計算機應用
Journal of Computer Applications
ISSN 1001-9081
CN 51-1307/TP

面向动态网络的介数中心度并行算法

刘震宇, 王朝坤, 郭高扬

Parallel algorithm of betweenness centrality for dynamic networks

引用本文:
刘震宇, 王朝坤, 郭高扬 . 面向动态网络的介数中心度并行算法[J]. 计算机应用, 2023, 43(7): 1987–1993.

Journal of Computer Applications
计算机应用,2023,43(7):1987 - 1993
ISSN 1001-9081
CODEN JYIIDU
2023-07-10
http://www.joca.cn

文章编号:1001-9081(2023)07-1987-07
DOI:10.11772/j.issn.1001-9081.2022071121

面向动态网络的介数中心度并行算法

刘震宇,王朝坤*,郭高扬
(清华大学 软件学院,北京 100084)
(*通信作者电子邮箱chaokun@tsinghua.edu.cn)

摘　要:介数中心度是评价图中节点重要性的一项常用指标,然而在大规模动态图中介数中心度的更新效率很难满足应用需求。随着多核技术的发展,算法并行化已成为解决该问题的有效手段之一。因此,提出一种面向动态网络的介数中心度并行算法(PAB)。首先,通过社区过滤、等距剪枝和分类筛选等操作减少了冗余点对的时间开销;然后,基于对算法确定性的分析和处理实现了并行化。在真实数据集和合成数据集上进行了对比实验,结果显示在添加边更新时PAB的更新效率为并行算法中最新的batch-iCENTRAL的4倍。可见,所提算法能够有效提高动态网络中介数中心度的更新效率。

关键词:介数中心度;动态网络;最短距离;并行算法;社区结构
中图分类号:TP311　　**文献标志码**:A

Parallel algorithm of betweenness centrality for dynamic networks

LIU Zhenyu, WANG Chaokun*, GUO Gaoyang
(*School of Software*, *Tsinghua University*, *Beijing* 100084, *China*)

Abstract: Betweenness centrality is a common metric for evaluating the importance of nodes in a graph. However, the update efficiency of betweenness centrality in large-scale dynamic graphs is not high enough to meet the application requirements. With the development of multi-core technology, algorithm parallelization has become one of the effective ways to solve this problem. Therefore, a Parallel Algorithm of Betweenness centrality for dynamic networks (PAB) was proposed. Firstly, the time cost of redundant point pairs was reduced through operations such as community filtering, equidistant pruning and classification screening. Then, the determinacy of the algorithm was analyzed and processed to realize parallelization. Comparison experiments were conducted on real datasets and synthetic datasets, and the results show that the update efficiency of PAB is 4 times that of the latest batch-iCENTRAL algorithm on average when adding edges. It can be seen that the proposed algorithm can improve the update efficiency of betweenness centrality in dynamic networks effectively.

Key words: betweenness centrality; dynamic network; shortest distance; parallel algorithm; community structure

0　引言

在日趋复杂的网络结构中,不同节点在信息流传播时的重要性可能存在较大差异。例如在微博平台上,新闻媒体的“官微”比普通用户有着更强的信息传播能力。因此,如何衡量一个节点在信息传播中的“重要程度”已成为一个重要的研究问题。

节点的介数中心度(betweenness centrality)是Freeman于1977年提出的概念[1],其值大小从最短路径角度刻画了节点作为“桥梁”能力的强弱。介数中心度在度量节点的信息交流能力方面效果突出,目前已被广泛应用于极端恐怖主义追踪[2]、基因与疾病关联性识别[3]和移动网络路由信息分析[4]等众多领域。但鉴于介数中心度的计算复杂度较高,人们一直致力于提升其计算效率。

随着计算机硬件的不断更迭,多核处理器已在服务器和PC设备中越来越常见,这为并行计算提供了有利条件。近年来,已有研究者将并行计算应用于介数中心度的计算问题,主要分为静态计算[5]与动态更新[6-7]两个方向。如今以社交网络为代表的许多网络结构都具有不断演化的特性,因此动态更新受到越来越多的关注。然而,上述与并行化动态更新相关的工作大多数基于分布式架构或GPU,对设备资源的要求较高,不利于应用的普及,而单机多核CPU设备则极为常见。除此之外,单机多核CPU相较于分布式架构能够省去机器间通信的时间开销。因此,本文针对单机多核CPU设备,设计并实现了动态网络中介数中心度的多线程并行算法。

在网络结构中除了节点和边之外,社区作为常见特征之一也备受关注。尽管社区已应用于足球转会市场分析[8]、蛋白质复合物预测[9]等众多领域,但将它用于介数中心度计算的研究目前还较少,相关工作有钱珺等[10]提出的CBU(Community based node Betweenness centrality Updating)算法。该工作提出在网络变化时利用社区过滤能够筛掉大量不受影响的点对,证明了社区在介数中心度动态更新领域具有强

收稿日期:2022-07-12;**修回日期**:2022-08-01;**录用日期**:2022-08-11。　**基金项目**:国家自然科学基金资助项目(61872207)。
作者简介:刘震宇(1999—),男,河南南阳人,硕士研究生,CCF会员,主要研究方向:社交网络、社区发现;　王朝坤(1976—),男,江苏东台人,副教授,博士,CCF高级会员,主要研究方向:社交网络分析、图数据管理、大数据系统;　郭高扬(1994—),男,山西运城人,博士研究生,主要研究方向:社交网络、图神经网络、知识图谱构建。

