文章编号：2095-2163(2022)03-0005-07 中图分类号：O157.5 文献标志码：A

动态图染色问题研究

邱雅娴[1]，郝元宵[2]，周军锋[1]，杜 明[1]
（1 东华大学 计算机学院，上海 201620；2 燕山大学 信息科学与工程学院，河北 秦皇岛 066004）

摘 要： 图染色是指在一个连通的无向图中，为每个结点指定一个颜色，使得在任意两相邻结点颜色不相同的前提下，使用最少颜色进行染色的问题。针对已有静态染色方法无法处理动态图，并且已有动态图染色方法效率较差的问题，提出了批量处理更新的高效染色方法，以及保证染色效果的无索引方法，来降低内存消耗。经在 4 个真实数据集上进行实验，从染色质量、染色效率、内存消耗角度验证了本文算法的有效性。

关键词： 图染色；动态图染色；贪心算法

Research on dynamic graph coloring problem

QIU Yaxian[1], HAO Yuanxiao[2], ZHOU Junfeng[1], DU Ming[1]
(1 School of Computer Science and Technology, Donghua University, Shanghai 201620, China;
2 School of Information Science and Engineering, Yanshan University, Qinhuangdao Hebei 066004, China)

【Abstract】 Graph coloring refers to the problem of assigning a color to each vertex of a connected undirected graph, and using the minimum number of colors for vertex coloring under the premise that no two adjacent vertices have the same color. Aiming at the problems that the existing static coloring methods cannot handle dynamic graphs and the efficiency of the existing dynamic graph coloring methods is poor, we first propose a high-efficiency coloring method for batch processing and updating, and secondly, we propose an index less method to ensure the coloring effect to reduce memory consumption. Finally, experiments are conducted based on four real data sets. The experimental results verify the effectiveness of the algorithm in the aspects of coloring quality, coloring efficiency, and memory consumption.

【Key words】 graph coloring; dynamic graph coloring; greedy algorithm

0 引 言

图染色是指为无向图中每个结点分配一个颜色，使得图中任意两个相邻结点具有不相同的颜色[1]。图染色问题应用广泛，是学术界的热点问题之一[2]，可用于核酸序列设计[3]、交通管理[4]、网络频道分配[5]、社团体检测[6]等方面。现实中的图更新频繁，静态图染色算法无法使用，因此研究者们将目光投向动态图染色问题上。

对于静态图染色问题，目前效果最好的是 Global 算法[7-8]。此方法按度大小降序，对结点依次染色，每个结点使用邻居结点未使用的颜色。动态图染色算法主要有：基于色彩饱和度的 DC-Local[9]、基于 OC 图的 DC-Global[10]、基于 bucket 分层的 Small-buckets 算法与 Big-buckets 算法[11]等等。其中，DC-Local 依据贪心策略进行结点着色，该方法会导致色数（色数为染色过程中需要使用的最小颜色数量[12-13]）快速增加。

DC-Global 基于贪心策略，优先选择度大的结点进行染色，每次仅处理一条更新边，处理效率较低[14]。Small-buckets 与 Big-buckets 算法基于桶分层思想，将图中结点划分到一组桶中，桶中结点对应的子图单独染色。该方法在图更新时，不会重新调整相邻结点颜色，导致使用的色数较多，染色质量较低。

本文针对现有动态图染色算法效率低下的问题，提出了基于批量处理的染色算法，即一次处理所有的更新边（包括新增边和删除边）；而针对现有动态图染色算法空间消耗大的问题，在保证染色质量的前提下，提出了去掉索引、降低内存消耗的算法。最后，基于真实数据集，将本文算法与现有算法从染色质量、染色效率、内存消耗 3 个方面进行比较，用以验证本文算法的有效性。

基金项目： 国家自然科学基金（61472339, 61572421, 61272124）；上海市自然科学基金（20ZR1402700）。

作者简介： 邱雅娴（1997-），女，硕士研究生，主要研究方向：大图的可达性查询、图染色技术；郝元宵（1993-），女，硕士研究生，主要研究方向：图染色技术；周军锋（1977-），男，博士，教授，博士生导师，主要研究方向：大图上的查询处理技术、推荐系统关键技术；杜 明（1975-），男，博士，副教授，主要研究方向：自然语言处理、信息查询、数据分析。

