大图中全部极大团的并行挖掘算法研究

汤小春 周佳文 田凯飞 李战怀
(西北工业大学计算机学院 西安 710129)

摘 要 该文的目的在于优化现有的大图数据中全部极大团挖掘算法.在生物网络、社会网络及 web 分析中,找出图中的全部极大团是一个重要的应用.随着图数据规模的增大,传统的极大团挖掘算法因无法满足性能要求而被并行处理方式取代.但是,在现有的并行处理方法中,需要过滤大量的重复极大团和检测非极大团,降低了算法的性能.论文在分析了现有的极大团并行算法后,提出了新的大图中全部极大团挖掘算法.首先,使用顶点的偏序关系消除了冗余极大团以及非极大团的产生;第二,根据两个极大团之间至少存在一对无边的顶点的特征,提出了多颜色顶点涂色分片算法,将大图的顶点分为全色和半色两个集合;第三,证明了涂色分片算法是 NP 完全问题以及有一个多项式时间的 2 近似算法,并给出了近似算法;第四,基于多色顶点分片实现了一个并行的全部极大团挖掘算法,该算法只对全色顶点与它的邻接顶点组成重叠子图进行极大团挖掘;最后,对算法的性能以及加速比特性进行了评价,得出该算法能够处理百万个节点的大图并且性能比现有的算法有较大提高的实验结果.

关键词 图挖掘;极大团 ;涂色分片;并行算法;重叠子图
中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2019.00513

An Efficient Parallel Algorithm for Maximal Clique Enumeration in a Large Graph

TANG Xiao-Chun ZHOU Jia-Wen TIAN Kai-Fei LI Zhan-Huai
(*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710129)

Abstract A maximal clique is perhaps the most fundamental dense substructure in a graph. Maximal clique enumeration is an important tool to discover densely connected sub graphs, with numerous applications on web graphs, social networks, and biological networks. Existing methods for finding all maximal cliques in a large graph based on distributed computing environment are, in our opinion, not quite efficient. We propose a better method using a multi-color partition algorithm to decompose large graph into lots of overlap sub graphs for finding all cliques in parallel. In the full paper, we explain our enumerating all maximal cliques method in detail. In this abstract, we just add some pertinent remarks to list the four topics of explanation. The first topic is: we use the heuristics in order to decrease the vertex search space. In this topic, we discuss the two proposes of the algorithm: using vertex order to remove non-maximal cliques and duplicates, parallel policy to fit in distributed computing environment. The second topic is: we propose multiple color method to divide the vertexes of graph into two sets, which are related vertexes set and irrelevant vertex set. The two sets stained by full colors and semi colors respectively. Full colored vertex and its adjacent vertexes compose an overlap sub graph. On the other hand, semi colored vertexes is used to assure no lost of cliques. The third topic is a problem of the minimum full colors set. We first designed an approximation algorithm. Next we proved that this problem is NP complete. Then we proved this algorithm have an 2 approximation algorithm for polynomial

收稿日期:2017-07-30;最终修改稿收到日期:2018-01-22.本课题得到中国科技部国家重点研发计划资助项目(2018YFB1003403)资助.
汤小春,男,1969 年生,博士,副教授,主要研究方向为图数据管理、分布式计算等.E-mail:tangxc@nwpu.edu.cn.**周佳文**,女,1994 年生,硕士研究生,主要研究方向为大数据计算.**田凯飞**,男,1993 年生,硕士研究生,主要研究方向为大数据计算.**李战怀**,男,1961 年生,博士,教授,中国计算机学会会员,主要研究方向为海量数据管理、大数据计算等.

time. The fourth topic is: a parallel algorithm for enumerating all maximal cliques based on full colors set is proposed. In this topic, we use full colored vertexes and its related semi colored vertexes to split input graph to overlapped sub graphs. The full colored vertex acts as a pivot to enumerate maximal cliques over an overlap sub graph in parallel. Finally, we do experiments to appraise our method. Experimental evaluation on real datasets from various domains show that the response time of our method based on multi-color algorithm is more efficient comparing with the existing methods. The experiments on cluster of PCs also show that the algorithm can effectively process a variety of large real-world graphs with millions of vertices and the scalability of the multi-color partition algorithm is very well.

Keywords graph mining; maximal clique enumeration; multi-color partition; parallel algorithm; overlap sub graph

1 引 言

目前,以大图数据为基础的应用程序正被广泛使用[1-3],其中完全子图结构[4-6]的挖掘是这类应用中的一个重要分支,它在复杂网络的社团发现[3]、实体识别[5]、生物信息[7-9]、信息检索[4,10]、计算机视觉[11]、计算拓扑学[12]以及电子商务[13]等领域中都发挥着重要作用.例如,在社会网络中,通过完全子图(也称团)的发现,可以找到多个关系非常密切的社区;在生物信息领域[8-9],通过团的发现,可以得到蛋白质结构中频繁出现的模式,从而预测蛋白质的结构,也可以得到形状相似点,从而发现蛋白质之间的功能关系.

对于一个连通图数据中团的计算有两类不同的方向,一类是计算图中的最大团,它是 NP 完全问题.文献[14-16]是解决此类问题的典型代表.文献[14]采用启发式方式进行剪枝,减少了一些不必要的计算,因而能够快速得到最大团.文献[15]采用启发式策略进行局部查找优化,通过对已经查找到的最大团的记录,优化局部查找过程,去掉一些不可能出现极大团的顶点;文献[16]采用贪心算法来计算图数据中的最大团,它以将度最大的顶点增加到团中的方式,执行到找不到顶点为止,该算法的效率是线性的,但是它要求图的顶点规模不能太大;另一类是挖掘图数据中包含的全部极大团(MCE),文献[17-21]提出了几种典型的挖掘算法和应用场景.对于某些稀疏图,如平面图以及度较小的图,往往只包含线性大小的团,挖掘全部的极大团可以在线性时间内完成.如果对图的度进行一定的限制,算法的性能也可以进一步提高.文献[17] 中提出的最大团计算方法,即 BK 算法,是其它各种计算方法的根基,它采用回溯技术来搜索问题空间和限制搜索空间的大小,是实际应用中最快的算法.文献[18]是文献[17]的改进,采用树的形式输出结果以及证明了算法的复杂度,其算法复杂度为 $O(3^{n/3})$,是目前最快的全部极大团挖掘算法.文献[19-20]都是对经典算法的改进,加入了一些约束条件.文献[21]是一种采用广度优先策略的极大团挖掘算法,它的运行时间与输出大小之间存在比例关系,算法运行时间是 $O(|V||E|\mu)$,其中 μ 代表极大团的数量.

上述算法只支持单机环境下的计算,随着图中包含的顶点数增多,最坏情况下 MCE 问题的算法复杂度可以达到指数级别[22],因此 MCE 问题逐渐由单机处理向分布式处理或并行处理转变.例如,在 Web 分析、社会网络以及科学应用中,图中包含几百万个顶点甚至几亿个顶点,单台计算机一般无法进行处理.因此,并行方式或分布式方式就成为解决此类问题的首选[23].

大图数据的 MCE 问题始于文献[23-24],文献[23]的算法采用自底向上策略,先得到较小的候选子团,然后在候选子团的基础上采用扩张的方式得到新的子团.一次计算完成后,采用过滤方法去掉重复团就得到新候选子团.这种计算方法需要大量的内存来存储中间结果.文献[24]改进了文献[23]中算法的不足之处,扩张的时候利用启发式策略进行剪枝,减少了不必要的计算,但是其计算任务到处理器的映射策略不佳.

内存的大小以及处理器性能是大图 MCE 问题的瓶颈,如果图数据不能被一个计算节点存储,算法就显得无能为力,因此并行或分布式处理成为首选.文献[25] 使用大型机解决该问题,它设计了一个基

于 MPI 的并行极大团挖掘算法(DMC)，该算法采用计算-过滤方式. 首先，每个顶点单独作为一个任务来执行顺序极大团挖掘算法，初步的结果中既包含极大团，又包含冗余的极大团以及非极大团，然后进行过滤，去掉冗余的极大团和非极大团. DMC 计算过程中，结果的过滤非常耗费时间，大大影响了系统的总体性能.

随着 MapReduce 编程模式的出现，研究者开始采用 MR 模型来解决 MCE 问题. 文献[26]提供了一个基于 MapReduce 的算法(MMCE). 该算法首先将输入图分解为多个子图，然后将每个子图提交到一个单独的 Reduce 任务上进行处理. 如何分解子图是该算法的关键问题，文献[26]将图中的每个顶点分解为一个子图，导致大量的非极大团及冗余极大团的存在，浪费了宝贵的计算资源.

文献[27]说明了并行计算中负载平衡的重要性以及如何在计算节点之间进行平衡，并比较了 MPI 和 MapReduce 处理模式的优缺点. MPI 并行框架的优点是用户可以控制处理器及其并行的方式，在处理任务前处理器之间可以动态负载平衡，其缺点是需要用户提前定义任务，无法自动划分子任务. MapReduce 平台的优点在于不需要人为地划分子任务，但是无法很好的实现负载平衡.

文献[28]在结合 MPI 和 MapReduce 二者优点的基础上，提出了一种基于 MapReduce 的极大团查找算法(PECO). 该算法通过顶点的排序以及计算顶点特征，即顶点的度、顶点包含的三角形的数量等，使得各个 Reduce 任务的计算时间尽量一致，从而改进了文献[26]中的负载不平衡问题.

现有的挖掘大图数据中全部极大团算法，基本上都是以文献[18]中的算法为基础，使其适用于不同的并行运行环境[23-24,28]，这些算法都存在着大量非极大团或者冗余极大团的计算问题. 本文的主要目的是对算法进行优化从而解决上述问题. 对多个串行算法进行分析后，本文选择了其中时间复杂度最小的算法，即文献[18]中的算法. 本文中优化算法的思想是舍弃不需要计算的顶点，只对必要的顶点计算其包含的全部极大团. 算法首先将图数据中的顶点分为全色和半色两类，全色顶点的集合称为相关顶点集合，半色顶点集合称为不相关顶点集合，然后以相关顶点集合中的每个顶点及其所有邻接顶点组成导出子图，之后作为一个独立的任务执行，其余不相关顶点不需要产生子图，也不需要作为一个可调度的任务来执行，因而减少了可执行的任务数量，从而提高了计算效率.

论文的主要工作有：

(1) 提出顶点涂色分片算法，大大降低了重叠子图的数量，既满足了并行化要求，又减少了并行任务的数量；

(2) 证明顶点涂色分片算法是 NP 完全问题并分析了近似算法的特性；

(3) 基于顶点涂色分片思想，设计并实现全部极大团并行挖掘算法.

本文第 2 节对问题进行描述；第 3 节介绍大图中的极大团顶点涂色分片方法；第 4 节介绍包含某个顶点的全部极大团的查找算法；第 5 节给出极大团的并行计算过程；论文的最后，在涂色分片和非涂色分片模式下，利用斯坦福大学的图数据①[29]比较算法性能，验证了算法的优点.

2　问题描述

给定一个图 $G(V,E)$，包含有限顶点的集合 V 和有限边的集合 E. 如果 $(v,w)\in E$，就称顶点 v 和 w 是相互邻接的.

对于图 G 中的任何一个顶点 v，使用 $\Gamma(v)$ 代表其邻接顶点 $\{w|(v,w)\in E\}$. 对于一个集合 $W\subset V$，其共同的邻接顶点可以用 $\Gamma(W)$ 表示，记为 $\bigcap_{w\in W}\Gamma(w)$.

对于一个顶点的子集 $W\subset V$，$G(W)=(W,E(W))$，其中，图 $G(W)$ 是图 G 由顶点 W 导出的子图，边的集合为：

$$E(W)=\{(v,w)\in W\times W \mid (v,w)\in E\}.$$

给定一个顶点的子集 $Q\subseteq V$，如果对于 Q 中任意一对顶点 $v,w(v\neq w)$，都存在 $(v,w)\in E$，则称其导出的子图 $G(Q)$ 是完全的. 在这种情况下，简单地称 Q 是一个完全子图. 一个完全子图也称为一个团.

一个大图中，对于任意的两个团，如果存在包含关系时，那么其中一个团不可能称为极大团. 如果两个团中至少存在一个不同的顶点，那么就认为这两个团是不一样的，因此挖掘一个大图中的全部的团就需要遍历整个图数据，找出所有不同的团. 同时，还要求找到的团中，任意两个团的顶点之间没有包含关系，也就是说得到的每个团的顶点都不是其它团的顶点集合的子集，这样的团定义为极大团. 故，挖掘大图中包含的全部极大团就是论文的核心问题.