大的潜力；但是CBU算法未考虑等距性，所以在社区过滤后仍存在大量冗余点对，且在边删除时更新点对间最短路信息的算法效率较低。

目前，尚未有工作将社区与并行计算结合应用于介数中心度的动态更新，因为其中存在诸多难点。例如，网络中的社区大小可能存在巨大差异，因此使用多线程进行并行化时易导致线程负载不均衡。参考图1所示的例子，该图中的网络包含c_1~c_4四个规模不同的社区，$|c_i|$代表社区c_i中的节点数量，图的其他部分暂时略去。当该网络发生动态变化后，需检测各社区对中受到网络变化影响的点对。若此时直接使用多线程对各社区对并行处理，同时假定线程1、线程2被分配到的社区对分别是(c_1, c_4)和(c_2, c_3)，则两个线程之间需要检测的点对数量相差上万倍，极大地影响了并行效率。而除了负载不均衡外，还存在内存访问冲突等问题。

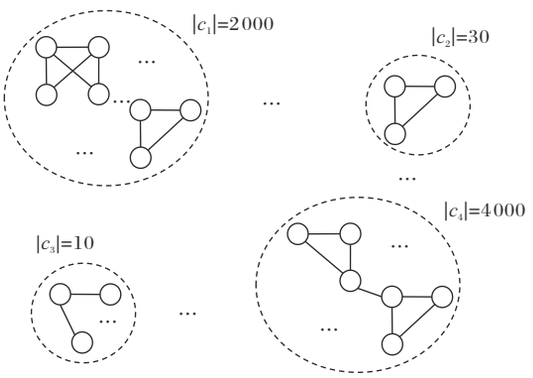

图1 不同大小社区示意图

Fig. 1 Schematic diagram of communities with different scales

针对前述问题，本文结合网络中的社区结构提出一种适用于单机多核CPU的动态网络介数中心度并行算法。

本文的主要工作包括：

1）提出了等距剪枝、分类筛选等点对更新策略。等距剪枝策略可以在社区过滤后直接剪枝最短路不变的点对，加快点对过滤；而根据分类筛选策略，在发生边删除时仅需重新计算原有最短路全部断开的点对，使其他点对的更新得到简化。

2）提出了一种面向动态网络的介数中心度并行算法（Parallel Algorithm of Betweenness centrality for dynamic networks，PAB）。在应用新提出的点对更新策略的基础上，基于共享内存模型实现了算法并行化，能够充分利用单机上的多核CPU。此外还提出了循环独立性的概念，并基于此概念对PAB的并行确定性进行了分析。

基于OpenMP实现了PAB，并在真实数据集和合成数据集上进行了性能对比实验和可扩展性实验。实验结果表明，本文提出的PAB具有较高的更新效率，且可以扩展到大规模数据上。

1 相关工作

随着各个领域网络规模的不断扩增，人们开始将更多的关注放在大量节点中的“重要节点”，而介数中心度是衡量节点重要程度的常用指标之一。由于介数中心度的时间复杂度达到$O(n^3)$，其中n代表网络中节点的数量，无法满足实际需求，因此人们开始致力于提高介数中心度的精确计算效率。Brandes算法[11]将时间复杂度降为了$O(mn)$，是目前静态图上最快且最常用的介数中心度算法，其中m分别代表网络中边的数量。然而，如今很多网络是动态变化的。例如在社交网络中，用户可能随时关注或取关另一位用户，而网络中与之对应的边也会相连接或断开。此时如果使用Brandes算法重新计算整个网络的介数中心度，将带来巨大的时间开销和资源浪费。因此，动态图中介数中心度的更新算法成了研究热点。

现有的介数中心度动态更新算法可按照串行与并行进行分类。串行算法方面，Lee等[12]首先提出了QUBE算法（Quick algorithm for Updating BEtweenness centrality），有选择性地过滤掉了当图中边发生增删变化时介数中心度保持不变的节点；Green等[13]提出了一种利用前期所计算的有效信息来维护最短路数据的算法；钱珺等[10]提出了CBU算法，利用社区过滤掉边更新时不受影响的点对。

与上述串行算法相比，并行算法在计算资源充足时一般有着更高的效率。Jamour等[6]提出了利用分布式计算机集群处理大规模图的算法；Shukla等[7]提出了并行处理批量边更新的算法。然而，这些并行算法都未能充分利用网络中的社区特征。

除了上述研究之外，还有一类工作是对于介数中心度的近似计算[14-15]，即通过牺牲精度来换取在大规模图中的计算效率。由于本文只考虑介数中心度的精确计算，因此不作过多探讨。

2 基本概念

给定一个复杂网络图$G=(V,E,C)$，V是节点集合，$E\subseteq V\times V$是边集合，$C=\{c_1,c_2,\cdots,c_k\}$是G上的社区列表。另外，本文用w表示边的权重，$d(s,t)$表示节点s、t之间的最短距离，$\sigma(s,t)$表示从节点s出发到节点t的最短路径的数量。为了不失一般性，本文假设G是无向无权图（权重$w=1$），且本文的算法可以推广到有向图和有权图。

定义1 介数中心度。给定网络图$G=(V,E,C)$，$v\in V$是G中的任一节点。节点v的介数中心度$c_B(v)$指的是图G中所有点对间最短路中经过节点v的比例，用公式表示为：

$$c_B(v)=\sum_{s\neq t\neq v,s,t\in V}\frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{1}$$