通讯作者： 周军锋 Email：zhoujf@dhu.edu.cn

收稿日期： 2021-10-08

1 背景知识

1.1 相关定义

问题定义:给定图 G 和一组新增边和删除边的操作,求得最佳图染色 $\sigma(G)$。

结点的度:给定图 $G=(V,E)$,其中 V 是顶点集,E 是边集。无向图中,用 $deg(x)$ 表示结点 x 的度。

全序排序:给定图 G 的两个结点 u 和 v,当 $deg(u) > deg(v)$ 或 $deg(u)=deg(v)$,且 $id(u) < id(v)$ 时,记作 $u \prec v$,称为全序排序。

OC 图:给定图 $G=(V,E)$,其 OC 图记为 G^*。是指对于图 G 中的任意一条边 $(u,v) \in E$,如果 $u \prec v$,则 G^* 中有一条从 u 到 v 的有向边,否则,G^* 中有一条从 v 到 u 的有向边[15]。

图染色:图 G 的图染色是一个映射函数 $f:V \rightarrow C$。其中 C 代表颜色集,并使得任意两个相邻结点 u、v 满足 $f(u) \neq f(v)$[15]。$f(G)$ 表示 f 中所使用的颜色数量。对于一个结点 $u \in V$,用 $u.color=f(u)$ 来表示通过 f 函数指定 u 的颜色。

色数:色数是指对图 G 染色所需的最少颜色数,记做 $\chi(G)$[15]。

最佳图染色:指对于图 G,使得 $|\sigma(G)|=\chi(G)$ 的染色。其中,$\sigma(G)$ 指对图染色的结果。

1.2 相关算法分析

目前存在的 3 种动态图染色的研究成果如下:

(1)DC-Local。基本思想:新增边 (u,v) 时,若结点 u、v 颜色相同,则对其中色彩饱和度小的结点重新染色;删除边 (u,v) 时,对结点 u、v 都重新染色[15]。重新染色时,尽量减少颜色避免颜色增加。由于只修改更新边的邻接结点及其附近结点,导致该方法无法减少总色数,染色效果差。

(2)DC-Global。此方法针对新增边和删除边,分别采用 DC-Orient-Ins 和 DC-Orient-Del 进行处理。新增边 (u,v) 时,将颜色可能会变的结点加入优先队列 q 中,q 按全序顺序排序。若存在某个结点其多个射入结点的颜色都改变了,则该结点被重复染色[15]。删除边 (u,v) 时,将待染色结点加入 q 中,初始候选染色结点为 $\{u,v\}$,q 中任意结点颜色改变时,其所有射出结点需全部加入 q 中。

DC-Global 处理一条更新边的时间复杂度为 $O(n_\Delta \cdot \log(n_\Delta))$。其中,$n_\Delta$ 为更新该边后需要重新染色的结点数目;空间复杂度为 $O(m+n)$,m 为边数,n 为结点数[15]。该算法的空间复杂度高,效率低。

(3)Small-buckets 和 Big-buckets 算法。这两种算法在色数消耗与图更新后需要重新着色的结点数上做权衡,是一组互补算法[16]。算法思想为:划分为一组桶,将图中结点映射到桶中,每个桶中包含一组结点及边,将所有桶分为 β 层(β 为正整数),每层约有 $n^{1/\beta}$ 个桶,每层中的桶容量相同,第 i 层桶容量约为 $n^{i/\beta}$(n 表示图中当前结点数目)。两种算法的区别在于桶容量:Small-buckets 算法桶容量小,故使用的总颜色数多,但是每次图更新后需要重染色的结点数目少;Big-buckets 算法桶容量大,故使用的总颜色数少,但是每次图更新后需要重染色的结点数目多。但二者都不能保证在图更新过程中,使用的色数最少,且该算法依赖于静态图染色算法,在图更新过程中,多次调用静态图染色算法,使其效率较低。

2 基于批量更新的时间高效策略

2.1 DC-Batch 基本思想

本文基于 DC-Global 算法,提出批量更新的优化策略,并提出 DC-Batch 算法。DC-Batch 基于 OC 图,可一次性处理所有更新边。DC-Batch 算法主要思想为:

(1) 在无向图中新增和删除边。

(2) 更新 OC 图。

(3) 处理度变化的结点。若结点 u 度增大/减小,将其加入 q 中,并遍历其射入/射出结点,全序顺序改变的结点 v,有向边 $v \rightarrow u$ ($u \rightarrow v$) 变为 $u \rightarrow v$ ($v \rightarrow u$),将 v 加入 q。