① Stanford large network dataset collection, http://snap.stanford.edu/data/index.html

设 $S_A(V_A,E_A)$和 $S_B(V_B,E_B)$是图 G 中任意的两个团，如果对于$|V_A|\leq|V_B|$，都存在 $V_A-V_B\neq\varphi$，那么就称 S_A和 S_B是图 G 中的两个不同的极大完全子图，简称极大团.

问题描述：给定一个大规模的连通图 G，设计一种并行的算法，可以使用多台处理器并行挖掘 G 中的全部极大团 S，对于 S 中的任何一个子图 s，s 在 G 中只存在唯一的单射 f，并且 f 是一个极大团.

3 大图中的极大团的计算方法

计算一个大图中的极大团时，要求算法满足正确性和高效性. 所谓的正确性指的是既不能出现冗余和遗漏情况，也不能出现非极大团，即每个计算出的团不能是其它团的子团. 所谓的高效性是指在许可的时间内计算出全部的极大团. 由于一个大图中可能会包含大量的极大团，单个处理器很难在指定的时间内检测出全部的极大团，因此，就需要使用多个处理器并行计算. 为了满足以上要求，本文采用的策略是针对图中的每个顶点，分别计算出它所包含的极大团，然后进行合并得到全部的极大团. 但是，这种并行计算方式可能带来其它问题，例如，两个并行进程之间计算出的极大团是相等的，即冗余问题，或者两个并行进程中计算的团之间存在包含关系，即非极大团问题. 故，常用的算法采用先并行计算后过滤的模式，先计算每个顶点与其邻接顶点组成的所有团，然后从这些团的集合中，去掉那些冗余部分以及非极大团，最后得到正确的结果.

3.1 计算包含一个指定顶点的极大团

选择一个顶点 v，将$\{v\}\cup\Gamma(v)$形成的导出子图 $G(v)$作为图 G 的一个重叠子图，以 v 为核心，计算重叠子图中的全部极大团. 计算出的每一个极大团 C_i，必定满足 $v\in C_i$；另外，也不存在极大团 C_i，C_j，$i\neq j$，满足 $C_i\subseteq C_j$或者 $C_j\subseteq C_i$.

下面将介绍包含一个指定顶点的极大团的挖掘算法. 最开始，假设 C 只包含指定顶点 v，通过一步一步追加 v 的邻接顶点到 C 的方法，形成更大的完全子图，持续进行直到得到一个极大的完全子图.

算法的计算过程如下：

(1) 选择图 G 中的一个顶点 q 作为根，计算由顶点 q 的邻接顶点组成的导出子图 $G(v)$，其中，顶点集合为：

$$V(G_q)=\{q\}\cup\Gamma(q)$$

边的集合为：

$$E(G_q)=\{(u,v)\mid u,v\in V(G_q)\wedge(u,v)\in E\}.$$

(2) 采用深度优先策略，对于每个顶点 $p\in V(G_q)$，如果 p 与候选团中的每个顶点之间都存在边，尝试将 p 扩展到极大团中，$V(G_{pq})=\{p,q\}\cup\Gamma(q)\cap\Gamma(p)$.

(3) 递归执行步骤(2)，直到导出子图的顶点集为空.

上述递归过程最终得到包含顶点 q 的全部极大团. 针对图 G 中的每个顶点，采用上述方法，就可以得到图 G 中包含的全部极大团. 但是，该方法计算出的全部极大团中会出现冗余极大团.

3.2 冗余极大团的去除

为了计算图 G 中包含的极大团，可以针对图 G 中的每个顶点，按照 3.1 节的方法来计算极大团，但是这种策略会导致结果中存在大量的冗余极大团. 当计算包含某个顶点的全部极大团的时候，可能会出现以下情况，假设有顶点 u 和顶点 v，如果顶点 u 和顶点 v 之间存在边，那么指定顶点 u 的极大团与指定顶点 v 的极大团之间一定存在冗余，即同样的极大团被计算两次. 例如，如图 1 所示，在计算包含顶点 6 的极大团时，可以计算出三个极大团{6,8,9}，{5,6,7,8}和{4,5,6}，而在计算以 4 为顶点的极大团时，可以得到极大团{2,3,4}和{4,5,6}. 由此看出，极大团{4,5,6}被计算了两次，是一个冗余的极大团. 另外，{5,6,7}也是一个团，但它是非极大团，因为它的顶点集合是团{5,6,7,8}的子集.

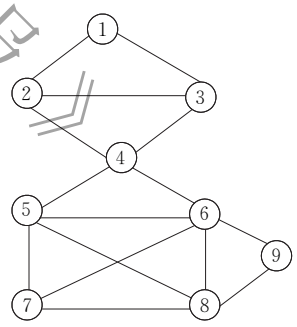


图 1 图的极大团

为了减少冗余的发生，可以先行定义图 G 中 n 个顶点的偏序关系，即 $v_1\prec v_2\cdots v_k\cdots\prec v_n$. 对于一个给定的集合 $P=\{v_1,\cdots,v_{k-1}\}$，在计算顶点 v_k 包含的极大团时，如果顶点 v_k 与顶点 $v_i\in P$ 和 $v_j\in P$ 之间存在边，那么在计算包含顶点 v_k 的极大团时，由于 v_k 是 v_i 和 v_j 的后继，因此需要去掉顶点 v_i 和 v_j，防止冗余极大团的出现. 例如，在计算顶点 6 的极大团时，设置顶点 6 的去除顶点集合为{4}，那么就不会计算出极大团{4,5,6}，而极大团{4,5,6}则可以通过顶点 4 计算得出.

3.3 大图中全部极大团并行挖掘算法

根据上面的计算过程，首先对图 G 中的顶点设置一个偏序关系，然后依据偏序关系计算每个顶点对应的去除顶点集合，最后并行计算每个顶点包含的极大团，得到全部的极大团.

算法 1 为图 G 中全部极大完全子图查找算法.

算法 1.　vCliques($G(V,E)$)

输入：连通图 G，包含顶点集合 V 和边的集合 E

输出：图 G 中的全部极大团

过程：vCliques(G)

1. 设置图 G 中顶点的偏序关系 $v_1 \prec v_2 \cdots v_k \cdots \prec v_n$；
2. FOR EACH $v \in V$
3. $G(v)=(V_v,E_v)$，$V_v=\{v\} \cap \Gamma(v)$，
   $E(v)=\{(u,v) \mid u,v \in V(v) \wedge (u,v) \in E\}$
4. $DEL(v)=\{v_i \mid (v,v_i) \in E \wedge v_i \prec v\}$
5. $Q=\varphi$；$SUBG=\varphi$；$CAND=\varphi$；
6. $SUBG=V_v \cap \Gamma(v)$
7. $CAND=SUBG-DEL(v)$
8. $Q=Q \cup \{v\}$
9. getCliques ($SUBG$,$CAND$)
10. RETURN

过程 1. getCliques($SUBG$,$CAND$)

1. IF $SUBG=\varphi$ RETURN Q；
2. ELSE
3. 从 $SUBG$ 中选择选择一个顶点 u
4. WHILE $CAND-\Gamma(u) \neq \varphi$｛
5. 从 $CAND-\Gamma(u)$ 中选择一个顶点 p
6. $Q=Q \cup \{p\}$
7. $SUBG_p=SUBG \cap \Gamma(p)$
8. $CAND_p=CAND \cap \Gamma(p)$
9. getCliques($SUBG_p$,$CAND_p$)
10. $CAND=CAND-\{p\}$
11. $Q=Q-\{p\}$
12. ｝
13. RETURN φ

算法 1 中，Q 是一个全局变量，它保存一个极大的完全子图. $SUBG$ 代表顶点 v 的邻接顶点 $\Gamma(v)$，另一部分 $CAND$ 代表顶点 v 的邻接顶点 $\Gamma(v)$ 与去除顶点集合的差. 在每次的递归调用中，若 $SUBG$ 为 φ，算法就输出 Q 的内容作为一个极大团；若 $CAND$ 不为空，则算法递归执行.

vCliques 过程的第 3 行得到顶点 v 的邻接顶点集合，第 4 行按照顶点的偏序关系进行去除，第 6 行和第 7 行分别计算出子图的顶点集合和候选顶点集合，然后调用 getCliques 过程来计算团. getCliques 首先检查 $SUBG$ 是否为空，若为空，输出一个极大团；若不为空，从 $CAND$ 中取一个顶点进行扩张，继续调用 getCliques 过程，最终可以得到包含顶点 v 的全部极大团.

例如，通过上述算法可以计算出包含图 1 中顶点 1 的一个极大团为{1,2,3}. vCliques 的第 8 行得到 Q 为{1}，第 6 行得到顶点集合 $SUBG$ 为{2,3}，$CAND$ 为{2,3}. 调用过程 getCliques，从{2,3}中获得一个顶点 3 后，得到 $CAND-\Gamma(3)$ 为{3}，getCliques 的第 6 行得到 Q 为{1,3}，$SUBG$ 为{2}. 递归调用 getCliques 后，选择顶点 2，Q 为{1,2,3}，$SUBG$ 为空，输出一个极大团{1,2,3}. 故，包含顶点 1 的极大团就为{1,2,3}. 同理，在计算包含顶点 2 的极大团的过程中，去除顶点 1，得到一个极大团{2,3,4}；顶点 3 计算过程中去除顶点 1 和 2 以后，无法得到任何的极大团；顶点 4 计算过程中去除顶点 2 和 3 以后，可以得到一个极大团{4,5,6}；顶点 5 计算过程中去除顶点 4 后，得到极大团{5,6,7,8}；而顶点 6 计算过程中去除顶点 4 和 5 以后，获得极大团{6,8,9}；其它的顶点计算过程中在去除掉对应的顶点后，都无法获得极大团.

综上所述，可以得到以下两个结论：(1) 对于算法 1 的第 2 行，顶点 v 的邻接顶点不应被执行；(2) 不与顶点 v 邻接的顶点必须被执行.

对于结论(1)，假设顶点 v_i 在算法 1 的第 2 行进入循环执行，那么顶点 v_i 的邻接顶点 $v_{i+1},\cdots,v_k$ 是不需要执行循环的，因为在过程 getCliques 的扩展中，包含顶点 $v_{i+1},\cdots,v_k$ 的团已经被搜索了一次，再次使用这些顶点搜索只能得到冗余极大团或者非极大团；对于结论(2)，虽然顶点 v_i 和 v_j 之间不存在边，但是具有相同的邻接顶点 $v_{i+1},\cdots,v_k$. 若 $v_{i+1},\cdots,v_k$ 包含一个去掉顶点 v_i 的团，相对于 v_i 来说，就是一个非极大团，但是如果这些非极大团在扩展一个顶点 v_j 后，仍然是极大团，那么它们就有可能是一个极大团，所以不与顶点 v_i 邻接的顶点必须执行算法 1 中的循环.

3.4 算法 1 的分析

定理 1.　设 $G(V,E)$ 中的一个顶点集合 $Q=\{p_1,p_2,\cdots,p_d\}$ 是一个极大团，那么以 Q 为基础的极大团必定有：

$$Q \subseteq (\{p_1\} \cup \Gamma(p_1)) \cap \cdots \cap (\{p_d\} \cup \Gamma(p_d)).$$

证明参考附录.

性质 1.　给定 $G(V,E)$ 中的一个顶点 v，包含 v 的极大团只能在顶点集合 $\{v\} \cup \Gamma(v)$ 中进行扩张.

证明.　依据定理 1，直接得证.

定理 2. 算法 1 可以正确、完备地计算出图中的全部极大子团(证明参考附录).

在算法 1 中,对顶点设置了偏序关系,在计算某个顶点时,首先要得到去除顶点集合 $DEL(v)$. 此时,$SUBG$ 和 $CAND$ 集合就不相同,$CAND$ 集合中的顶点数明显少于 $SUBG$,因此,过程 1 中第 4 行的循环结束时,$SUBG$ 不为空,此时,计算出的团是一个非极大团,扩展后的结果不输出. 例如,在数据图图 1 中,计算包含顶点 6 的极大团时,按照算法 1 中定义的偏序关系,顶点 4 和顶点 5 不包含在 $CAND=\{6,7,8,9\}$中,而 $SUBG=\{4,5,6,7,8,9\}$却包含顶点 4 和 5,在递归过程中,当计算团$\{6,7,8\}$时,$CAND$ 为空,$SUBG$ 不为空,所以不输出团$\{6,7,8\}$,因为它是一个非极大团.