其中：$\sigma(s,t|v)$表示从节点s出发到节点t且途径节点v的最短路径的数量。

定义2 社区-社区最短距离集合。给定网络图$G=(V,E,C)$，$c_1,c_2\in C$，用$d_{cc}(c_1,c_2)$表示社区c_1、c_2间最短距离集合，即：

$$d_{cc}(c_1,c_2)=\{d(v,u)|v\in c_1,u\in c_2\} \tag{2}$$

定义3 社区-节点最短距离集合。给定网络图$G=(V,E,C)$，$c_1\in C$，$u\in V$，用$d_{cv}(c_1,u)$表示社区c_1中所有节点到节点u的最短距离集合，即：

$$d_{cv}(c_1,u)=\{d(v,u)|v\in c_1\} \tag{3}$$

例1 如图2所示，该网络可划分为c_1、c_2、c_3三个社区。社区c_1、c_2间最短距离集合$d_{cc}(c_1,c_2)=\{1,2,3,4\}$；社区$c_2$、节点$v_3$间的最短距离集合$d_{cv}(c_2,v_3)=\{1,2\}$。

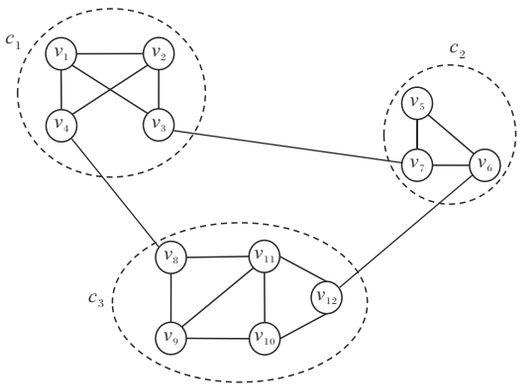

图2　复杂网络里的社区示意图

Fig. 2　Schematic diagram of communities in complex network

3　点对更新策略

因为网络增加或删除边后,并非所有点对间的最短路径都会受到影响,所以有效的点对更新策略是动态网络中介数中心度计算的关键所在,也是PAB的基础。除了已有的社区过滤策略外,本章给出点对更新的两种新策略,即等距剪枝和分类筛选。

3.1　社区过滤

定理1　给定网络图 $G=(V,E,C)$, $c_1,c_2\in C$ 是 G 中的两个社区, $u,v\in V$ 是 G 中的两个节点。向 G 中添加一条权重为 w 的边 (u,v),如果满足 $d_{cc}(c_i,c_j)_{max}<d_{cv}(c_i,u)_{min}+d_{cv}(c_j,v)_{min}+w$,则社区 c_1 内的点到社区 c_2 内的点最短路径均不变。

当动态网络中添加边后需要更新介数中心度时,社区过滤能够以社区为单位过滤掉大量点对。同时,该策略可以应用于删除边情况。有关社区过滤的详细证明可参见文献[10]。

3.2　等距剪枝

当需要对两社区之间所有点对最短路变化情况进行逐一判别时,利用等距剪枝策略可排除大量点对。首先给出策略依据的定理并进行证明,随后提出等距剪枝的内容,最后通过例子说明具体的应用场景。

定理2　给定网络图 $G=(V,E,C)$, $c_1,c_2\in C$ 是 G 中的两个社区。向 G 中添加一条边 (u,v),对于社区 c_1 中的某节点 s,若 $d(s,u)=d(s,v)$,则 s 到社区 c_2 中任意节点的最短路径保持不变。

证明　假设社区 c_2 中存在某节点 t,在添加边 (u,v) 后 s 到 t 的最短路径发生变化。由此可知, s 与 t 之间必然存在经过边 (u,v) 的新最短路径 p_1,可表示为 $(s,a_1,\cdots,u,v,\cdots,a_2,t)$,路径长度可表示为 $d_{p1}=d(s,u)+d(u,v)+d(v,t)$。根据条件 $d(s,u)=d(s,v)$ 可知, s 到 t 之间存在经过 v 且不经过 u 的另一条路径 p_2,可表示为 $(s,b_1,\cdots,v,\cdots,a_2,t)$,路径长度 $d_{p2}=d(s,v)+d(v,t)$。显然 $d_{p1}>d_{p2}$,由此可知 p_1 不是 s 与 t 之间最短路径,假设不成立。

在社区过滤之后,若社区对 (c_1,c_2) 没有被直接过滤掉,则需要对其中的点对一一判断,即进行 $|c_1|\times|c_2|$ 次遍历。此时可根据定理2,直接跳过社区 c_1 中与更新边两端节点最短距离相同的节点,即进行等距剪枝。

例2　如图3所示,向网络中添加边 (v_3,v_4)。当遍历到社区对 (c_1,c_2) 时,它不符合过滤条件,需要逐一判断 (c_1,c_2) 中的点对。当遍历到 c_1 中节点 v_1 时,因为 $d(v_1,v_3)=d(v_1,v_4)=1$,所以节点 v_1 到社区 c_2 中所有节点的最短路径不变。此时可进行剪枝直接跳过节点 v_1,即无需判断点对 (v_1,v_5)、(v_1,v_6)、(v_1,v_7)。