(4) 将 q 中的结点重新染色。

当图 G* 中任一结点 u 颜色改变时(如图 1),u、v 旁边的数字代表其初始颜色,用 $u.old$、$v.old$ 表示;(1) → (2) 表示 u 的颜色从 1 改变为 2,改变后的颜色用 $u.new$ 表示。

$u.old$、$u.new$、$v.old$ 之间的关系存在以下 7 种情况:

(1)如图 1(a):$u.new=v.old$ 时,v 必须重新染色。

(2)如图 1(b):$u.old < u.new < v.old$ 时,v 可能染色为 $u.old$。

(3)如图 1(c):$u.new < u.old < v.old$ 时,v 可能染色为 $u.old$。

(4)如图 1(d):$u.old > v.old > u.new$ 时,v 不需重新染色。

(5)如图 1(e):$u.old < v.old < u.new$ 时,v 可

能染色为 $u.old$。

(6)如图 1(f)：$v.old < u.old < u.new$ 时，v 不需重新染色。

(7)如图 1(g)：$v.old < u.new < u.old$ 时，v 不需重新染色。

综上所述，当 $u.new \neq v.old$ 时，v 的颜色是否受到 u 颜色变化的影响，取决于 $u.old$ 和 $v.old$。若 $u.old < v.old$ 时，v 的颜色可能改变，否则颜色不会改变；当 $u.new = v.old$ 或者 $u.color \neq v.color$ 且 $u.old < v.old$ 时，v 需要重新染色。

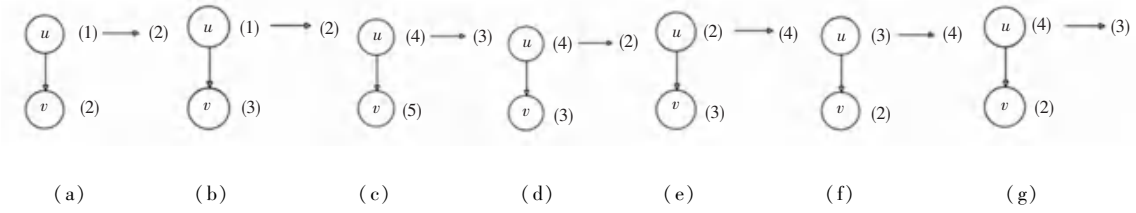


图 1　射出结点的染色

Fig. 1　Coloring of the projected vertex

基于 OC 图的染色方法，重要的是维护 OC 图。插入或删除边时，会使图中结点度发生变化，结点的全序顺序会改变。当结点 u 的度增大或减小时，分别遍历其射入结点或射出结点，若全序顺序改变，则更新 OC 图。

结点 u 的颜色改变时，其射出结点有可能重新染色。结点重新染色步骤如下：

(1)收集射入结点的颜色：对于图 $G*$ 和任意结点 u，其射入结点颜色表示为集合 AC。

(2)给当前结点染色：对于图 $G*$ 和任意结点 u，利用集合 AC 中未使用的最小颜色为结点 u 染色。

(3)射出结点重新染色：若 u 颜色改变，则对其射出结点重新染色(即重复执行(1)、(2)步骤)。

2.2　算法描述

本文提出的 DC-Batch 方法可同时处理新增边和删除边，并在保证染色效果的前提下，提高染色效率。算法 1 给出了基于 OC 图的 DC-BATCH 算法，展示了动态图中结点染色的具体步骤。

算法 1　DC-BATCH 算法

输入：无向图 G，OC 图 G^*，新增边 E_{add}，删除边 E_{del}

输出：无向图 G'

map $m \leftarrow \emptyset$, *PriorityQueue* $q \leftarrow \emptyset$

for each $(u,v) \in E_{del}$ do　/*处理删除边*/

update edge (u,v) of G^* according to the total order

update the degree of u and v in m

remove edge (u,v) from G

process E_{add} like lines 2-5　/*处理新增边*/

for each $< u, num >$ *in* m　/*处理所有度变化的结点*/

$q.push(u)$

if $num > 0$ then　/*处理射入结点*/

for each $u' \in nbr^-(u)$ do

if $u \prec u'$ then

remove edge $< u', u >$ and add edge $< u, u' >$

if $v \notin q$ then $q.push(u')$

else if $num < 0$ then /*处理射出结点*/

process $u' \in nbr^+(u)$ like lines 10-13

while $q \neq \emptyset$ do

$u \leftarrow q.pop()$

$AC \leftarrow \emptyset$

for each $v \in nbr^-(u)$ do　/*收集结点 u 所有射入结点颜色*/

$AC \leftarrow AC \cup \{v.color\}$

$C \leftarrow \{0,1,2\cdots deg^-(u)\}$

$c_{new} \leftarrow \min\{c \mid c \in C, C \notin AC\}$　/*为结点 u 重新染色*/

if $c_{new} \neq u.color$　/*将需重新染色的结点加入 q 中*/

$c_{old} = u.color$

$u.color \leftarrow c_{new}$

for each $v \in nbr^+(u)$ do

if $v \notin q$ and ($u.color = v.color$ *or* $c_{old} < v.color$) then $q.push(v)$