由上述分析可以看出,算法 1 中存在大量的非极大团计算,浪费了系统资源. 本论文的核心在于减少这些非极大团的计算,对算法 1 进行优化.

4 极大团并行挖掘算法的优化

对于图 1 中的图数据,可以看出,当计算了顶点 1 之后,围绕顶点 2 和顶点 3 的扩张过程也就没有必要了. 顶点 4 计算后,围绕顶点 5 和 6 的扩张也没有必要进行. 顶点 8 计算完后,围绕顶点 7 和 9 的扩张过程亦可以去掉. 因此,对算法 1 进行优化的策略是找到必须要进行计算的顶点的集合,而其它的顶点就无需进行扩张计算. 下面将讨论如何通过涂色方法,得到必须进行扩张计算的顶点.

4.1 顶点的涂色分片

在一个团中,任意两个顶点之间的最短距离为一条边的长度,也就是说,如果顶点集合 Q 中的任意一个顶点到某个顶点 v 之间的最短距离为 1,那么我们认为 Q 中的顶点可能包含在 v 所在的极大团中;反过来,如果顶点集合 Q 中的任何一个顶点到某个顶点 v 之间的最短距离大于 1,那么我们认为 Q 中的顶点一定不可能包含在 v 所在的极大团中.

定理 3. 给定一个图 $G(V,E)$,$|V|=n$,顶点集合 V 被分解为子集 V_1,V_2,其中 $V_1=\{v_1\}\cup\Gamma(v_1)$,$V_2=V-V_1\cup M$,而 M 表示为:

$$M=\{v_k\mid(v_k,v_j)\in E\wedge v_k\in V_1\wedge v_j\notin V_1\}$$

则必然存在顶点 $u\in V_2$,使得$(u,v_1)\notin E$,即在集合 V_2 中必定存在一个顶点 u,顶点 u 和顶点 $v_1\in V_1$ 之间不存在边. 此时,子集 V_1 对应的图 G 中的极大团与子集 V_2 对应的图 G 中的极大团不可能重复(证明参考附录).

对于任意一个图的顶点集合 V,如果先找到一个包含顶点 v_1 以及其邻接顶点形成的导出子图 $G(v_1)$,然后得到另一个子图,其顶点集合由:(1)剩余的全部顶点 $V-V_1$;(2)这样的一些顶点 v_k,存在边$(v_k,v_l)\in E$,$v_k\in V_1$,$v_l\in V-V_1$. (1)和(2)的顶点集合构成另外一个导出子图. 这样就形成图的两个不同的分片. 这样的两个分片中,按照定理 3,它们包含的极大团是不同的.

根据定理 3,如果对顶点集合 V_2,选择一个顶点 $u\in V_2$,使得$(u,v_1)\notin E$,那么就可以将 V_2 再次分解,得到一个顶点集合$\{u\}\cup\Gamma(u)$形成的子图 $G(u)$ 和新的顶点子集 V_3 形成的子图,当然,它们之间也满足定理 3. 这样的分解过程递归执行,直到不能分解为止.

从以上的分解过程,可以得出一个结论,给定一个图 $G(V,E)$,$|V|=n$,按照定理 3 进行 k 次分解后,顶点集合 V 被分解为 k 个子集 $V_1,V_2,\cdots,V_k$. 对于任意两个子集 V_i 和 V_j,V_i 和 V_j 都包含一个唯一的顶点,满足:$V_i\neq V_j$.

定义 1. 相关顶点. 由顶点$\{u\}\cup\Gamma(u)$形成的导出子图 $G(u)$中,u 被称为相关顶点.

按照定理 3 分解的子集 $V_1,V_2,\cdots,V_k$ 中,相关顶点只可能存在于某个分片中,不可能被多个分片共享. 也就是说,子图 $G(v)$的顶点集 V_k 中,$V_k-\{v\}$ 都是顶点 v 的邻接顶点,即任何一个顶点 $u\in V_k$,$(u,v)\in E$ 成立. 这样的顶点 v 也称为 V_k 的相关顶点.

因此,可以采用自顶向下的方法来分解原始图,使其形成多个子图. 首先,对于图 G 中的顶点 V,任意选择一个顶点 v 为相关顶点,形成两个顶点的集合 V_1 和 V_1',其中:$V_1=\{v\}\cup\Gamma(v)$,$V_1'=(V-V_1)\cup\{v_i\mid(v_i,v_k)\in E\wedge v_i\in V_1\wedge v_k\in V-V_1\}$;然后在$V_1'$中选择一个相关顶点 w,再对 V_1'进行分解,得到两个顶点的集合 V_2 和 V_2',$V_2=\{w\}\cup\Gamma(w)$,$V_2'=(V_1'-V_2)\cup\{v_i\mid(v_i,v_k)\in E\wedge v_i\in V_2\wedge v_k\in V_1'-V_2\}$;依次不断地递归,第 k 次对 V_k 进行分解,直到 V_k 为空. 集合 $V_1,V_2,\cdots,V_k$ 的相关顶点构成集合 S,其构成图 G 的 k 个分片,表示为:$G(v)=(N(v),E(v))$,其中,$N(v)=\{v\}\cup\Gamma(v)$;

$$E(v)=\{(u,v)\mid u,v\in N(v),(u,v)\in E\}$$

$$G(S)=\bigcup_{v\in S}G(v)$$

每个重叠子图只包含一个相关顶点. 图 G 中的其它顶点被称为不相关顶点.

定义 2. 间接顶点. 无向连通图 $G(V,E)$ 中，如果存在两点 $u\in V$，$v\in V$，满足：(1) $(u,v)\notin E$；(2) 至少存在一个顶点 $w\in V$，使得 $(u,w)\in E$ 且 $(w,v)\in E$ 成立，那么就称顶点 w 是顶点 u 和顶点 v 的间接顶点. 后面的论述中，若无特别说明，默认讨论的是无向连通图.

根据定义 1 的思想，如果能够将图分解成 k 个子集，并且每个子集都包含一个相关顶点，那么就可以通过计算 k 个相关顶点形成的重叠子图中的极大团，得到原图 $G(V,E)$ 中的全部极大团.

接下来的问题是如何得到这些相关顶点的集合. 假如相关顶点之间是不存在边的，那么就可以认为相关顶点集合与图的独立集是相同的. 但是，这个概念不正确，它们是有区别的，例如，如图 2 所示，如果选择相关顶点为 1、5 和 6，它也是独立集，但是，按照这个相关顶点集合计算极大团时可能导致某些极大团的遗漏问题. 按照相关顶点集合 1、5 和 6，将图分为三个重叠子图 $V_1=\{1,2,3\}$，$V_2=\{3,4,5\}$，$V_3=\{2,4,6\}$，采用算法 1 来计算图 2 的极大团时，就会遗漏极大团 $\{2,3,4\}$，遗漏的根本原因是极大团 $\{2,3,4\}$ 是由 3 个重叠子图的共享边组合而成.

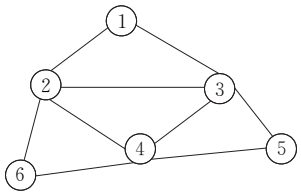


图 2　图的分解

采用图的分片不但满足并行执行的需要，还减少了不相关顶点的计算过程，但随之而来的是极大团的遗漏问题. 所以，接下来讨论如何采取策略解决该问题.

定义 3. 顶点涂色的颜色. 设有两种颜色 c 和 μc，它们组合在一起形成一个有序对，我们称其为涂色的颜色. 在这个有序对中，我们称 c 为颜色的全色，μc 称为颜色 c 的半色. 我们称多个这样的有序对组成的集合为涂色的颜色集.

定义 4. 顶点的涂色覆盖. 设 $C=\{\langle c_i,\mu c_i\rangle\}$ 是覆盖颜色的集合. 集合 V_I 是无向图 $G(V,E)$ 的顶点集合 V 的子集，其满足：(1) 顶点集合 V_I 中的任意一个顶点都被 C 中的一个全色涂色，对于任何两个不同的顶点 $u\in V_I$ 和 $v\in V_I$，u 和 v 的涂色一定不一样. (2) 顶点集合 $V-V_I$ 中的每个顶点都必须被 C 中的一个半色涂色. (3) 对于 V_I 中的任何一个顶点 $u\in V_I$，u 被涂成一个全色 c_i，那么，如果在 $V-V_I$ 存在一个顶点 v，并且 $(u,v)\in E$，那么顶点 v 一定被覆盖为 c_i 的半色 μc_i；或者存在顶点 $u,w\in V_I$，其涂色分别为全色 c_i 和 c_j $(i\neq j)$，并且存在 $(u,v),(w,v)\in E$，那么 v 要么被涂为半色 μc_j，要么被涂为半色 μc_i. (4) 如果在 $V-V_I$ 中存在两个顶点 v、u，并且 $(u,v)\in E$，那么顶点 v,u 的涂色要么必须为相同的半色，要么只能有一个顶点存在于 $V-V_I$，即只能有一个顶点是半色. 此时，V_I 被称为顶点的涂色覆盖.

顶点涂色覆盖的物理意义是，对于图 $G(V,E)$ 中的任何一条边，不允许出现该边的两个端点被涂成两种不同的半色.

针对图 2 中的连通图，给出了一个顶点涂色覆盖后的结果. 如图 3 所示. 在图 3(a) 中，集合 $V_I=\{2,5\}$，集合 $V-V_I=\{1,3,4,6\}$. 顶点 2 和顶点 5 为两种不同的全色，顶点 1,3,4 和 6 为顶点 2 的半色，它们的半颜色是相同的.

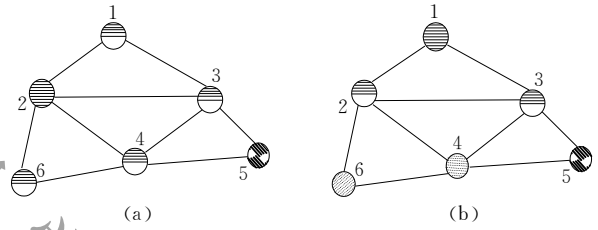


图 3　顶点的涂色覆盖

如果将图中的全色顶点作为相关顶点集合，分别用每个全色顶点对应的子图来计算极大团，遗漏问题得以解决. 例如图 3 中的顶点 2，可以得到极大团 $\{1,2,3\}$，$\{2,3,4\}$，$\{2,4,6\}$，通过顶点 5，可以得到极大团 $\{3,4,5\}$，得到了全部的极大团.

定义 5. 可涂色的颜色数. 如果一个连通图 $G(V,E)$，根据定义 4，使用了 k 种覆盖颜色，那么我们就说图 $G(V,E)$ 是 k 色涂色.

图 3(a) 的颜色数量为 2，即顶点 1 为一种全色，顶点 5 为另外一种全色.

对于图 2，如果采用另外一种颜色涂色方法，那么就得到图 3(b) 的覆盖结果. 顶点 1,4,5 和 6 分别涂一种全色，而顶点 2 和 3 涂为顶点 1 对应的半色. 此时，可以得到图 3(b) 的颜色数为 4.

从图 3 的例子可以看出，选择不同的全色顶点，那么就会需要不同的颜色数，也就是说顶点涂色覆盖 V_I 的规模不确定，图 3(a) 需要 2 个全色，图 3(b) 需要 4 个全色.

定义 6. 顶点涂色覆盖规模问题 (Indirect-Cover). 给定一个连通无向图，找到一个规模最小的

涂色覆盖的颜色数量,即顶点子集的最小值,称这是一个顶点涂色覆盖规模问题.

针对顶点的涂色规模问题,采用强力搜索策略可以得到最优涂色结果,其包含 3 个步骤:

(1)对图中的顶点执行排列;

(2)对每一个排列,依次选择一个顶点:如果该顶点未涂色,则对其涂全色,并对未涂色的邻接顶点涂上半色,然后将该边放入已涂色集合中;如果该顶点为半色,当它的邻接顶点也为相同的半色或者全色时,将该边放入已涂色集合中;如果该顶点为全色,则继续选择下一个顶点执行上述过程;

(3)顶点遍历完成后,计算全色顶点数量;接着,选择下一个排列,继续执行步骤(2).

上述过程可以得到一个最小规模的顶点涂色覆盖,但是复杂度为 $O(E|V|!)$. 当顶点的数量超过 20 时,几个小时内都无法计算出结果.

定理 4. 顶点涂色覆盖规模问题是 NP 完全的.

把顶点涂色覆盖这一最优化问题,可重新表述为一个判定问题,即确定一个无向图是否具有一个给定规模为 k 的顶点涂色覆盖. 作为一种语言,作如下的定义:

IndirectCover=$\{\langle G,k\rangle$:图 G 存在一个规模为 k 的顶点涂色覆盖$\}$.