此外,等距剪枝可推广到删除边更新的情况。证明时,只需将删边问题转化为向删边后的图添加边的问题即可。

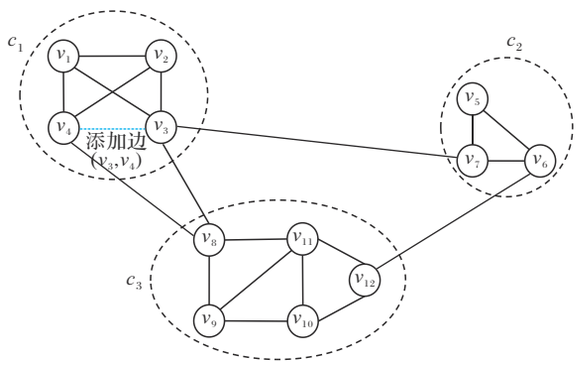

图3　添加边操作示意图

Fig. 3　Schematic diagram of adding an edge

3.3　分类筛选

本节首先给出删除边时点对分类筛选所依据的定理并进行证明,接着提出分类筛选的具体策略,最后给出例子。

定理3　给定网络图 $G=(V,E,C)$。删除 G 中的一条边 (u,v) 后,用集合 *Nodepair* 记录所有最短路发生变化的点对。若点对 $(s,t)\in$ *Nodepair* 满足:

$$\sigma(s,t)=\sigma(s,u)\times\sigma(v,t) \tag{4}$$

则点对 (s,t) 间的原有最短路径全部断开;反之则原有最短路径仅部分断开。

证明　由于点对 (s,t) 之间的最短路在边 (u,v) 删除之后发生变化,因此点对 (s,t) 之间在发生边删除前必然存在经过边 (u,v) 的最短路径,数量是 $\sigma(s,u)\times\sigma(v,t)$。若点对 (s,t) 之间原有的最短路总数 $\sigma(s,t)=\sigma(s,u)\times\sigma(v,t)$,则说明点对 (s,t) 间原有的最短路全部经过边 (u,v);反之则仅部分最短路经过边 (u,v)。

根据定理3,可以在删除边时对 *Nodepair* 中的点对进行分类筛选,即对于原有最短路径全部断开的情况, $d(s,t)$ 和 $\sigma(s,t)$ 重新计算;其他情况下,无需更新 $d(s,t)$,而 $\sigma(s,t)$ 可以直接增量更新。

例3　如图4所示,当删去边 (v_2,v_3) 后,其对应变化点对集合 *Nodepair* 中的点对可进行如下划分:

1)点对间最短路全部断开。例如点对 (v_2,v_7), $\sigma(v_2,v_7)=\sigma(v_2,v_2)\times\sigma(v_3,v_7)$,符合式(4)。此情况下,所删除的边 (v_2,v_3) 是点对 (v_2,v_7) 间所有最短路径的必经之路,因此删除边之后点对 (v_2,v_7) 之间最短路全部断开, $d(v_2,v_7)$ 和 $\sigma(v_2,v_7)$ 都要重新计算(算法见第4章)。

2)点对间最短路仅部分断开。例如点对 (v_4,v_7), $\sigma(v_4,v_7)<\sigma(v_4,v_2)\times\sigma(v_3,v_7)$,不符合式(4)。这说明除了经过删除边的路径,点对间还有其他最短路径。对点对 (v_4,v_7) 来说,最初最短路有 (v_4,v_2,v_3,v_7) 和 (v_4,v_1,v_3,v_7) 这两

条；当删除边(v_2, v_3)后，仅(v_4, v_2, v_3, v_7)这条最短路受影响断开，而(v_4, v_1, v_3, v_7)依旧存在。此时，$d(v_4, v_7)$无需更新，$\sigma(v_4, v_7)$只需减去受删边影响而消失的最短路数量即可完成更新。

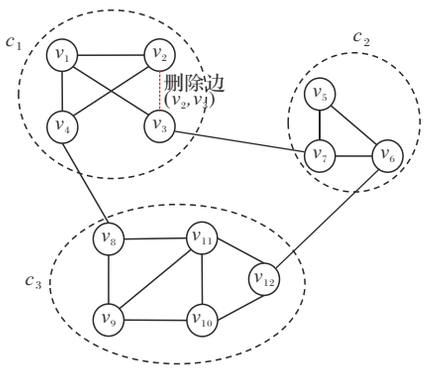

图4 删除边操作示意图

Fig. 4 Schematic diagram of deleting an edge

4 动态网络中介数中心度并行算法

基于上述三种更新策略，本章提出动态网络中介数中心度并行算法PAB，充分利用单机多核CPU的固有并行性。首先给出并行算法的确定性的概念；随后分别在添加边与删除边情况下，对PAB的步骤、确定性和复杂度等方面进行具体分析。

4.1 并行算法的确定性

如果一个并行算法对于相同的输入总会执行相同的任务，且与指令的调度方式无关，则称该并行算法具有确定性[16]。本文的并行化主要针对算法中的循环结构，而若要保证循环的并行确定性，首先需要循环各迭代间不存在顺序依赖；同时由于本文使用共享存储模型OpenMP实现并行化，因此必须避免共享存储区的访问冲突[17]。综上，本文给出了循环独立性的定义，可作为本文并行算法确定性的判断标准。

定义4 循环独立性。给定任一算法中的某循环体结构，若该循环无迭代顺序要求，且不同迭代之间不存在对同一元素的访问冲突，则称该循环具有循环独立性。

4.2 添加边时的并行算法

添加边时并行算法的具体实现如算法1所示。当网络中发生边添加更新后，首先遍历所有的社区对，进行社区过滤（算法1第1）~2）行）。接着以并行方式对无法直接过滤的社区对内部的点对进行检测（算法1第3）行）。显然，选择在此处并行化避免了在社区层面并行时的线程负载不均衡情况。在进行具体检测时，首先通过本文提出的等距剪枝减少循环次数（算法1第5）~6）行）。随后若检测到最短路发生变化的点对，则将该点对和所属社区对分别加入集合*Nodepair*和*Compair*中（第8）~10）行）。完成筛选后，遍历*Nodepair*中每个点对，并行删去这些点对于每个节点介数中心度的旧影响（第11）~15）行）。随后，并行更新*Nodepair*中每个点对的最短路信息（第16）~21）行）。接着仿照删除时操作，重新加上*Nodepair*对每个节点介数中心度新的影响（第22）~26）行）。最后，基于*Compair*并行完成对于d_{cv}和d_{cc}的更新（第27）~28）行）。