3　基于批量更新的空间高效策略

3.1　算法思想

针对现有动态图染色方法空间消耗大的问题，本文提出 DC-Simple 方法。当新增/删除边时，用 DC-Simple 对部分结点进行重新染色，染色结果与

Global 方法保持一致。对于 OC 图 G^* 中的任一结点 u，有 $u.color = \min\{c \mid c \in N, c \notin \cup_{v \in nbr^-(u)} v.color\}$；对于无向图 G，u 的颜色是全序排序在其之前的邻居结点未使用的最小颜色，即 $u.color = \min\{c \mid c \in N, c \notin \cup_{v<u} v.color\}$。

对于图 G，不构建其 OC 图，当新增或删除一条边 (u,v) 后，则需要染色的结点包含以下 3 部分：

(1)更新边的邻接结点 u 和 v。

(2)更新前，全序排序在 u 和 v 前面，更新后在 u 和 v 后面的结点。

(3)更新前后，全序排序都在 u 和 v 之后的结点。

3.1.1 新增边处理

综合上述 3 种情况，需要重新染色的结点为 $\{\{u,v\} \cup I_{u1} \cup I_{u2} \cup I_{v1} \cup I_{v2}\}$。其中，$I_{u1} = \{x \mid x \prec u \& x, u \in G\} \cap \{x \mid u \prec x \& x, u \in G + (u,v)\}$；$I_{u2} = \{x \mid u \prec x \& x, u \in G\} \cap \{x \mid u \prec x \& x, u \in G + (u,v)\}$；$I_{v1} = \{x \mid x \prec v \& x, v \in G\} \cap \{x \mid v \prec x \& x, v \in G + (u,v)\}$；$I_{v2} = \{x \mid v \prec x \& x, v \in G\} \cap \{x \mid v \prec x \& x, v \in G + (u,v)\}$。

上文讨论了结点 u 的度改变时，其射出结点颜色变化的情况。这部分结点相当于 I_{u2}，所以继续沿用上文的分析结果，讨论当图 G 中任一结点 u 颜色改变时，I_{u1} 中结点的颜色变化情况。如图 2 所示，u 旁边的数字表示其颜色。为表示方便，使用有向边表示全序排序，x 代表 I_{u1} 中的任一元素。如图 2(a)所示，表示 x 结点全序顺序在结点 u 之前。用 $u.old$ 表示结点 u 的初始颜色，用 $u.new$ 表示 u 重染色后的颜色(x 同理)。当任意结点 u 的颜色改变时，射出结点是否重新染色需根据以下情况判定：

(1)如图 2(b)，$u.new = x.old$。此时，x 必须重染色。

(2)如图 2(c)，$u.new < x.old$。此时，x 颜色不改变。

(3)如图 2(d)，$u.new > x.old$。此时，x 颜色不改变。

(a) 更新前 (b) 更新后情况 1 (c) 更新后情况 2 (d) 更新后情况 3

图 2 I_{u1} 中结点重新染色

Fig. 2 Vertices recoloring in I_{u1}

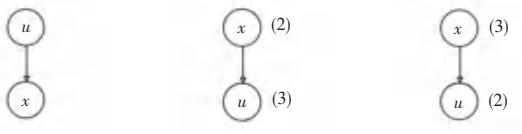
3.1.2 删除边处理

综合上述 3 种情况，需要重新染色的结点为 $\{\{u,v\} \cup D_{u1} \cup D_{u2} \cup D_{v1} \cup D_{v2}\}$。其中，$D_{u1} = \{x \mid u \prec x \& x, u \in G\} \cap \{x \mid x \prec u \& x, u \in G + (u,v)\}$；$D_{u2} = \{x \mid u \prec x \& x, u \in G\} \cap \{x \mid u \prec x \& x, u \in G + (u,v)\}$；$D_{v1} = \{x \mid v \prec x \& x, v \in G\} \cap \{x \mid x \prec v \& x, v \in G + (u,v)\}$；$D_{v2} = \{x \mid v \prec x \& x, v \in G\} \cap \{x \mid v \prec x \& x, v \in G + (u,v)\}$。