(证明参考附录).

4.2 顶点涂色覆盖问题的近似算法

顶点涂色覆盖问题要求在一个给定的无向连通图中,找到一个最小规模的覆盖颜色,这样的一个颜色集合被称为顶点涂色覆盖的一个最优的颜色集合,也称为最优的顶点集合.

虽然在一个图 G 中寻找最优顶点覆盖颜色集合比较困难,但是要找到一个近似最优的顶点覆盖颜色集合则相对容易. 下面给出的近似算法以一个无向连通图 G 为输入,返回涂色颜色规模不超过最优涂色颜色规模两倍的结果.

算法 2. Approx-IndirectCover(G)

1. $C=\varphi$
2. $E'=E(G)$
3. WHILE $E'\neq\varphi$
4. $e=(u,v)$是 E'中的任意一条边
5. 顶点 u 和 v 覆盖不同的全色 c_u和 c_v
6. $C=C\cup\{c_u,c_v\}$
7. u 的邻接顶点 $\Gamma(u)$全部覆盖半色 μc_u
8. v 的邻接顶点 $\Gamma(v)$全部覆盖半色 μc_v
9. 删除 E'中与 u 和 v 邻接的全部边
10. 将与 $\Gamma(u)$和 $\Gamma(v)$邻接的边插入 E''中
11. WHILE $E''\neq\varphi$
12. $e=(u,v)$是 E''中的任意一条边
13. 顶点 u 和 v 覆盖不同的全色 c_u和 c_v
14. 删除 E''中与 u 和 v 邻接的全部边
15. $C=C\cup\{c_u,c_v\}$
16. RETURN C

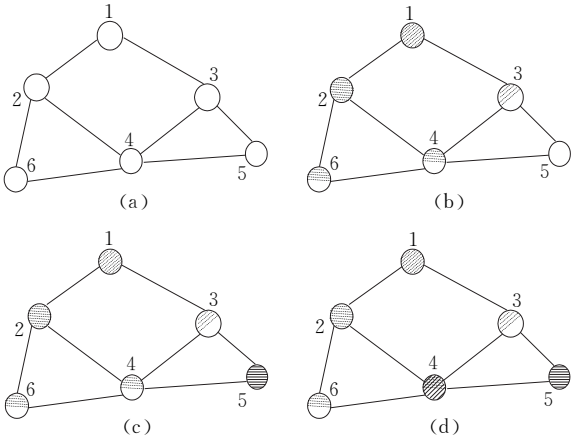


图 4 近似多色顶点覆盖例子

下面以图 4 为例,讲述 Approx-IndirectCover 的执行过程. 变量 C 包含了正在构造的顶点颜色有序对集合. 第 4 行选择顶点 1 和 2 之间的边,执行第 5 行,分别覆盖两种全色;然后将顶点 1 和 2 之间的边、顶点 1 和 3 之间的边、顶点 2 和 6 之间的边以及顶点 2 和 4 之间的边从 E'删除,由于顶点 6 和 4 具有相同的半色,所以再将顶点 6 和 4 之间的边也从 E'中删除;再将顶点 3 和 4 之间的边、顶点 3 和 5 之间的边以及 4 和 5 之间的边插入到 E''中. 再次循环时,E'中已经不存在边,因此第一个循环结束;第二个循环中,从 E''选择顶点 4 和 5 之间的边,执行第 13 行,删除顶点 4 和 5 之间的邻接边;接下来的循环中,由于 E''为空,循环结束,得到子集$\{1,2,4,5\}$.

该算法的运行时间为 $O(E)$,其中 E'采用邻接表来表示.

定理 5. Approx-IndirectCover 有一个多项式时间的 2 近似算法.

(证明参考附录).

4.3 基于顶点涂色分片的极大团挖掘算法

基于多色顶点覆盖的极大团查找算法是对算法 1 的优化,目的在于减少冗余的计算过程. 算法首先采用顶点涂色覆盖算法得到全部的全色顶点,将全色顶点作为相关顶点集合,然后计算出每个顶点的导出子图 $G(v)$,即 G 的重叠子图,也称为涂色分片,最后对每个涂色分片,采用串行算法,得到全部的极大团.

由于无法得到最小的全色顶点集合,只能得到

近似解，而近似解中，全色顶点之间就存在包含关系，即两个全色顶点 v_i 和 v_j 之间存在边，故 v_i 的重叠子图 $G(v_i)$ 中包含 v_j，而 v_j 的重叠子图 $G(v_j)$ 中也包含 v_i. 此时，如果利用近似解计算极大团时，不可避免地会出现冗余计算问题. 例如，采用图 3(b)的全色顶点集合 1,4,5 和 6 来计算极大团时，就会出现冗余团{2,4,6}和{3,4,5}.

出现冗余的根本原因是重叠子图中出现了一个以上的全色顶点，为了去掉冗余，需要从重叠子图中去掉多余的全色顶点. 通过设置颜色的偏序关系 $c_1 \prec c_2 \prec \cdots \prec c_k$ 后，再根据颜色的偏序关系来设置每个重叠子图中的去除顶点集合. 若 $G(v_i)$ 中包含 v_j，v_i 和 v_j 之间存在关系 $c(v_i) \prec c(v_j)$，则 $G(v_i)$ 的去除顶点集合就变更为 $DEL(v_i)=DEL(v_i)\cup\{v_j\}$. 通过设置去除顶点集合，减少了冗余极大团的出现. 例如，图 3(b)的全色顶点 1,4,5 和 6 中的颜色序列为 $c(1) \prec c(4) \prec c(5) \prec c(6)$，计算顶点 4 的重叠子图的极大团时，由于 $DEL(4)=\{5,6\}$，所以只能得到{2,3,4}，计算顶点 5 的重叠子图的极大团时，顶点 6 不在 5 的重叠子图中，所以 $DEL(5)=\varphi$，能得到{3,4,5}，同理，顶点 6 的极大团为{2,4,6}.

算法 3 描述了利用涂色分片思想设计的一个并行挖掘大图 G 中全部极大团的算法.

算法 3.　vCliques($G(V,E)$)

输入：连通图 G，颜色的拟序集合 $c_1 \prec c_2 \cdots \prec c_n$

输出：图 G 中的全部极大团

过程：vCliques(G)

　　//获得全部的全色顶点集合

1. $SUBV$=Approx-IndirectCover(G)；
2. FOR EACH $v \in SUNV$
3. $G(v)=(V_v,E_v)$，$V_v=\{v\}\cap\Gamma(v)$，
   $E(v)=\{(u,v)\mid u\in V(v)\wedge v\in V(v)\wedge(u,v)\in E\}$
4. $DEL(v)=\{v_i\mid(v,v_i)\in E\wedge c_{v_i}\prec c_v\}$
5. $Q=\varphi$；$SUBG=\varphi$；$CAND=\varphi$；
6. $SUBG=V_v\cap\Gamma(v)$
7. $CAND=SUBG-DEL(v)$
8. $Q=Q\cup\{v\}$

   //调用算法 1 中的 *getCliques* 过程

9. getCliques($SUBG$,$CAND$)
10. RETURN

算法 3 中，第 1 行得到顶点涂色覆盖，第 3 行计算每个涂色分片，第 4 行根据颜色的偏序关系，计算去除顶点集合，第 6、7 行计算出涂色分片的顶点集合和候选顶点集合，第 9 行执行 getCliques 过程.

4.4　并行算法复杂度分析

根据文献[18]的结论，对于包含有 n 个顶点的图 $G(V,E)$，使用非并行算法来挖掘全部极大团的时间复杂度是 $O(3^{n/3})$，它的时间复杂度与顶点数量有关，算法也是目前最快的. 为了利用并行计算，大图数据通常按照顶点分解为 n 个重叠子图，然后对每个子图进行极大团的挖掘. 假设 $G(V,E)$ 中顶点度的最大值为 d，那么最坏情况下，并行计算复杂度为 $O(n+n\times 3^{d/3}/p+n\times d\times\mu)$，其中 n 是顶点数量，表示计算重叠子图的时间，p 代表机器数量，μ 则表示每个顶点所属极大团的数量. n 个顶点中每个重叠子图的复杂度与其顶点最大度 d 有关，即为 $O(3^{d/3})$，过滤冗余极大团的复杂度为 $O(n\times d\times\mu)$，若对顶点排序，则可以节省过滤冗余极大团的时间. 当采用涂色分片算法时，涂色分片过程是串行执行，其近似算法的时间复杂度为 $O(E)$. 分片后图的全色顶点数量不超过 $n/2$，最坏情况下并行算法时间复杂度为 $O(E+n/2+n/2\times 3^{d/3}/p)$. 由于涂色分片算法减少了重叠子图的数量，故提高了计算的效率.

定理 6.　算法 3 可以得到连通图 G 的全部极大团.

(证明参考附录).

5　极大团并行挖掘算法的实现

论文中的计算平台采用 Master-slave 模型，一台 Master 机器用于对图数据的顶点涂色分片，多台 Slave 机器用于执行极大团的查找过程. Master 节点执行时，得到每个重叠子图中的相关顶点. 然后将各个重叠子图信息发送到不同的 Slave 机器. Slave 机器在收到执行请求后，在本地执行极大团的查找过程，并将结果返回给 Master 机器.

平台采用 JAVA 编程语言，使用并发、分布式和容错的 akka①[30] 工具包来建立计算平台，编写了一个基于 Master/Slave 模式的计算程序，实现了论文中所描述的算法. 过程 2 和 3 描述了 Master 端程序和 Slave 端程序的逻辑，其并行计算过程如下.

过程 2.　Master 机器的执行过程.

输入：大图数据

输出：大图中包含的全部极大团

1. class CFindMaster extends UntypedActor{
2. 　onReceive(Object message){

① http://doc.akka.io/docs/akka/2.2.5/AkkaJava.pdf?_ga=1.46999460.2066384711.1487575280

```
3.    IF (message instance of String){
4.      P=vCliques1(G);
5.    FOR EACH p∈P
6.      worker=Actors.actorOf(new
                CFindCliques (p)).start();
7.    }ELSE{
8.      output clique;
9.    }
10.  }
11. }
```

master 程序中的第 4 行 vCliques1 方法，它是对算法 3 中的涂色分片功能（第 1,3,4,5,6,7 和 8 行代码）的封装，即得到全部的分片集合；Slave 机器上的 vCliques2 方法，则是将算法 3 中的第 9 行进行封装，其目的是计算各个分片中的极大团.

当 CFindMaster 接收到类型为 String 的消息时，执行 vCliques1 得到涂色分片集合，其中每个涂色分片中，有且只有一个顶点为分片的核心，该顶点一定被涂为全色，分片中的其它顶点一定是核心顶点的邻接顶点. 过程 2 的第 5 行，对每一个涂色分片子图，第 6 行启动一个 Actor 来执行极大团的挖掘. 当 CFindMaster 接收到其它消息时，输出挖掘出的极大团.

过程 3. Slave 机器的执行过程.

输入：重叠子图、相关顶点及去除顶点

输出：重叠子图中的极大团

```
1. class CFindCliques extends UntypedActor{
2.   void onReceive(Object o){
3.     String result=vCliques2(o);
4.     replyUnsafe(result);
5.   }
6. }
```

当 Slave 收到 Master 发送来的消息后，就执行 vCliques2 过程，即调用极大团挖掘算法，得到涂色分片中的极大团. 计算结束后，Slave 机器将计算结果返回给 Master 机器.

6 实验评价

在建立测试用的集群平台时，极大团计算相关的 Actor 代码由自己实现. 平台使用了 16 台中科曙光服务器 620/420，操作系统为 redhat Enterprise server 7.1 X86_64，JAVA 执行环境采用 JDK8u131 的 64 位版本虚拟机，每台物理服务器安装 2 个 8 核的 AMD Opteron（TM） Processor 6212 处理器，16GB内存. 测试时，在每台机器上可以启动多个 JVM 进程并行运行.

6.1 实验数据

实验数据来自于斯坦福大学的大图数据库[29]和 ER 模型产生的随机图，表 1 给出了测试用的图结构以及图包含的特性. 其中图 soc-sign-epinion[31]、soc-epinions[32]、loc-gowalla[33] 和 soc-slashdot0902[34] 是社会网络，顶点代表用户，边代表朋友关系. web-google[34] 是 Web 网页关联图，顶点代表网页，边代表超链接. Wiki-talk[35] 图中的顶点代表用户，而边代表对其它用户的评价页的修改. As-skitter[36] 代表的是一个因特网的路由图. 为了进行极大团的挖掘，这些图都按无向图进行处理，并且去掉了度为 2 的顶点和其关联的边. UG100k.003 是一个随机图，该图包含 100000 个顶点，边的生成概率为 0.0003. 另一个随机图 UG1k.30 中顶点的数量是 1000，边的生成概率是 0.3.