算法1 添加边时的介数中心度并行更新算法。

输入 $G=(V, E, C)$, d, d_{cv}, d_{cc}, σ, c_B, (u, v, ω)。

输出 更新后的介数中心度 c_B。

1) for $(c_m, c_n) \in C \times C$
2) if $d_{cc}(c_m, c_n)_{max} \geqslant d_{cv}(c_m, u)_{min} + d_{cv}(v, c_n)_{min} + w$
3) # pragma omp parallel for
4) for $s \in c_m$
5) if $d(s, u) = d(s, v)$ //等距剪枝
6) continue
7) for $t \in c_n$
8) if $d(s, t) \geqslant d(s, u) + d(v, t) + \omega$
9) #pragma omp critical //设置临界区
10) 将点对(s, t)加入集合*Nodepair*，将点对所属社区对(c_m, c_n)加入集合*Compair*
11) for $(s, t) \in Nodepair$
12) # pragma omp parallel for
13) for $r \in V$
14) if $d(s, t) = d(s, r) + d(r, t)$
15) $c_B(r) = c_B(r) - \sigma(s, r) \times \sigma(r, t)/\sigma(s, t)$
16) # pragma omp parallel for
17) for $(s, t) \in Nodepair$
18) if $d(s, t) = d(s, u) + d(v, t) + w$ //d不变
19) $\sigma(s, t) = \sigma(s, t) + \sigma(s, u) \times \sigma(v, t)$
20) else
21) $\sigma(s, t) = \sigma(s, u) \times \sigma(v, t)$,
$d(s, t) = d(s, u) + d(v, t) + w$
22) for $(s, t) \in Nodepair$
23) # pragma omp parallel for
24) for $r \in V$
25) if $d(s, t) = d(s, r) + d(r, t)$
26) $c_B(r) = c_B(r) + \sigma(s, r) \times \sigma(r, t)/\sigma(s, t)$
27) # pragma omp parallel for
28) 基于社区对集合*Compair*对 d_{cv}、d_{cc}进行更新

定义5 并行化循环。对于算法中的某循环结构，利用并行编程指令使该循环中的迭代并发执行，则称新得到的循环为并行化循环。

引理1 算法1中所有的并行化循环都不存在迭代顺序依赖。

证明 从算法1的流程易见，其中所有并行化循环对循环执行的顺序无要求，即执行结果与循环迭代顺序无关，因此得证。

引理2 算法1中所有的并行化循环都不存在不同迭代对同一元素的访问冲突。

证明 对于算法1中第3）~10）行点对检测的循环，由于本文在第9）行使用了OpenMP中的临界区，因此避免了第10）行*Nodepair*和*Compair*进行更新时的访问冲突；对于第11）~15）行删除旧介数中心度影响的循环，由于本文在内层循环实施并行（第12）行），不同线程将分别更新数组$C_B[]$中的不同元素，无访问冲突；第22）~26）行与第11）~15）行情况类似。除以上三处之外，算法1中的其他并行循环不涉及对同一元素的更新操作，因此得证。

定理4 算法1具有确定性。

证明 由引理1和引理2可得，算法1中所有的并行循环具有循环独立性。又因为整个算法1仅针对循环体结构进行了并行化，所以算法1具有确定性。

定理5 算法1的时间复杂度为$O(\rho|V|^3/T)$。其中：T为

程序运行时可用的并行线程数量;ρ为每次边更新时,最短路发生变化的点对数量在所有点对中所占比例。

证明　当边添加操作发生后,通过社区过滤和等距剪枝找到最短路发生变化点对的时间复杂度约为$O(|V|^2/T)$,找到的点对数量为$\rho|V|^2$;接下来针对每个点对进行介数中心度更新、最短路更新等操作的总时间开销近似为$O(\rho|V|^3/T)$。其中:$0<\rho<1$,且通常情况下$\rho>\frac{1}{|V|}$,因此算法1的时间复杂度为$O\left(|V|^2/T+\rho|V|^3/T\right)=O\left(\rho|V|^3/T\right)$。

定理6　算法1的空间复杂度为$O\left(|V|^2\right)$。

证明　算法1需要维护的数据包括网络图的节点与边、节点间最短距离d与最短路数量σ、社区-节点间最短距离d_{cv}、社区-社区间最短距离d_{cc}等数据。因此算法1的空间复杂度为$O\left(|V|+|E|+2|V|^2+|C||V|+2|C|^2\right)=O\left(|V|^2\right)$。