根据上述分析，当删除边 (u,v) 时，应先对 u 和 v 的部分邻居结点重新染色，再对 u 结点和 v 结点重新染色。因为更新后，有一些结点全序排序在其前面。D_{u1} 为 u 和 v 的部分邻居结点，u 和 x 相邻，则二者颜色不同，需要重新染色的情况为：

(1)如图 3(b)，$u.old > x.old$。此时 x 的颜色不会改变。

(2)如图 3(c)，$u.old < x.old$，此时 x 可能需要重新染色。



(a) 更新前 (b) 更新后情况 1 (c) 更新后情况 2

图 3 D_{u1} 中结点重新染色

Fig. 3 Vertices recoloring in D_{u1}

DC-Simple 算法对新增边和删除边分别使用 DC-Simple-Ins 和 DC-Simple-Del 来处理。DC-Simple-Ins 思想是，当新增边 $< u,v >$ 时，将结点 u 和 v 放入 q 中，q 依据全序顺序来排序，全序排序在前的结点优先级更高。对 q 中的每个结点 u，遍历其邻居结点并分为 3 部分：

(1)全序排序在 u 之前的结点，收集这些结点的颜色，并找出其未使用最小颜色。

(2)更新前全序排序在 u 之前，更新后全序排序在 u 之后的结点 x。当 x 满足 $u.new = x.old$ 时，加入 q。

(3)更新前后全序排序都在 u 之后的结点 x。当 x 满足 $u.color = x.old$ or $u.old < v.color$ 时加入 q。

3.2 算法描述

DC-Simple 与 DC-Batch 方法最大的区别是未使用 OC 图。DC-Simple 可以在保证染色效果的前提下，降低空间复杂度，解决了 DC-Batch 方法空间复杂度高的问题。算法 2 给出了 DC-Simple-Ins 方法实现过程。

算法 2　DC-Simple-Ins 算法

输入：无向图 G，新增边 E_{add}

输出：无向图 G'

PriorityQueue $q \leftarrow \emptyset$

for each $(u,v) \in E_{add}$ do　/* 处理新增边 */

add edge (u, v) in G

if $u \notin q$ then $q.push(u)$

if $v \notin q$ then $q.push(v)$

while $q \neq \emptyset$ do　/* 对 q 中结点重新染色 */

$u \leftarrow q.pop()$

$availColor \leftarrow \emptyset, S \leftarrow \emptyset, I \leftarrow \emptyset$

for each $x \in$ nbr(u) do /* 遍历结点 u 的所有邻接结点 */

if $x \prec u$ then /* 记录全序顺序大于 u 的结点颜色 */

$availColor = availColor \cup x.color$

else if $x \prec u$ *before update* do /* 更新前全序顺序大于 u 的结点加入集合 I 中 */

$I \leftarrow I \cup x$

else　/* 更新前全序顺序不大于 u 的结点加入集合 S 中 */

$S \leftarrow S \cup x$

$C \leftarrow \{0,1,2\cdots deg^-(u)\}$　/* 寻找结点 u 可以使用的新颜色 */

$c_{new} \leftarrow \min\{c \mid c \in C, C \notin availColor\}$

if $c_{new} \neq c_{old}$ then　/* 当结点 u 重新染色时，将集合 S 中结点加入 q 中 */

$c_{old} = u.color$

$u.color = c_{new}$

for each $v \in S$

if $v \notin q$ and ($u.color = v.color$ *or* $c_{old} < v.color$) then $q.push(v)$

for each $v \in I$ /* 将集合 I 中结点加入 q 中 */

if $v \notin q$ and $(u.color = v.color)$ then $q.push(v)$

无向图 G 及其根据 Global 方法得到的染色结果如图 4 所示。根据 DC-Simple-Ins 算法分析，当在图 G 新增边 (v_5, v_8)，则将 $\{v_5, v_8\}$ 加入 q 中。首先遍历 v_5 的邻接结点，发现 v_5 的全序顺序比其所有邻接结点都大，故 c_{new} 为 0，v_5 无需重新染色。之后遍历 v_8 的邻接结点，由于 v_5 的度大于 v_8，所以 c_{new} 为 1，v_8 可使用的最小颜色为 1，则 v_8 的重新染色为 1。因为 v_8 的颜色改变，所以将 v_9、v_{11} 加入 q 中。然后依次对 q 中的结点重新染色。最终的染色结果如图 5 所示。