表 1 实验中的图结构数据说明

编号	图名称	顶点数	边数	极大团数	最大度	平均度
1	UG1k.30	1000	149831	15112758	350	299.50
2	soc-epinions	75879	405740	1680933	3044	10.70
3	soc-slashdot0902	82168	504230	642132	2552	12.30
4	UG100k.003	100000	14997891	4488630	379	300.20
5	soc-sign-epinion	131828	711210	22067495	3558	10.79
6	loc-gowalla	196591	980327	1005048	14730	9.60
7	web-google	875713	4322051	939059	6332	9.90
8	as-skitter	1696415	11095298	35002548	35455	13.10
9	wiki-talk	2294385	4659565	83355058	100029	3.90

表 2 给出了各个图在进行涂色分片前后包含的相关顶点数（涂全色的顶点数）以及涂色分片过程需要花费的时间. 从表中可以看出，涂色分片所花费的时间基本都在秒级范围，与极大团的计算时间相比，涂色分片时间所占比例非常小. 另外，分片时间与边的数量之间呈线性关系，边的数量越多，分片时间越长. 观察表 2 中涂色后全色顶点数一栏，可看出涂色分片后的相关顶点数明显减少，其减少的顶点数比例大约为 30% 左右；而对于顶点的平均度较小的图，如 wiki-talk，其相关顶点减少的比例多达 90%以

上，其它几个图的相关顶点减少比例在40%～60%范围内. 平均来看，只有50%左右的顶点需要进行计算，而另外50%左右的顶点则不需要启动进程计算. 为了描述顶点减少的效果，采用压缩比 ρ 来描述，其中 $\rho=|V1|/|V2|$，$|V1|$ 是表2中涂色后全色顶点数，$|V2|$ 是表2中涂色前顶点数即全部顶点数，从表2中数据可以看出，压缩比基本上在50%左右.

表 2　涂色前后顶点数对比

编号	图名称	涂色前顶点数	涂色后全色顶点数	压缩比	分片时间/s
1	UG1k.30	1000	712	0.712	0.332
2	soc-epinions	75879	34202	0.451	0.130
3	soc-slashdot0902	82168	38206	0.465	0.169
4	UG100k.003	100000	86391	0.864	3.210
5	soc-sign-epinion	131828	60055	0.456	0.240
6	loc-gowalla	196591	112861	0.574	0.323
7	web-google	875713	419886	0.480	1.775
8	as-skitter	1696415	875906	0.516	1.622
9	wiki-talk	2294385	104963	0.045	3.115

6.2　并行挖掘算法的性能测试

为了比较不同算法在分布式环境下的运行时间，论文比较了传统的BK串行算法、本文中的并行算法以及涂色分片后并行算法的运行时间，采用分布式模式来运行表1中的图数据，表3为运行结果. 测试时，Master机器按照轮循法将顶点分配到不同的Slave机器上，各Slave机器接收到子图后，启动actor进行计算，并记录开始时间，所有任务结束再次记录时间，两个时间差为运行时间. 从表3中可以看出，BK串行算法时间最长. 不进行涂色分片直接并行计算极大团，运行时间较好. 涂色分片后并行计算，运行时间最短.

表 3　不同图数据的运行时间/s

算法	机器数	图标号								
		1	2	3	4	5	6	7	8	9
串行算法	1	14210	1046	375	*	12081	1620	6818	*	*
未涂色算法	4	3684	324	150	35213	3924	413	2080	35553	23702
	8	1827	171	123	18532	2180	231	1092	19789	13178
	15	887	103	84	10983	1368	127	683	12425	7031
涂色分片算法	4	2578	226	108	24661	2786	285	1518	24887	16827
	8	1431	121	101	14825	1637	178	772	15835	9885
	15	785	85	79	8789	1141	104	520	10532	5631

表3中有3种极大团枚举算法，使用BK串行算法，只有一个JVM，而使用未涂色分片算法和涂色分片算法时，Slave机器的数量分别为4，8和15. 表3中对应的数据代表运算时间，表中 * 的含义是计算时间太长，即在10 h内无法计算出结果.

从表3中的数据可以看出，并行的未涂色算法与并行的涂色算法相比，涂色算法的运行时间普遍较短. 使用未涂色分片算法时，4台服务器的运行时间是串行算法的30%左右，其原因是并行运行带来性能的提高；8台服务器时运行时间更短，大约是串行算法运行时间的20%左右，原因是并行度的增加，更加充分地利用了计算资源，执行效率得到提升；15台服务器时运行时间更短，大约是串行算法运行时间的14%左右. 从运行时间看，15台服务器并没有导致大幅度地缩短时间，原因在于有些服务器计算结束的较早，而有些服务器计算结束的较晚，影响了最后的结束时间，导致运行时间变长. 而涂色分片算法与未涂色分片算法比较后显示，不论4台服务器、8台服务器或者15台服务器，性能都得到提升，其性能提升基本在30%左右. 针对不同的图数据，性能提升幅度变化差异较大，分析原因在于负载平衡的问题，即某些服务器中的计算任务结束的慢，导致整个任务完成时间较长.

6.3　负载平衡测试

针对未涂色分片算法和涂色分片算法，采用Master/Slave模型进行计算，Master机器用来向各个Slave机器发送执行任务，并管理各个Slave机器的状态. Slave机器负责具体的团的计算，每台Slave机器上启动1个JVM进程，每个JVM进程负责计算一部分极大团. Master机器上，分别采用轮循法以及任务请求方法将包含各个顶点的重叠子图分配到Slave机器上. 轮循法中，将各个JVM进行排队，

需要进行计算的重叠子图被依次分配到各个不同的JVM进程中,最后一个JVM分配任务后,又从第一个JVM开始循环,该方法保证了每个JVM获得公平的计算任务数.任务请求法的分配原则是先给每个JVM分配一个任务进行计算,然后当某个Slave机器上的JVM计算任务完成,Master再向该Slave机器的JVM发送新的计算任务,持续执行直到所有的任务都完成为止.

图5和图6分别给出了使用轮循法与任务请求法进行任务分配时,对图数据soc-epinions以及web-google进行极大团计算的结果,其中横坐标代表不同机器上的8个不同的JVM进程,纵坐标代表JVM的计算完成时间.图5(a)的数据是采用未涂色算法计算极大团时各个JVM运行时间;图5(b)的数据是采用涂色分片算法时各个JVM的运行时间.图6(a)代表使用未涂色分片算法时,各个JVM计算极大团的时间,图6(b)代表使用涂色分片算法后各个JVM计算极大团的时间.很明显,在负载平衡方面,任务请求法比轮循法效果更好.对于图5(a)中的任务请求法,每个JVM进程的运行时间基本在120 s,第一个JVM进程由于最后一个团的计算花费较多时间,所以结束的相对晚.而图5(a)中的轮循法,各个JVM的结束时间变化非常大,有的结束的较早,而有的结束的较晚.对于图5(b)中的任务请求法和轮循法,除了使用涂色算法减少了计算时间外,每个JVM的计算结束时间与图5(a)中计算结束时间相同.对于图6中的web-google图数据,实验得出的结论与图5的实验得出的结论相同.所以得出结论,在负载平衡方面,不管是涂色算法还是未涂色并行算法,各个JVM的运行时间差距都得到一定的缩小.

由于每个顶点包含的分片大小不等,造成各个任务执行时间长短不一.轮循法一次性将所有任务平均分配到各个JVM上,分配到较多小任务的JVM结束的比较早,而任务请求法是当某个JVM队列中的任务数小于预先设定的阈值时,该JVM向Master节点请求分派任务.该方法既保证了JVM不空转,也满足任务在各个JVM上均匀分配.所以,从图5和图6中,可以看出任务请求法缩小了不同节点之间结束执行时间的差别.但是,每一次任务请求时,都需要经过请求、调度和发送过程,增加了额外开销.相较而言,任务请求法使得各个节点执行时间较均匀,但是额外开销大,而轮循法的各个节点执行时间差别大,但是额外开销小.

相对于未涂色分片算法,本文采用的涂色分片算法,减少了顶点的个数,不管是soc-epinions图还是web-google图,图5和图6的实验结果均显示,CPU的运行时间减少了30%左右,因此,得出结论,涂色分片算法可以降低计算极大团的运行时间.

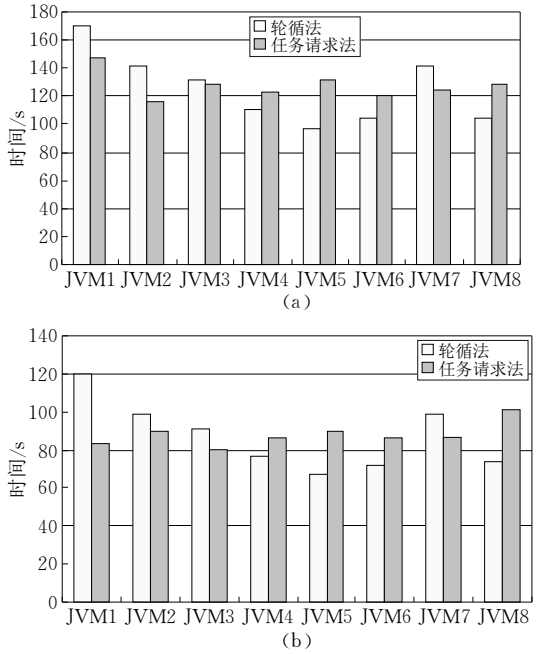


图5 soc-epinions负载平衡结果

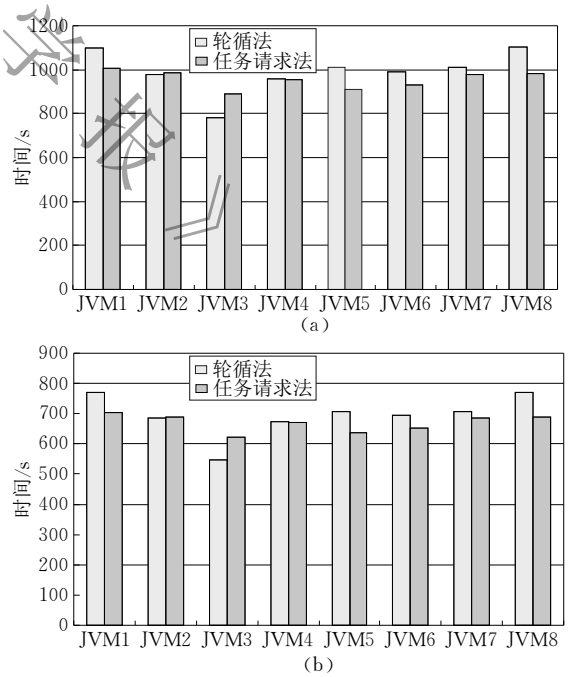


图6 web-google负载平衡结果

运行时间降低的幅度取决于两个因素,非极大团的计算时间以及冗余顶点的计算时间.

6.4 Master-Slave模型的通信量测试

为了测试在Master-Slave模型下Master机器

与 Slave 机器之间通信量，本文针对不同的图数据、涂色分片和不涂色分片策略，测试了执行过程中网络的通信量. 通信量由 3 部分组成：(1) Master 向各 Slave 发送图数据；(2) 各 Slave 向 Master 返回计算结果的数据；(3) 各 Slave 定时向 Master 发送状态数据. 对于任务请求法与轮循法，由于发送和接收的数据大小一样，只是时机不同，因此没有单独进行测试. 表 4 中的数据是在所有 Slave 节点正常运行状况下的 MG 端统计结果，论文只统计了全部通信数据量.

表 4　Master-Slave 模型下的通信量/M

图名称	未涂色分片	涂色分片
UG1k. 30	452. 30	446. 70
soc-epinions	23. 66	19. 70
soc-slashdot0902	19. 17	17. 90
UG100k. 003	170. 20	166. 90
soc-sign-epinion	236. 90	221. 50
loc-gowalla	34. 74	28. 22
web-google	129. 30	116. 70
as-skitter	482. 40	397. 50
wiki-talk	435. 60	315. 50

从表 4 可以看出，图 UG1k. 30 数据量小，每个极大团的顶点数多，通信量大. 图 soc-slashdot0902 包含的极大团数量小，每个极大团包含的顶点也少，通信量较小. 由此可知，Master 和 Slave 之间的数据通信主要与输入文件、极大团的数量以及每个极大团包含的顶点有关.