4.3　删除边时的并行算法

删除边时并行算法的部分操作与添加边时类似,具体实现如算法2所示。

算法2　删除边时的介数中心度并行更新算法。

输入　$G=(V,E,C)$, d, d_{cv}, d_{cc}, σ, c_B, (u,v,ω)。

输出　更新后的介数中心度 c_B。

```
1) 执行算法1中第1)~15)行
2) # pragma omp parallel for
3) for (s,t) ∈ Nodepair
4)  if σ(s,t) ≠ σ(s,u)*σ(v,t)                          //分类筛选
5)    σ(s,t) = σ(s,t) - σ(s,u)*σ(v,t)
6)  else
7)    bool find = False                                 //记录s能否到达t
8)    vector <bool> vis                                 //记录图中的节点是否已走过
9)    vector <int> dis, cnt
        //分别记录从s出发到其他节点的最短距离、最短路数量
10)   将vis、dis、cnt大小统一设置为|V|
11)   vis[s] = True, dis[s] = 0, cnt[s] = 1
12)   queue < Node* > Q                                 //设置节点类型的队列
13)   将节点s压入队列Q
14)   while(队列Q不为空){
15)     取出队列Q头节点u
16)     if u = t                                        //成功从s到达t,计算完成
17)       find = True
18)       break
19)     for (u,v) ∈ E                                   //遍历u的邻居节点
20)       if vis[v] = False
21)         vis[v] = True
22)         将节点v压入队列Q
23)       if dis[v] = dis[u] + 1
24)         cnt[v] = cnt[v] + cnt[u]
25)   }
26) if find = True
27)   d(s,t) = dis[t], σ(s,t) = cnt[t]
28) else
29)   d(s,t) = MAX, σ(s,t) = 0
30) 执行算法1中第22)~28)行
```

当发生边删除更新时,首先挑选出变化点对,并删去旧介数中心度(算法2第1)行)。随后并行遍历*Nodepair*中的所有点对,通过本文提出的分类筛选,对于原有最短路未全部断开的点对直接更新(第4)~5)行);对于其他最短路全部断开需重新计算的点对,本文使用基于BFS(Breadth First Search)思想的算法同时完成点对最短路距离和最短路数量的计算(第6)~29)行)。最后,加上新介数中心度并对社区间最短路等数据进行更新(第30)行)。

定理7　算法2具有确定性。

证明　从算法2中易知,第2)~29)行的并行化循环不存在迭代顺序依赖和访问冲突,因此具有循环独立性。又因为算法2中的其他部分与算法1相同,且由定理4可知算法1具有确定性,因此易得算法2具有确定性。

定理8　算法2的时间复杂度为$O\left(\rho_1|V|^2|E|/T+\rho_2|E|^3/T\right)$。其中:$T$为程序运行时可用的并行线程数量;$\rho_1$、$\rho_2$分别为发生边删除时,点对间原有最短路径全部断开和仅部分断开的点对在所有点对中所占比例。

证明　算法2中除部分点对需使用复杂度较高的BFS算法重新计算最短路外,与算法1步骤大致相似,因此定理7可类比定理5得证。

定理9　算法2的空间复杂度为$O(|V|^2)$。

证明　算法2与算法1需维护的数据一致,因此参考定理6可得证。

需要注意的是,本文提出的PAB可以扩展到有权图和有向图。处理有向图时,算法框架基本不变,需根据自己的代码设计自行调整;处理有权图时,应将算法2中的BFS算法改为Dijkstra算法。

5　实验与结果分析

5.1　实验准备

实验所用操作系统是Ubuntu 16.04,CPU是Intel Xeon E5-2630(20-core),内存为315 GB。本文所提算法和实验代码都基于C++语言和VS 2017实现。

实验使用了3个合成数据集(Opera_1000、Opera_10000和Opera_100000)与7个真实数据集(ego-Facebook、Oregon-010331、Oregon-010428、p2p-Gnutella25、Wiki-Vote、Email-Enron和Brightkite),见表1。真实数据集主要来自SNAP库(http://snap.stanford.edu/data)。其中,ego-Facebook是来自平台Facebook上的社交网络信息;Oregon-010331和Oregon-010428分别是俄勒冈在2001年3月、4月的网络信息;p2p-Gnutella25是2002年8月的一份Gnutella对等文件共享网络的快照数据;Wiki-Vote是维基百科管理员投票数据;Email-Enron是安然电子邮件信息;Brightkite是来自社交网络公司Brightkite的用户友谊网络信息。

表1　数据集的统计信息

Tab. 1　Statistics of datasets

数据集	节点数	边数	类型
opera_1000	1 000	15 484	无向
opera_10000	10 000	307 562	无向
opera_100000	100 000	1 335 005	无向
ego-Facebook	4 039	88 234	无向
Oregon-010331	10 670	22 002	无向
Oregon-010428	10 886	22 493	无向
p2p-Gnutella25	22 687	54 705	有向
Wiki-Vote	7 115	103 689	有向
Email-Enron	36 692	183 831	无向
Brightkite	58 228	214 078	无向

本文将数据统一处理为无向图,并通过程序随机挑选边

的方式进行边添加、删除操作，以此模拟现实中图的动态更新。

本文主要考虑的评价指标是时间（T）和加速比（Ra）。所用时间越少，加速比越大，则说明本文算法的效率越高。

5.2 性能对比实验

为了评价本文所提PAB的有效性，选取目前并行算法中最新的batch-iCENTRAL（记为b-iC）[7]，以及经典的Brandes算法[11]并行实现Brandes-parallel（记为Br-p）作为基准算法，在合成数据集与真实数据集上开展对比实验，实验结果见表2。其中在社区初始化划分阶段，本文使用的社区发现算法是HANP（Hop Attenuation & Node Preference）。

表2中：Br-p算法不区分删除边和添加边；对大规模数据集opera_100000记录5次操作的平均用时，对其他数据集皆记录50次操作的平均用时；时间T/s下的Br-p表示Br-p算法所用时间，加速比Ra下的Br-p表示Br-p算法与PAB所用时间的比值。

从表2中可以看出，PAB相较于其他基准算法通常有着更高的更新效率。与Br-p相比，最高是其322.7倍，平均快55倍；与b-iC相比，最高是其17.01倍，平均是其4倍。具体来说，在3个合成数据集、ego-Facebook和Wiki-Vote这几个数据集上，PAB相较于其他基准算法的加速效果非常显著。这主要是因为这些数据集的社区特性较为突出，当添加边或删除边后，PAB能够更高效地利用社区性质过滤掉冗余点对，提高介数中心度的更新效率。PAB效率与社区精度的关系将在5.4节进一步分析。