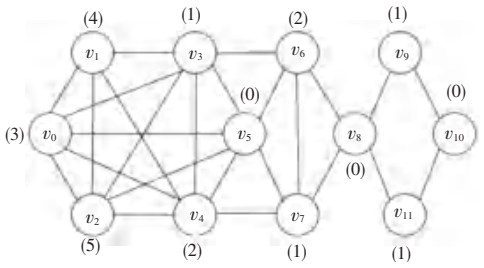


图 4　无向图 G

Fig. 4　Undirected graph G

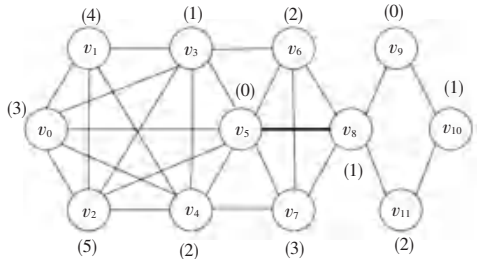


图 5　无向图 $G+(v_5, v_8)$

Fig. 5　Undirected graph $G+(v_5, v_8)$

算法 3　DC-Simple-Del 算法

输入：无向图 G，删除边 E_{del}

输出：无向图 G'

PriorityQueue $q \leftarrow \phi$

for each $(u,v) \in E_{del}$　/* 处理删除边 */

remove edge (u, v) in G

$V \leftarrow V \cup x$

for each $x \in nbr(u)$ do　/* 遍历 u、v 相邻结点，找到需重新染色结点 */

if $u \prec x$ and $x \prec u$ *before update* then

if $x \notin q$ and $u.color < x.color$ then $q.push(x)$

process line 5-7 by replacing u with v

if $u \notin q$ then $q.push(u)$

if $v \notin q$ then $q.push(v)$

while $q \neq \emptyset$ do　/* 对 q 中结点重新染色 */

$u \leftarrow q.pop()$

$availColor \leftarrow \emptyset, S \leftarrow \emptyset$

for each x $\in nbr(u)$ do

if $x \prec u$　then　/* 记录全序顺序大于 u 的结点颜色 */

$availColor = availColor \cup x.color$

else

$S \leftarrow S \cup x$　/* 将全序顺序不大于 u 的结点加入集合 S */

$C \leftarrow \{0,1,2,\cdots,deg^-(u)\}$

$c_{new} \leftarrow \min\{c \mid c \in C, C \notin availColor\}$

If $c_{new} \neq c_{old}$ then　/* 结点 u 重新染色时，将全

序顺序在 u 后的结点加入 q 中 */

$c_{old} = u.color$

$u.color = c_{new}$

for each $v \in S$

if $v \notin q$ and ($u.color = v.color$ or $c_{old} < v.color$) then $q.push(v)$

4 实验结果分析

4.1 实验环境

本文实验环境为:Intel Core i5 主频为 2.50 GHz 的 CPU;Windows 7 的 32 位操作系统;运行内存 4 GB;开发环境为 Microsoft Visual Studio 2013,使用 C++语言实现代码。

4.2 数据集及评价指标

数据集来自 KONECT,本文在 4 个真实数据集上进行实验。其中,MovieLens 为美国的电影评价网站;DBLPwe 为文献网站;Flickr 为社交网络图;Amazon 为美国的电商平台。

表 1 为数据集信息。其中,$|V|$ 为结点数;$|E|$ 为边数;d_{max} 表示图中结点的最大度。本节将从染色质量、染色效率、内存消耗 3 个方面对 DC-Batch、DC-Simple 和 DC-Local、DC-Global 算法进行对比分析。

表 1 真实数据集统计信息

Tab. 1 Real data sets statistics

编号	数据集 G	类型	$\|V\|$	$\|E\|$	d_{max}
1	MovieLens	评分	150 433	10 000 054	34 864
2	DBLP	文献	317 080	1 049 866	343
3	baidu	电商	2 141 300	17 794 839	97 950
4	Flickr	社交网络	2 302 925	33 140 017	34 174

4.3 性能比较分析

4.3.1 染色质量对比分析

在实验中,保留每个数据集 5%边作为初始图,以保证本文算法具有良好的可扩展性。每次增加 30%的边,记录色数。4 个数据集上的色数如图 6~图 9 所示。由图 6 和图 8 可知:对于 MovieLens 和 DBLP,随着边的增加,DC-Local 算法的色数快速增长,其它方法的色数保持稳定。由图 7 和图 9 可知:对于 Baidu 和 Flickr,随着边的增加,DC-Local 与其它算法使用的色数都会快速增加,但 DC-Local 的增加速率最快。