6.5　并行算法的加速比和效率

加速比经常用来衡量串行算法和并行算法的关系，它表示为串行时间与并行时间的比值：$S(n,p)=T_s(n)/T_p(n,p)$，其中 n 表示输入数据规模，p 表示并行处理器的数量，T_s 表示串行计算的时间，T_p 表示并行计算的时间. 论文中对输入的图数据 soc-epinions 和 web-google 使用两种算法分别测试，JVM 数量依次从 1,8,16 增加到 120，通过增加并行进程数(一台服务器上启动多个独立运行的 JVM 进程)的方式增加处理器的数量来展示算法的加速比. 图 7(a)和 7(b)显示了对图数据 soc-epinions 和 web-google 执行不同算法的加速比. 观察实验结果，两个图数据都显示出了较好的加速度，当 JVM 数量较少时，获得了近似于线性的加速比. 随着 JVM 的数量增加，加速比的增长趋势变得缓慢，JVM 数量约为 80 时，加速比达到最大值，此后加速比回落，分析后发现，随着 JVM 的增加，通信延迟、调度处理以及 JVM 内存回收开销增多，导致并行执行时间增加. 从图 7 显示的数据可以看出，图 soc-epinions 看起来比图 web-google 有些许变化，因为存在顶点涂色的代价对总运行时间的影响.

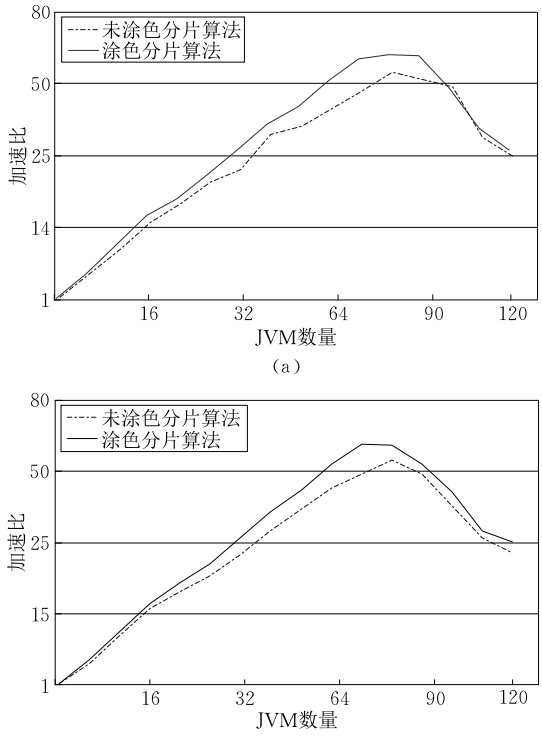


图 7　并行算法加速比

另外，为了说明极大团挖掘过程中“每一个 JVM”的加速比，论文测试了并行算法的效率，用 $E(n,p)$ 表示，$E(n,p)=S(n,p)/p$，通常并行效率都会小于 1. 图 8(a)和 8(b)分别对图数据 soc-epinions 和

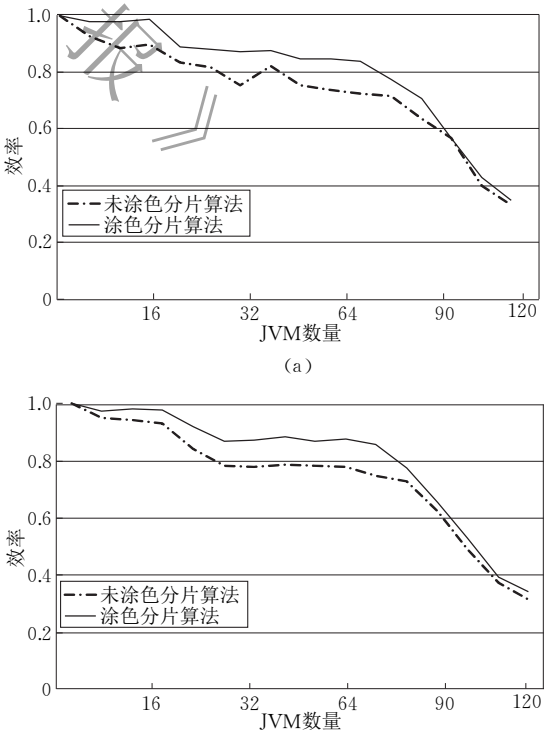


图 8　并行算法的效率

web-google 的并行算法效率进行了测试. 从图 8 显示的曲线看出,并行算法的效率开始的时候下降比较缓慢,在 JVM 数量为 80 左右后开始急剧下降.

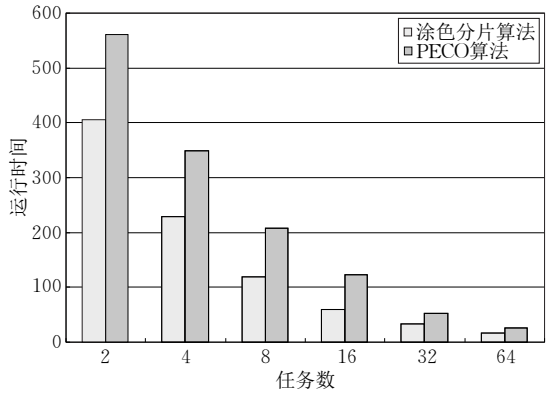
从图 7 和图 8 的结果看出,虽然 web-google 的规模比 soc-epinions 的规模大,但是它计算出来的极大团数量远远小于 soc-epinions 的极大团数量,所以两者加速比区别不是特别明显.

从图 7 和图 8 的结果看出,使用顶点涂色分片算法,其加速比曲线好于未涂色分片算法,但是,两者的差别不是特别大,说明了顶点涂色算法对加速比的影响比较小.

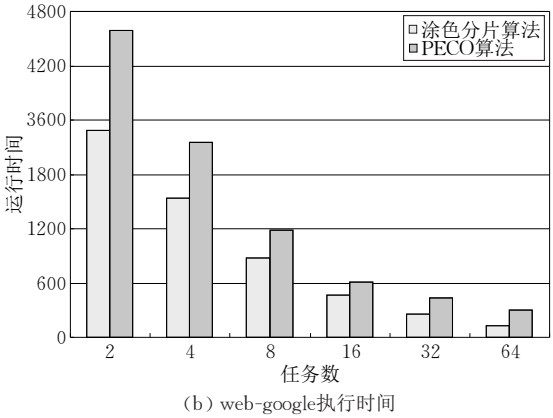
6.6 与其它方法的比较

论文还验证了文献[23]中的算法 PECO,一个基于 MapReduce 的并行 MCE 算法,并且比较了该算法与本文中算法的性能. PECO 算法把输入的图数据分割成多个子图,然后独立地处理每个子图,以便得到极大团. 对于这个实验,使用了基于 Hadoop 2.6 建立起来的集群,其中包括 16 台曙光 620 服务器,每台服务器安装 2 个 8 核的 AMD 处理器和 16 GB 的 RAM.

从图 9(a)中可以看出两个算法的运行时间的对比,对于 PECO 算法,图中的任务数指的是 Reduce 任务数量,它从 8 变化到 64;而对于涂色分片算法,任务数指的是 JVM 进程的个数,它也从 8 变化到 64. 在 soc-epinions 图数据上,使用 64 个任务,涂色分片算法比 PECO 算法快 1.5 倍左右,在任务数为 16 时,涂色分片算法比 PECO 算法大约快 2 倍(两种算法都使用相同排序规则来计算团的数量). 本文算法的优势在于减少了计算过程中的通信时间以及非极大团的计算时间,另外,本文中的算法可以很好的控制各个任务之间的负载平衡,使得算法可以高效地使用更多的处理器.



(a) soc-epinions执行时间



(b) web-google执行时间

图 9 soc-epinions 和 web-google 执行时间

相同的结论也在图数据 web-goole 上得到证明(图 9(b)所示). 从两个图的结果可以看出,涂色分片算法在计算大图数据的极大团时,比单纯的 MapReduce计算模型更能节省计算资源和计算时间.

7 结 论

极大团挖掘在许多领域中都有重要的应用. 顺序极大团挖掘算法得到了广泛而深入的研究,随着图数据规模的扩大,并行极大团挖掘算法显得越来越重要. 论文通过顶点的涂色后分片策略,解决了现有算法中大量的非极大团计算问题,通过对包含几百万顶点的图数据进行测试,算法都能够提高 30% 左右的效率. 但是,算法在分片规模上还有待进一步的提高,另外,还应当尽量减少计算过程中的负载不平衡问题.

致 谢 诚挚感谢评阅老师对论文提出的改进意见.

参 考 文 献

[1] Galaskiewicz J, Wasserman S. Social Network Analysis. Sociological Methods & Research, 1993, 22(1): 3-22

[2] Leskovec J, Lang KJ, Dasgupta A, Mahoney MW. Statistical properties of community structure in large social and information networks//Proceedings of the 17th International World Wide Web Conference. Beijing, China, 2008, 695-704

[3] Harary F, Ross IC. A procedure for clique diction using the group matrix. Sociometry, 1957, 20(3): 205-215

[4] Haraguchi M, Okubo Y. A method for pinpoint clustering of web pages with pseudo-clique search//Proceedings of the International Conference on Federation Over the Web. Berlin, Germany, 2005, 3847: 59-78

[5] On B W, Elmacioglu E, Lee D, et al. Improving grouped-entity resolution using quasi-cliques//Proceedings of the International Conference on Data Mining WorkShops. Hong Kong, China, 2006: 1008-1015

[6] Wang J, Zeng Z, Zhou L. CLAN: An algorithm for mining closed cliques from large dense graph databases//Proceedings of the International Conference on Data Engineering, IEEE Computer Society. Atlanta, USA, 2006: 73-82

[7] Zhang Y, Abu-Khzam F N, Baldwin N E, et al. Genome-scale computational approaches to memory-intensive applications in systems biology//Proceedings of the ACM/IEEE SC 2005 Conference. Seattle, USA, 2005: 12

[8] Grindley H M, Artymiuk P J, Rice D W, et al. Identification of tertiary structure resemblance in proteins using a maximal common subgraph isom-orphism algorithm. Journal of Molecular Biology, 1993, 229(3): 707-721

[9] Pavlopoulos G A, Secrier M, Moschopoulos C N, et al. Using graph theory to analyze biological networks. BioData Mining, 2011, 4(1): 10

[10] Augustson J G, Minker J. An analysis of some graph theoretical cluster techniques. Journal of the ACM, 1970, 17(4): 571-588

[11] Horaud R, Skordas T. Stereo correspondence through feature grouping and maximal cliques. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989, 11(11): 1168-1180

[12] Zomorodian A. The tidy set: A minimal simplicial set for computing homology of clique complexes//Proceedings of the ACM Symposium on Computational Geometry. Snowbird, Utah, USA, 2010: 257-266

[13] Zaki M J, Parthasarathy S, Ogihara M, et al. New algorithms for fast discovery of association rules//Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining. New York, USA, 1997: 283-286

[14] Rossi R A, Gleich D F, Gebremedhin A H, et al. Parallel maximum clique algorithms with applications to network analysis and storage. Siam Journal on Scientific Computing, 2014, 37(5): C589-C616

[15] Battiti R, Protasi M. Reactive local search for the maximum clique problem 1. Algorithmica, 2001, 29(4): 610-637

[16] Rossi R A, Gleich D F, Gebremedhin A H, et al. A fast parallel maximum clique algorithm for large sparse graphs and temporal strong components. arXiv: 1302.6256(2013): 1-9

[17] Bron, Coen, Kerbosch, Joep. Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM, 1973, 16(9): 575-576

[18] Tomitaa E. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science, 2006, 363(1): 28-42

[19] Cazals F, Karande C. A note on the problem of reporting maximal cliques. Theoretical Computer Science, 2008, 407(1): 564-568

[20] Eppstein D, Löffler M, Strash D. Listing all maximal cliques in sparse graphs in near-optimal time. Computer Science, 2010, 6506: 403-414

[21] Kose F, Weckwerth W, Linke T, et al. Visualizing plant metabolomic correlation networks using clique—Metabolite matrices. Bioinformatics, 2001, 17(12): 1198-1208

[22] Moon J W, Moser L. On cliques in graphs. Israel Journal of Mathematics, 1965, 3(1): 23-28

[23] Zhang Y, Abu-Khzam F N, Baldwin N E, et al. Genome-scale computational approaches to memory-intensive applications in systems biology//Proceedings of the ACM/IEEE Conference on Supercomputing. Seattle, USA, 2005: 1-12