需要指出的是，在部分数据集（p2p-Gnutella25等）上处理删除边更新时，PAB的效率低于最新的b-iC算法。主要原因有两方面：一方面，这些数据集的社区特征相对较弱，不能充分发挥PAB优势；另一方面，处理删除边时，PAB会因BFS等操作产生额外时间开销。但整体而言，在多数情况下使用本文PAB的效率更高，特别是处理添加边更新时。

5.3 可扩展性实验

为了探究本文算法在数据集规模扩增时的性能表现，选取同样使用社区过滤思想的CBU算法[10]作为基准算法，在合成数据集opera_1000、opera_10000和opera_100000上开展对比实验。图5展示了随着合成数据集节点规模的扩增，本文PAB相较于CBU算法加速比的变化情况。

根据实验结果可以看出，随着数据集规模的不断扩大，添加边和删除边情况的加速比都在提升，这说明PAB可以有效地扩展到大规模网络中（其中最大的数据集opera_100000规模达到了百万）。另外可以发现，在小规模数据集opera_1000上加速效果与大规模数据集差距较大，这是因为在小规模数据集上计算本身耗时很少，而并行计算存在共享内存和缓存等额外时间开销，导致整体加速比较低。

表2 不同数据集上的更新效率

Tab. 2 Efficiency of update on different datasets

数据集	时间T/s					加速比Ra			
	Br-p	b-iC(+)	b-iC(−)	PAB(+)	PAB(−)	Br-p(+)	Br-p(−)	b-iC(+)	b-iC(−)
opera_1000	1.16	0.21	0.23	0.04	0.06	29.00	19.33	5.25	3.83
opera_10000	264.61	13.95	17.16	0.82	1.58	322.70	167.47	17.01	10.86
opera_100000	15 147.30	1 399.09	1 970.63	160.63	631.26	94.30	24.00	8.71	3.12
ego-Facebook	24.95	2.79	0.87	3.68	0.11	6.78	226.82	0.76	7.91
Oregon-010331	34.40	3.89	4.21	3.50	6.14	9.83	5.60	1.11	0.69
Oregon-010428	36.72	4.14	4.26	3.20	5.86	11.48	6.27	1.29	0.73
p2p-Gnutella25	179.39	16.52	17.89	12.23	35.05	14.67	5.12	1.35	0.51
Wiki-Vote	60.81	4.17	4.50	0.79	1.71	76.97	35.56	5.28	2.63
Email-Enron	773.63	78.82	77.30	55.62	106.68	13.90	7.25	1.42	0.72
Brightkite	2 180.86	193.34	199.84	171.03	434.78	12.75	5.02	1.13	0.46

注：(+)代表边添加操作，(−)代表边删除操作。

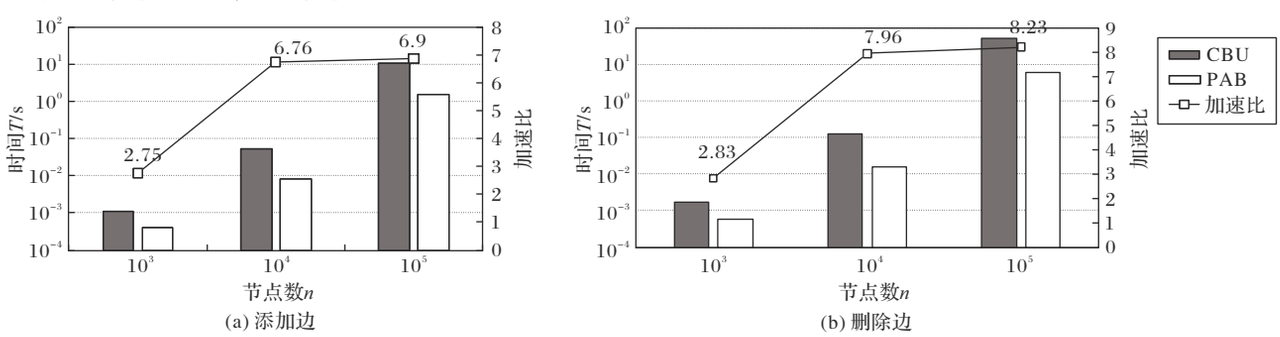

图5 可扩展性测试结果

Fig. 5 Results of scalability test

5.4 社区质量实验

由于本文提出的PAB应用了社区的性质，因此有必要探究社区结构的质量对算法效率的影响。

本文选取模块度作为评估社区质量的指标，该指标由Newman等[18]于2004年提出，其取值范围为[−0.5, 1)。模块度的取值取决于各社区内实际边数与期望边数之差，因此模块度取值越大代表该网络图中社区划分的质量越高。

本文针对合成数据集opera_10000进行社区质量实验：首先，使用多个社区发现算法（DSGE（Dense SubGraph Extraction）[19]、LPA（Label Propagation Algorithm）[20]、

HANP[21])进行初始社区划分,计算模块度;随后,在不同的社区划分下,分别实施PAB,并统计耗时。

从表3中可以看到,模块度值越高,PAB的运行用时越少。其中在添加边更新时,PAB使用HANP划分的社区进行运算的速度是使用DSGE的2.4倍。总之,社区结构的质量越高,PAB的效率就越高。

表3　不同社区结构质量下的更新效率

Tab. 3　Efficiency of update under different quality of community structure

社区发现算法	模块度	时间 T/s	
		PAB(+)	PAB(−)
DSGE	0.17	1.95	2.27
LPA	0.36	1.39	2.02
HANP	0.40	0.82	1.58

6　结语

本文提出了一种动态网络中介数中心度的并行算法PAB,通过社区过滤、等距剪枝、分类筛选等操作减少了冗余点对开销;之后又使用OpenMP完成了整体算法的并行化,并对算法的并行确定性进行了分析。实验结果表明,PAB有效提高了动态网络中节点介数中心度的更新效率。未来会考虑将本文工作应用于介数中心度近似计算。

参考文献 (References)

[1] FREEMAN L C. A set of measures of centrality based on betweenness[J]. Sociometry, 1977, 40(1): 35-41.