4.3.2 染色效率对比分析

首先使用 Global 算法对每个数据集染色,之后在每个数据集上随机增加和删除 10 000 条边,计算出处理一条边的平均运行时间并进行对比。3 个算法的运行时间如图 10 所示。在所有数据集上,DC-Batch 比 DC-Global 运行时间短,表明批量更新策略更高效。在 MovieLens 上,DC-Simple 运行时间比 DC-Batch 短;在其他数据集上,DC-Simple 和 DC-Batch 运行时间几乎相同,这说明 DC-Simple 效率也比较高。

4.3.3 内存消耗对比分析

内存消耗在 Flickr 和 Baidu 上进行实验,其结果如图 11 所示。可以看出,在 Flickr 上 DC-Simple 与 DC-Batch 的内存消耗峰值分别是 1 000 MB 和 1 200 MB;在 Baidu 上 DC-Simple 与 DC-Batch 算法的内存消耗峰值分别是 8 980 MB 和 1 300 MB。证明 DC-Simple 比 DC-Batch 内存消耗更低。这是因为,DC-Batch 算法基于 OC 图,而 DC-Simple 算法不使用 OC 图,节省了构建 OC 图的内存消耗及 OC 图本身占用的内存消耗。

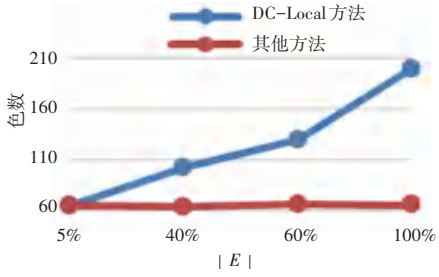


图 6 MovieLens 数据集染色质量图

Fig. 6 Coloring quality chart of MovieLens

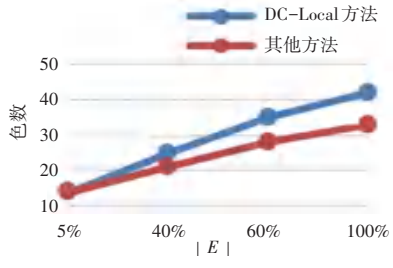


图 7 Baidu 数据集染色质量图

Fig. 7 Coloring quality chart of Baidu

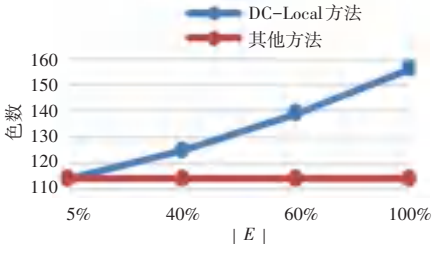


图 8 DBLP 数据集染色质量图

Fig. 8 Coloring quality chart of DBLP

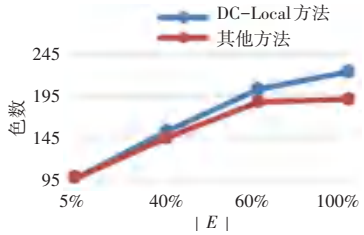


图 9　Flickr 数据集染色质量图

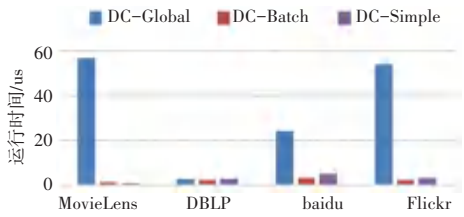


图 10　各算法平均运行时间图

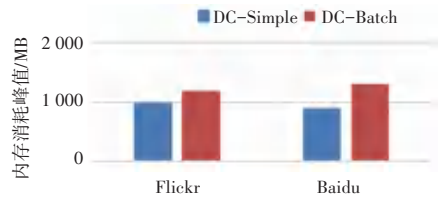


图 11　内存消耗峰值比较图

Fig. 9　Coloring quality chart of Flickr　Fig. 10　Average running time of each algorithm　Fig. 11　Memory consumption peak comparison chart

5　结束语

针对现有算法染色效率低、内存消耗大的问题，本文提出批量处理更新的高效染色方法 DC-Batch。该算法并不关心图更新时的中间状态，而是针对一段时间图更新的最终状态，一次性处理所有的更新，减少重复处理结点的操作，提高处理效率。同时，提出了高效的结点剪枝策略来加速计算，在保证染色质量的前提下，染色效率平均提高 50%。针对已有方法索引占用空间大的问题，提出无需维护额外索引的结点剪枝策略，以及基于此策略的 DC-Simple 方法。该方法针对所有更新边，根据结点的度来动态维护结点的处理顺序[15]。实验结果表明，DC-Simple 方法与 DC-Batch 方法相比，在保证染色效果的同时，内存消耗平均降低 24%。

参考文献

[1] CHARTRAND G, ZHANG Ping. Chromatic Graph Theory[M]. Boca Raton, London, New York: CRC Press, 2009:33-44.