[24] Du N, Wu B, Xu L, et al. A parallel algorithm for enumerating all maximal cliques in complex network//Proceedings of the IEEE International Conference on Data Mining Workshops. Hong Kong, China, 2006: 320-324

[25] Lu L, Gu Y, Grossman Y, et al. A distributed algorithm to enumerate all maximal cliques and maximal clique distribution//Proceedings of the IEEE International Conference on Data Mining Workshops. Sydney, Australia, 2010: 1320-1327

[26] Wu B, Yang S, Zhao H, et al. A distributed algorithm to enumerate all maximal cliques in MapReduce//Proceedings of the 4th International Conference on Frontier of Computer Science and Technology. Washington, USA, 2009: 45-51

[27] Schmidt M C, Samatova N F, Thomas K, et al. A scalable, parallel algorithm for maximal clique enumeration. Journal of Parallel & Distributed Computing, 2009, 69(4): 417-428

[28] Svendsen M, Mukherjee AP, Tirthapura S. Mining maximal cliques from a large graph using MapReduce: Tackling highly uneven subproblem sizes. Journal of Parallel & Distributed Computing, 2015, 79(C): 104-114

[29] Eblen J D, Phillips C A, Rogers G L, et al. The maximum clique enumeration problem: Algorithms, applications and implementations. International Symposium on Bioinformatics Research and Applications, 2011, 13(10): 306-319

[30] Tasharofi S, Dinges P, Johnson R E. Why do Scala developers mix the actor model with other concurrency models?// Proceedings of the European Conference on Object-Oriented Programming. Finland, 2013: 302-326

[31] Mcauley J, Leskovec J. Learning to discover social circles in ego networks//Proceedings of the International Conference on Neural Information Processing Systems. Lake Tahoe, USA, 2012: 539-547

[32] Richardson M, Agrawal R, Domingos P. Trust management for the semantic web. Lecture Notes in Computer Science, 2003, 2870(October): 351-368

[33] Cho E, Myers S A, Leskovec J. Friendship and mobility: User movement in location-based social networks//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Diego, USA, 2011: 1082-1090

[34] Leskovec J, Lang K J, Dasgupta A, et al. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics, 2009, 6(1): 29-123

[35] Leskovec J, Huttenlocher D, Kleinberg J. Signed networks in social media//Proceedings of the Sigchi Conference on Human Factors in Computing Systems. New York, USA, 2010: 1361-1370

[36] Leskovec J, Kleinberg J, Faloutsos C. Graphs over time: Densi_cation laws, shrinking diameters and possible explanations//Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. Chicago, USA, 2005: 177-187

附 录

定理 1. 设图 $G(V,E)$ 中的一个顶点的集合 $Q=\{p_1,p_2,\cdots,p_d\}$ 是一个极大团，那么以 Q 为基础的极大团必定有：

$$Q\subseteq(\{p_1\}\cup\Gamma(p_1))\cap\cdots\cap(\{p_d\}\cup\Gamma(p_d)).$$

证明. 设 $Q=\{p_1,p_2,\cdots,p_d\}$ 是一个团，假设存在一个顶点 $v\in V-Q$，满足 $(v,p_1)\in E,(v,p_2)\in E,\cdots,(v,p_d)\in E$，那么按照团的定义 $\{v\}\cup Q$ 必定是团.

故，$\{v\}\cup Q\subseteq(\{p_1\}\cup\Gamma(p_1))\cap\cdots(\{p_d\}\cup\Gamma(p_d))$ 成立.

下面证明团 Q 是极大的.

假设 $\{v\}\cup Q$ 是一个极大团，但是顶点 $v\notin\{p_k\}\cup\Gamma(p_k)$，$1\leq k\leq d$. 由于 $\{v\}\cup Q$ 是一个团，那么顶点 v 必定与 Q 中的每个顶点相邻，顶点 v 与顶点 p_k 之间必定存在边，所以，假设不成立. 定理 1 得证. 证毕.

定理 2. 算法 1 可以正确且完备地计算出图中的全部极大子团.

证明. 算法 1 中的 getCliques 的第 5 行，取得一个顶点并加入团中，第 7 行和第 8 行，计算该顶点的邻接顶点集合，递归调用 getCliques，再得到一个顶点及其邻接顶点，一直持续到 $SUBG=\varphi$ 为止. 而该过程其实就是对定理 1 中的各个顶点进行逐步求交的过程，即：先计算 $\{p_1\}\cup\Gamma(p_1)$，再计算 $\{p_1\}\cup\Gamma(p_1))\cap\{p_2\}\cup\Gamma(p_2)$，再依次计算.

所以，它可以正确计算出一个极大团，算法 1 中，每个顶点都调用并执行 getCliques 过程，因此，可以得到输入图中的全部极大团. 故：定理 2 成立. 证毕.

定理 3. 给定一个图 $G(V,E)$，$|V|=n$，顶点集合 V 被分解为 2 个子集 V_1,V_2，其中：

$V_1=\{v_1\}\cup\Gamma(v_1)$，$V_2=V-V_1\cup\{v_k\mid(v_k,v_j)\in E\wedge v_k\in V_1\wedge v_j\notin V_1\}$. 则必然存在顶点 $u\in V_2$，使得 $(u,v_1)\notin E$，即在集合 V_2 中必定存在一个顶点 u，顶点 u 和顶点 $v_1\in V_1$ 之间不存在边. 此时，子集 V_1 对应的图 G 中的极大团与子集 V_2 对应的图 G 中的极大团不可能重复.

证明. 需要证明两个问题. 一是 V_1 中的极大团不会与 V_2 中的极大团相同；二是它们各自的极大团之间不存在包含关系.

因为顶点集合 V_1 由 v_1 的邻点组成，所以根据性质 1(给定 $G(V,E)$ 中的一个顶点 v，包含 v 的极大团只能在顶点集合 $\{v\}\cup\Gamma(v)$ 中进行扩张)可知，V_1 中的任何极大团必定包含顶点 v_1. 而 $V_2=V-V_1\cup\{v_k\mid(v_k,v_j)\in E\wedge v_k\in V_1\wedge v_j\notin V_1\}$，所以 V_2 中不可能包含顶点 v_1，V_2 中的极大团不可能包含顶点 v_1，故，V_1 和 V_2 中包含的极大团不会重复.

假设 c_1 是 V_1 中的一个极大团，而 c_2 是 V_2 中的一个极大团，那么根据定理 1 可以得知，c_1 中一定包含顶点 v_1；如果存在极大团 $c_1\subset c_2$，那么 c_2 中也必定包含 v_1. 但是，由于 V_2 中不可能包含 v_1，这显然矛盾，所以 $c_1\subset c_2$ 不成立. 定理 3 得证. 证毕.

定理 4. 顶点涂色覆盖规模问题是 NP 完全的.

把顶点涂色覆盖这一最优化问题，可重新表述为一个判定问题，即确定一个无向图是否具有一个给定规模为 k 的顶点涂色覆盖. 作为一种语言，作如下的定义：

$IndirectCover=\{\langle G,k\rangle:$图 G 存在一个规模为 k 的顶点涂色覆盖$\}$.

下面证明这个问题是 NP 完全的.

证明. 首先来证明 $IndirectCover\in NP$. 假定已知一个图 $G=(V,E)$ 和整数 k，我们选取的整数就是顶点涂色覆盖 $V_I\subseteq V$. 检验算法可以证明 $|V_I|=k$，检查两个条件：(1) 对于每个顶点 $v\in V-V_I$，该顶点一定属于某个顶点的全色涂色的一种半色；(2) 对于任意两个顶点 $v\in V-V_I$ 以及 $u\in V-V_I$，如果存在 $(v,u)\in E$，那么对于顶点 v 和顶点 u 来说，它们必然具有相同的半色. 也就是说，任意一个顶点 $v\in V-V_I$，它一定与 V_I 中的某个顶点之间存在边，另外，如果 $V-V_I$ 中有两个顶点是两种不同颜色的半色，它们之间不存在边.

对于一个给定的子集 V_I，确认 V_I 中的顶点与 $V-V_I$ 中的顶点之间是否存在边，同时确认若 $V-V_I$ 中任意两个顶点之间有边，它们是同样的半色. 针对这样的过程，我们很容易在多项式时间内验证这个问题. 所以，我们判定 $IndirectCover\in NP$.

我们用 $3SAT$ 问题证明 $3SAT\leq_P IndirectCover$ 关系成立. 由于 $3SAT$ 是 NP 完全问题，所以我们通过规约判断 $IndirectCover\in NP$ 成立.

要证 $3SAT\leq_P IndirectCover$. 任给 $3SAT$ 的一个实例，它由合取范式 $F=C_1\wedge C_2\wedge\cdots\wedge C_m$ 和变元 $x_1,x_2,\cdots,x_n$ 组成，其中 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$，每个 $z_{jk}(1\leq k\leq 3)$ 为某个 x_i 或 $\neg x_i$. 把这个实例映射到 IndirectCover 的实例 $f(I)$，$f(I)$ 由图 $G=(V,E)$ 和 $K=n+3m$ 构成，其中：

$V=V_1\cup V_2\cup V_3$，$E=E_1\cup E_2\cup E_3$.

$V_1=\{x_i,\bar{x}_i\mid 1\leq i\leq n\}$，

$V_2=\{(x_{jk},j)\mid 1\leq j\leq m,1\leq k\leq 3\}$，

$V_3=\{y_{jk}\mid 1\leq j\leq m,k=1,2,3\}$，

$E_1=\{\{x_i,\bar{x}_i\}\mid 1\leq i\leq n\}$，

$E_2=\{\{(x_{jk},j),z_{jk}\}\cup\{(x_{jk},j),y_{jk}\}\mid 1\leq j\leq m,k=1,$

2,3},

$E_3=\{\{y_{j1},y_{j2}\},\{y_{j2},y_{j3}\},\{y_{j1},y_{j3}\}\mid 1\leq j\leq m\}$,

$F=(x_1\vee\neg x_3\vee\neg x_4)\wedge(\neg x_1\vee x_2\vee\neg x_4)$.

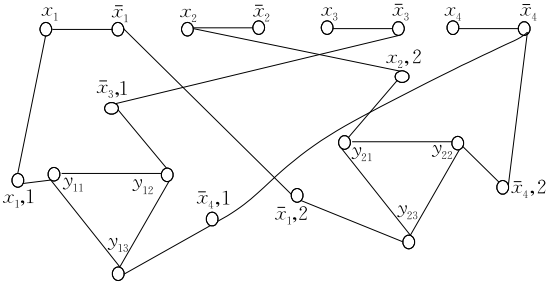
当 $z_{jk}=x_i$ 时,$y_{jk}=x_i$;当 $z_{jk}=\bar{x}_i$ 时,$x_{jk},j=\bar{x}_i$.

图 G 可以分为 3 个部分,每一部分都有各自的功能.第一部分,对于每一个变元 x_i,G 有两个顶点 $x_i,\bar{x}_i$ 和一条边 $\{x_i,\bar{x}_i\}$,为了满足顶点的涂色覆盖,即顶点 $x_i,\bar{x}_i$ 之间必须存在一个全色,也即顶点涂色覆盖中必须包含顶点 x_i 或者 $\bar{x}_i$,分别对应于赋值 $t(x_i)=1$ 或者 $t(\bar{x}_i)=0$.如果 $x_i,\bar{x}_i$ 具有相同的半色,这就要求顶点 $x_i,\bar{x}_i$ 之间存在一个共同的顶点,而根据我们构造的图结构,在 $x_i,\bar{x}_i$ 之间不可能存在一个共同顶点,该顶点与 $x_i,\bar{x}_i$ 都存在边,所以 $x_i,\bar{x}_i$ 之间一定要有一个顶点为全色覆盖.

第二部分表示与 V_1 中顶点之间的边的连接关系.对于每一个简单的析取式 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$,$G$ 有三条边$((x_{j1},j),y_{j1})$,$((x_{j2},j),y_{j2})$,$((x_{j3},j),y_{j3})$对应,同时还包含三条边$((x_{j1},j),x_1)$,$((x_{j2},j),x_2)$,$((x_{j3},j),x_3)$,这三条边与变量对应.这六条边中,每两条边之间都共用一个顶点(x_{j1},j),它们分别形成三条路径.为了对这三条路径中的顶点涂色,针对每一条路径,它由 3 个顶点组成,因此,要么对公共顶点(x_{j1},j)覆盖全色,要么对公共顶点(x_{j1},j)涂上一种半色,当公共顶点为半色时,路径上的两个端点就必须被涂上两种不同的全色,否则无法满足顶点涂色覆盖问题的要求.