[2] BADER D A, KINTALI S, MADDURI K, et al. Approximating betweenness centrality[C]// Proceedings of the 2007 International Workshop on Algorithms and Models for the Web-Graph, LNC 4863. Berlin: Springer, 2007: 124-137.

[3] ÖZGÜR A, VU T, ERKAN G, et al. Identifying gene-disease associations using centrality on a literature mined gene-interaction network[J]. Bioinformatics, 2008, 24(13): i277-i285.

[4] DALY E M, HAAHR M. Social network analysis for routing in disconnected delay-tolerant MANETs[C]// Proceedings of the 8th ACM International Symposium on Mobile Ad Hoc Networking and Computing. New York: ACM, 2007: 32-40.

[5] McLAUGHLIN A, BADER D A. Revisiting edge and node parallelism for dynamic GPU graph analytics[C]// Proceedings of the 2014 IEEE International Parallel and Distributed Processing Symposium Workshops. Piscataway: IEEE, 2014: 1396-1406.

[6] JAMOUR F, SKIADOPOULOS S, KALNIS P. Parallel algorithm for incremental betweenness centrality on large graphs[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(3): 659-672.

[7] SHUKLA K, REGUNTA S C, TONDOMKER S H, et al. Efficient parallel algorithms for betweenness- and closeness-centrality in dynamic graphs[C]// Proceedings of the 34th ACM International Conference on Supercomputing. New York: ACM, 2020: No. 10.

[8] CLEMENTE G P, CORNARO A. Community detection in attributed networks for global transfer market[J]. Annals of Operations Research, 2022: 1-27.

[9] DILMAGHANI S, BRUST M R, RIBEIRO C H C, et al. From communities to protein complexes: a local community detection algorithm on PPI networks[J]. PLoS ONE, 2022, 17(1): No. e0260484.

[10] 钱珺,王朝坤,郭高扬. 基于社区的动态网络节点介数中心度更新算法[J]. 软件学报, 2018, 29(3): 853-868. (QIAN J, WANG C K, GUO G Y. Community based node betweenness centrality updating algorithms in dynamic networks[J]. Journal of Software, 2018, 29(3): 853-868.)

[11] BRANDES U. A faster algorithm for betweenness centrality[J]. The Journal of Mathematical Sociology, 2001, 25(2): 163-177.

[12] LEE M J, LEE J, PARK J Y, et al. QUBE: a quick algorithm for updating betweenness centrality[C]// Proceedings of the 21st International Conference on World Wide Web. New York: ACM, 2012: 351-360.

[13] GREEN O, McCOLL R, BADER D A. A fast algorithm for streaming betweenness centrality[C]// Proceedings of the 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust and 2012 ASE/IEEE International Conference on Social Computing. Piscataway: IEEE, 2012: 11-20.

[14] MAURYA S K, LIU X, MURATA T. Fast approximations of betweenness centrality with graph neural networks[C]// Proceedings of the 28th ACM International Conference on Information and Knowledge Management. New York: ACM, 2019: 2149-2152.

[15] CHEHREGHANI M H, BIFET A, ABDESSALEM T. Exact and approximate algorithms for computing betweenness centrality in directed graphs[J]. Fundamenta Informaticae, 2021, 182(3): 219-242.

[16] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to Algorithms[M]. 3rd ed. Cambridge: MIT Press, 2009: 787-788.

[17] 王堃. 基于多核的并行程序设计及优化[D]. 南京:南京大学, 2012: 10-16. (WANG K. Parallel programming and optimization based on multi-core processors[D]. Nanjing: Nanjing University, 2012:10-16.)

[18] NEWMAN M E J, GIRVAN M. Finding and evaluating community structure in networks[J]. Physical Review E: Statistical, Nonlinear, and Soft Matter Physics, 2004, 69(2): No. 026113.

[19] CHEN J, SAAD Y. Dense subgraph extraction with application to community detection[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(7): 1216-1230.

[20] RAGHAVAN U N, ALBERT R, KUMARA S. Near linear time algorithm to detect community structures in large-scale networks[J]. Physical Review E: Statistical, Nonlinear, and Soft Matter Physics, 2007, 76(3): No. 036106.

[21] LEUNG I X Y, HUI P, LIÒ P, et al. Towards real-time community detection in large networks[J]. Physical Review E: Statistical, Nonlinear, and Soft Matter Physics, 2009, 79(6): No. 066107.

This work is partially supported by National Natural Science Foundation of China (61872207).

LIU Zhenyu, born in 1999, M. S. candidate. His research interests include social network, community detection.

WANG Chaokun, born in 1976, Ph. D., associate professor. His research interests include social network analysis, graph data management, big data system.

GUO Gaoyang, born in 1994, Ph. D. candidate. His research interests include social network, graph neural network, knowledge graph construction.