[2] 梁政. 图染色问题应用研究[D]. 江西:江西师范大学,2016.

[3] PENG Y, CHOI B, HE B, et al. VColor: A practical vertex-cut based approach for coloring large graphs[C]//2016 IEEE 32nd International Conference on Data Engineering (ICDE). IEEE, 2016: 97-108.

[4] BARNIER N, BRISSET P. Graph coloring for air traffic flow management[J]. Annals of operations research, 2014, 130(1): 163-178.

[5] BALASUNDARAM B, BUTENKO S. Graph domination, coloring and cliques in telecommunications [M]//Handbook of Optimization in Telecommunications. Springer, Boston, MA, 2006: 865-890.

[6] MORADI F, OLOVSSON T, TSIGAS P. A local seed selection algorithm for overlapping community detection [C]//In Proceedings of ASONAM, 2014:1-8.

[7] ZUCKERMAN D. Linear degree extractors and the inapproximability of max clique and chromatic number [C]//In Proceedings of STOC, 2006:681-690.

[8] WELSH D J, POWELL M B. An upper bound for the chromatic number of a graph and its application to timetabling problems[J]. The Computer Journal, 1967, 10(1):85-86.

[9] PREUVENEERS D, BERBERS Y. Acodygra: an agent algorithm for coloring dynamic graphs [J]. Symbolic and Numeric Algorithms for Scientific Computing, 2014, 6:381-390.

[10] Long Yuan, Lu Qin, Xuemin Lin, et al. Effective and Efficient Dynamic Graph Coloring[J]. PVLDB, 2017, 11(3): 338-351.

[11] Luis Barba, Jean Cardinal, Matias Korman, et al. Dynamic Graph Coloring[J]. Algorithmica, 2019, 81(4): 1319-1341.

[12] 杨兆程. 图染色算法的并行化[J]. 电脑编程技巧与维护,2018(3):121-123,130.

[13] ROKOS G, GORMAN G J, KELLY P H, et al. A Fast and Scalable Graph Coloring Algorithm for Multi-core and Many-core Architectures[C]// European conference on parallel processing, 2015: 414-425.

[14] HERTZ A, DE WERRA D. Using tabu search techniques for graph coloring[J]. Computing, 1987, 39(4): 345-351.

[15] 郝元宵. 动态图染色问题研究[D]. 秦皇岛:燕山大学, 2019.

[16] Shay Solomon, Nicole Wein. Improved Dynamic Graph Coloring [J]. CoRR abs/1904.12427 (2019).

(上接第 4 页)

[9] LIU Xingjie, TIAN Yuan, YE Mao, et al. 2012. Exploring personal impact for group recommendation[C]// In Proceedings of the 21st ACM international conference on Information and knowledge management, 2012:674-683.

[10] Quan Yuan, Gao Cong, Chin-Yew Lin. 2014. COM: a generative model for group recommendation[C]// In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014:163-172.

[11] HU Liang, CAO Jian, XU Guandong, et al. Deep Modeling of Group Preferences for Group-Based Recommendation[C]// In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014:1861-1867.

[12] TRAN L V, PHAM T A N, TAY Y, et al. Interact and Decide: Medley of Sub-Attention Networks for Effective Group Recommendation [C]// the 42nd International ACM SIGIR Conference. ACM, 2019:255-264.

[13] YIN H, WANG Q, ZHENG K, et al. Social Influence-Based Group Representation Learning for Group Recommendation[C]// IEEE 35th International Conference on Data Engineering, 2019: 566-577.

[14] ZHANG J, GAO M, YU J, et al. Double-Scale Self-Supervised Hypergraph Learning for Group Recommendation [C]// In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, 2021:2557-2567.

[15] GUO L, YIN H, CHEN T, et al. Hierarchical Hyperedge Embedding-based Representation Learning for Group Recommendation [J]. ACM Transactions on Information Systems, 2021, 40(1):1-27.