第三部分表示与 V_2 中顶点之间的边的连接关系.它是三条边的集合 E_3.E_3 中的 3 条边将 V_3 中顶点之间进行相互连接.同时 V_3 中每个顶点与 V_2 中的 3 个顶点形成一一映射关系,即它们形成一些特殊的边,这些特殊的边与 F 有关.对于每一个 C_j,把对应的 C_j 中的文字 z_{jk} 与特殊边$((x_{jk},j),y_{jk})$对应起来,对于每一个使得文字 z_{jk} 为真和假的情况,分别设置特殊边$((x_{jk},j),y_{jk})$包含的两个端点之间的一个顶点为全色覆盖.V'能否满足顶点涂色覆盖,取决于对应的赋值 t 是否使 F 为真.

因此为了满足顶点涂色覆盖,V'至少要包含 V_1 中的 n 个顶点以及 V_2 和 V_3 顶点之间的特殊 3 条边中的一个顶点,即 $3m$ 个顶点,共计 $n+3m$ 个顶点.由于限制 $|V'|\leq K=n+3m$,故可以满足要求.附图 1 给出了一个例子.



附图 1　把 3SAT 规约到 IndirectCover

附图 1 给出了对应的转换无向图,这里 $n=4$,$m=2$,$K=10$.图 G 用来说明从 3SAT 到 IndirectCover 的规约过程.

设 t 是 F 的成真赋值.对于每一个 $i(1\leq i\leq n)$,若 $t(x_i)=1$,则取顶点 x_i;若 $t(x_i)=0$,则取顶点 $\bar{x}_i$.这 n 个顶点保证了 x_i 和 $\bar{x}_i$ 之间存在边,两个顶点中只有一个在 V' 中出现.对于每一个 $j(1\leq j\leq m)$,由于 $t(C_j)=1$,C_j 中至少有一个文字 z_{jk} 的值为 1.于是必须保证边$(x_j,\bar{x}_j)$之间的顶点 x_j 为全色覆盖,另外,还需要保证边 V_2 从对应的特殊边$((x_{jk},j),y_{jk})$的两个顶点中引入保证顶点被覆盖的一个全色顶点 y_{jk},并且该顶点的全色覆盖使得 V_3 中另外两个顶点是相同的半色覆盖,这样就保证解决覆盖 V_3 中的所有顶点的多色覆盖问题.如果 C_j 中至少有一个文字 z_{jk} 的值为 0,那么必然使得对应的边$(x_j,\bar{x}_j)$的另外一个顶点 $\bar{x}_i$ 是全色覆盖,另外,还需要保证边 V_2 从对应的特殊边$((x_{jk},j),y_{jk})$的两个顶点中引入保证顶点被覆盖的一个全色顶点(x_{jk},j),这样就能保证与 V_3 中的顶点满足多色覆盖要求.因此,得到的 $n+3m$ 个顶点是 G 的一个间接顶点覆盖.

反之,设 $V'\subseteq V$ 是 G 的一个顶点覆盖且 $|V'|\leq K=n+3m$.根据前面的分析,每一对 x_i 和 $\bar{x}_i$ 之间需恰好有一个顶点属于 V',这样才能保证顶点 x_i 和 $\bar{x}_i$ 之间的多色覆盖.特殊边$((x_{jk},j),y_{jk})$的 2 条边恰好有 3 个顶点属于 V'.对于每一个 $i(1\leq i\leq n)$,若 $x_i\in V'$,则令 $t(x_i)=1$;若 $\bar{x}_i\in V'$,则令 $t(x_i)=0$.对于任何一个 C_j,不妨设(x_{j2},j),$(x_{j3},j)\in V'$.而顶点 y_{j2} 与(x_{j2},j)之间的边也满足涂色要求,从而 $t(z_{j1})=1$,$t(C_j)=1$.因此,t 是 F 成真的赋值.

以上的过程证明了 F 是可满足的当且仅当 G 有不超过 $K=n+3m$ 的间接顶点覆盖.

若 G 有 $2n+6m$ 个顶点以及 $n+9m$ 条边,G 和 K 都能在多项式时间内构造出来.从而,这是一个 3SAT 到 Indirect-Cover 的多项式时间变换.

故,IndirectCover 是一个 NP 完全问题.　　证毕.

定理 5.　Approx-IndirectCover 有一个多项式时间的 2 近似算法.

证明.　前面,已经说明 Approx-IndirectCover 的运行时间为多项式;

由 Approx-IndirectCover 返回的覆盖顶点颜色的集合 C 是一个顶点颜色有序对集合,这是由于算法会一直循环,直到 $E(G)$中边的顶点都被 C 中的某个颜色涂成对应的半色或者全色为止.

为了说明 Approx-IndirectCover 所返回的多色顶点覆盖的规模至多为最优覆盖的 2 倍,设 A_1 表示 Approx-Indirect-Cover 的第 4 行选择出来的边的集合.为了将边 A 的两个顶点邻接的所有顶点涂色,任意一个顶点涂色覆盖(尤其是最优顶点涂色覆盖 C_1^*)都必须选择一个顶点为全色.如果一条边在第 4 行被选中,那么第 9 行就会将 E' 中与其邻接的所有边删除,第 10 行将邻接边的邻接边插入 E'' 中,因此 A_1 中就不可能存在任何一个被涂色的顶点(包括全色和半色),从而在 A_1 中就不可能存在重复半色涂色的情况.第一个循环执行完成后,E'' 中的边的两个顶点一定是由两种不同的半色涂色的.此时,为了满足顶点涂色覆盖中的两个半色顶点之间不存在边的要求,则需要选择将这类边转换为一个顶点全

色涂色，另外一个顶点半色涂色，设 A_2 是第二个循环中的第 12 行选择出的 E'' 中的一条边，因此任何一个多色顶点覆盖（尤其是最优多色顶点覆盖 C_2^*）都必须选择一个顶点为全色. 如果一条边在第 12 行被选中，那么第 14 行就会从 E'' 中删除与其邻接的所有边，此时，A_2 中不会存在两条边具有共同的端点，从而 A_2 中不可能存在两条边由 C_2^* 中的颜色共同覆盖. 于是最优多色顶点覆盖的下界如下：

$$|C_1^*|+|C_2^*|\geq|A_1|+|A_2|$$

算法每次执行到第 4 行时会挑选一条边，其两个端点都不会在 C_1^* 中，每次执行第 12 行时会挑选一条边，其两个顶点也都不会在 C_1^* 和 C_2^* 中，所以返回一个顶点涂色覆盖的颜色规模的上界为：

$$|C|=2|A_1|+2|A_2|$$

结合这两个式子，就会得到：

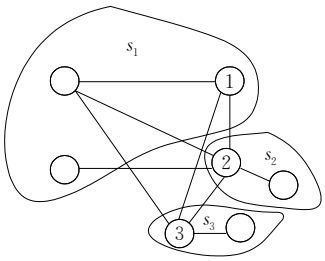
$$|C|=2|A_1|+2|A_2|\leq 2(|C_1^*|+|C_2^*|)$$

因此，定理成立. 证毕.

定理 6. 算法 3 可以得到连通图 G 的全部极大团.

证明. 算法 3 中，使用了 getCliques 过程，该过程可以保证得到包含在邻接顶点组成的导出子图中的全部极大团. 需要证明的是，只对这些全色顶点集合中的顶点进行搜索，可以得到整个图中的全部极大团.

首先证明完全问题. 假设存在一个极大团顶点 q，它无法通过顶点涂色覆盖中的某个顶点计算其极大团. 将会存在以下几种情况：情况(1)，存在这样一个极大团，它包含的顶点是由相同的半色覆盖 uc_i 组成. 这种情况下，这些顶点对应的全色 c_i 的顶点为 w，由于该极大团的各个顶点都是顶点 w 的邻接顶点，那么顶点 w 与该极大团的各个顶点之间存在边. 此时，极大团 q 再添加一个顶点 w 后一定会形成一个新的极大团，这与 q 是极大团矛盾，故 q 不是极大团. 情况(2)，存在这样一个极大团，它包含的顶点是由不同的半色覆盖 $uc_1, uc_1, \cdots, uc_k$ 覆盖，如附图 2 所示. 附图 2 中的顶点有 3 种半颜色，分别属于区域 S1，S2 和 S3. 它们组成了一个极大团 q，由于 q 中包含不同的半色，所以它不可能属于区域 S1，S2 和 S3 中的某个极大团的子团（如果属于区域 S1，S2 和 S3 的某个极大团的子团，那么这些顶点应该属于相同的半色）. 如果存在这样的极大团，按照顶点涂色覆盖的定义，半色顶点 1 和 2，1 和 3 以及 2 和 3 之间必定有边存在，由于存在边，而边的两个端点半色又不一样，这不满足多色顶点覆盖定义，因此，这些边上必须存在一个顶点 w，它是一种全色覆盖. 由于顶点 w 的存在，极大团 q 不可能被遗漏. 故，利用各顶点涂色覆盖计算的极大团必定是完全的.



附图 2 顶点的半色涂色形成的团

下面证明冗余问题. 首先，通过某个涂色分片计算出的极大团内部不可能出现得到冗余极大团的情况，这个通常由串行算法得到保证. 主要问题在于由两个不同的涂色分片计算得到的极大团集合 P 和 Q 之间是否存在冗余的问题.

对于两个全色顶点 p 和 q，它们之间存在两种关系，顶点 p 和 q 之间不存在边，顶点 p 和 q 之间存在边. 如果顶点 p 和 q 之间不存在边，由于极大团集合 P 中的极大团必定包含 p，极大团集合 Q 中的极大团必定包含 q，此时极大团集合 P 和 Q 不可能存在冗余极大团问题. 如果顶点 p 和 q 之间存在边，由于设置了覆盖颜色的拟序关系，假设 p 的颜色与 q 的颜色存在偏序 $c_p \prec c_q$，此时，通过极大团的颜色偏序关系的设置，使得计算出的极大团集合 P 和 Q 中，P 中的极大团必定包含 p，也可以包含 q，但是，Q 中的极大团必定只包含 q 而一定不包含 p，此时极大团集合 P 和 Q 就不可能存在冗余极大团问题.

故，采用算法 3 得到的极大团是正确的. 证毕.

TANG Xiao-Chun, born in 1969, Ph. D., associate professor. His research interests include graph base management and distributed computing.

ZHOU Jia-Wen, born in 1994, graduated student. His research is big data.

TIAN Kai-Fei, born in 1993, graduated student. His research is big data.

LI Zhan-Huai, born in 1961, Ph. D., professor. His research interests include ocean base management and big data computing.

Background

Perhaps the most elementary dense substructure in a graph, also probably the most commonly used, is a maximal clique. Enumerating all maximal cliques in a graph is known as the maximal clique enumeration problem (MCE). MCE is a fundamental problem in graph analysis, and has been used widely, for instance, in clustering and community detection

in social and biological networks, in the study of the co-expression of genes under stress, in integrating different types of genome mapping data, and other applications in bioinformatics and data mining.

In processing a large graph, it is natural to try breaking up the graph into subgraphs and process the subgraphs by parallel tasks. This approach presents some challenges. First, it is necessary to avoid overlap among different coordinating tasks. The difficulty is that almost every method of dividing a graph for parallel processing for MCE, subgraphs assigned to different tasks will overlap. However, the algorithm should be careful in not to repeat the same work among different tasks, at the same time enumerate all maximal cliques. The second challenge is that the scalability for big graph. Many recent work's strategy is enumerating a set of cliques that are not necessarily maximal, then followed by a post processing step that removes non-maximal cliques and duplicates cliques. Some research is based on the Apriori-like concepts. It takes advantage of the fact that every clique of size k, where $k\geq 2$, is comprised of two cliques of size $k-1$ that share $k-2$ vertices. However, the exponentially increasing time for the computation confines the scale of the graph.

We propose multiple color method to divide the vertexes of graph into two sets which are related vertex set and irrelevant vertex set. The two sets stained by full colors and semi colors respectively. Full colored vertex and its adjacent vertexes compose an overlap subgraph. We proved that this problem of the minimum full colors set is NP complete and have an 2 approximation algorithm for polynomial time. Based on related vertex set, we decompose large graph into lots of overlap subgraphs for finding all cliques in parallel. First it outputs only contain maximal cliques without duplicates cliques, and does not need an additional post processing step. Second it reduces the process of finding non-maximal cliques.

This work is supported by the key research program of Ministry of science and technology China (Project No. 2018YFB1003403).