越来越大时,收集各个计算节点上的中间结果以及对其进行合并已经成为影响算法执行效率的主要因素。集群的扩增会产生通信代价,因此算法执行时间会随着集群规模的扩增呈小幅度的上升趋势。除此之外,数据图规模越大,算法执行时间受集群规模影响的趋势越明显。综上所述,DSsearch算法在大规模图数据上的查询具有高效性和可扩展性。

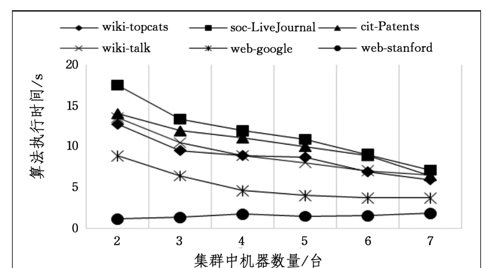


图9 算法运行时间随集群规模的变化

Fig. 9 Change of running time of algorithms with cluster size

结束语　本文针对大规模图数据设计了 DSsearch 分布式子图匹配算法,该算法通过查询图拆分、数据图预处理、候选顶点过滤、中间结果合并一系列步骤有效地解决了在分布式环境下对大规模图数据的子图匹配问题,通过特定的查询图拆分方法和数据图预处理策略降低了子图匹配过程中分布式环境下各个计算节点之间数据传输的代价,并针对查询图设计存储结构 DSgraph 来推迟笛卡尔积的产生,避免了在匹配过程中产生大量的中间结果。最后在分布式环境 Spark 集群下使用真实图数据验证了该算法的高效性和可扩展性。本文所研究的是大规模图数据上的精确匹配,对于近似匹配还有待进一步的研究。

参 考 文 献

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

[2] CORDELLA L P,FOGGIA P,SANSONE C,et al. A (sub) graph isomorphism algorithm for matching large graphs[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,2004,26(10):1367-1372.

[3] LEE J,HAN W S,KASPEROVICS R,et al. An in-depth comparison of subgraph isomorphism algorithms in graph databases [J]. Proceedings of the VLDB Endowment,2012,6(2):133-144.

[4] HAN W S,LEE J,LEE J H. Turbo iso:towards ultrafast and robust subgraph isomorphism search in large graph databases [C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM,2013:337-348.

[5] BI F,CHANG L,LIN X,et al. Efficient subgraph matching by postponing cartesian products[C]// Proceedings of the 2016 International Conference on Management of Data. ACM,2016:1199-1214.

[6] GIUGNO R,SHASHA D. Graphgrep:A fast and universal method for querying graphs[C]// 16th International Conference on Pattern Recognition. IEEE,2002:112-115.

[7] ZOU L,CHEN L,YU J X,et al. A novel spectral coding in a large graph database[C]// Proceedings of the 11th International Conference on Extending Database Technology:Advances in database technology. ACM,2008:181-192.

[8] HE H,SINGH A K. Closure-tree:An index structure for graph queries[C]// Proceedings of the 22nd International Conference on Data Engineering(ICDE'06). IEEE,2006:38-38.

[9] SUN Z,WANG H,WANG H,et al. Efficient subgraph matching on billion node graphs[J]. Proceedings of the VLDB Endowment,2012,5(9):788-799.

[10] SHAO B,WANG H,LI Y. Trinity:A distributed graph engine on a memory cloud[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM,2013:505-516.

[11] ZHAO Z,WANG G,BUTT A R,et al. Sahad:Subgraph analysis in massive networks using hadoop[C]// 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS). IEEE,2012:390-401.

[12] ALON N,DAO P,HAJIRASOULIHA I,et al. Biomolecular network motif counting and discovery by color coding[J]. Bioinformatics,2008,24(13):i241-i249.

[13] DEAN J,GHEMAWAT S. MapReduce:simplified data processing on large clusters[J]. Communications of the ACM,2008,51(1):107-113.

[14] LAI L,QIN L,LIN X,et al. Scalable subgraph enumeration in mapreduce[J]. Proceedings of the VLDB Endowment,2015,8(10):974-985.

[15] SHAO Y,CUI B,CHEN L,et al. Parallel subgraph listing in a large-scale graph[C]// Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM,2014:625-636.

[16] REZA T,KLYMKO C,RIPEANU M,et al. Towards Practical and Robust Labeled Pattern Matching in Trillion-Edge Graphs [C]// 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE,2017:1-12.

[17] SUO B,LI Z,PAN W. Parallel subgraph matching on massive graphs[C]// International Congress on Image and Signal Processing,BioMedical Engineering and Informatics (CISP-BMEI). IEEE,2016:1932-1937.

[18] BENLIC U,HAO J K. A multilevel memetic approach for improving graph k-partitions[J]. IEEE Transactions on Evolutionary Computation,2011,15(5):624-642.

[19] ZAHARIA M,CHOWDHURY M,FRANKLIN M J,et al. Spark:Cluster computing with working sets[J]. HotCloud,2010,10(10-10):95.

[20] LIU X,ZHOU Y,GUAN X,et al. A feasible graph partition framework for parallel computing of big graph[J]. Knowledge-Based Systems,2017,134:228-239.

[21] KARYPIS G,KUMAR V. A fast and high quality multilevel scheme for partitioning irregular graphs[J]. SIAM Journal on Scientific Computing,1998,20(1):359-392.

[22] KERNIGHAN B W,LIN S. An efficient heuristic procedure for partitioning graphs[J]. The Bell System Technical Journal,1970,49(2):291-307.