软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(3):853−868 [doi: 10.13328/j.cnki.jos.005457]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于社区的动态网络节点介数中心度更新算法*

钱 珺, 王朝坤, 郭高扬

(清华大学 软件学院,北京 100084)

通讯作者: 王朝坤, E-mail:chaokun@tsinghua.edu.cn

摘 要: 随着互联网技术的迅猛发展,社会网络呈现出爆炸增长的趋势,传统的静态网络分析方法越来越难以达到令人满意的效果.于是,对网络进行动态分析就成为社会网数据管理领域的一个研究热点.节点介数中心度衡量的是一个节点对图中其他点对最短路径的控制能力,有利于挖掘社会网络中的重要节点.在图结构频繁变化的场合,若每次变化后都重新计算整个图中所有节点的介数中心度,效率将会降低.针对动态网络中节点介数中心度计算困难的问题,提出一种基于社区的节点介数中心度更新算法.通过维护社区与社区、社区与节点的最短距离集合,快速过滤掉那些在网络动态更新中不受影响的点对,从而提高节点介数中心度的更新效率.真实数据集和合成数据集上的实验结果表明了所提算法的有效性.

关键词: 节点介数中心度;社区;动态网络;CBU;最短距离

中图法分类号: TP311

中文引用格式: 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.软件学报,2018,29(3):853−868. http://www.jos.org.cn/1000-9825/5457.htm

英文引用格式: Qian J, Wang CK, Guo GY. Community based node betweenness centrality updating algorithms in dynamic networks. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):853−868 (in Chinese). http://www.jos.org.cn/1000-9825/5457.htm

Community Based Node Betweenness Centrality Updating Algorithms in Dynamic Networks

QIAN Jun, WANG Chao-Kun, GUO Gao-Yang

(School of Software, Tsinghua University, Beijing 100084, China)

Abstract: With the rapid development of Internet technology, social networks show a trend of explosive growth. As the traditional analysis on static networks becomes more and more difficult to achieve satisfactory results, dynamic network analysis has turned into a research hotspot in the field of social network data management. Node betweenness centrality measures the ability of a node to control the shortest paths between other nodes in the graph, which is useful for mining important nodes in social networks. However, the efficiency will be low if the betweenness centrality of all nodes needs to be calculated each time while the graph structure changes frequently. To address the difficult problem of computing node betweenness centrality in dynamic networks, a community based betweenness centrality updating algorithm is proposed in this paper. By maintaining the shortest distance sets between communities and communities, as well as between communities and nodes, the node pairs which are not affected during the dynamically updating process can be quickly filtered out, thus greatly improving the updating efficiency of node betweenness centrality. Experimental results conducted on real-world datasets and synthetic datasets show the effectiveness of the proposed algorithms.

Key words: node betweenness centrality; community; dynamic network; CBU; shortest distance

* 基金项目: 国家自然科学基金(61373023); 工业和信息化部智能制造综合标准化与新模式应用项目

Foundation item: National Natural Science Foundation of China (61373023); Intelligent Manufacturing Comprehensive Standardization and New Pattern Application Project of Ministry of Industry and Information Technology

本文由基于图结构的大数据分析与管理技术专刊特约编辑林学民教授、杜小勇教授、李翠平教授推荐.

收稿时间: 2017-08-07; 修改时间: 2017-09-05; 采用时间: 2017-11-07; jos 在线出版时间: 2017-12-05

CNKI 网络优先出版: 2017-12-06 15:37:14, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1536.020.html

挖掘网络中的重要节点是图分析领域中的基础问题,例如在社交网络中,找到那些影响力大的用户节点可以更有效地实现病毒营销等任务.衡量节点自身重要性的标准有很多,比如点度中心度(degree centrality)、接近中心度(closeness centrality)、特征向量中心度(eigenvector centrality)和介数中心度(betweenness centrality)[1]等.其中,节点的介数中心度(betweenness centrality)量化了一个节点作为其他节点之间最短路径上桥梁的能力,刻画出了社会网络中一个用户对于其他用户之间交流的影响能力,是一种非常重要的度量指标.于是,节点介数中心度(本文以下简称为介数中心度)经常被应用于供应链管理、恐怖分子发现和艾滋病网络等诸多领域[2].

近些年来,作为网络图分析领域的一个研究热点,涌现出了一批介数中心度计算的优秀成果[2−8].然而,因为介数中心度的精确计算依赖于图中所有点对间的最短路径,所以其计算效率依然不高.对于介数中心度精确计算任务,目前公认的最快方法在无权图上的时间复杂度为 $O(mn)$,在有权图上的时间复杂度为 $O(mn+n^2\log n)$[3].显然,上述效率无法满足现实生活中的大量需求.例如,在以微博为代表的主流社交网络中,用户关系是不断变化的.网络中一方面持续产生大量新“网红”;另一方面,绝大多数“网红”过一段时间后就人气滑落,甚至消失.这些“网红”节点的重要性往往存在一个急速上升而后又急速下降的过程.如果在网络每次变化时,都重新计算所有节点的介数中心度,那么时间开销显然是巨大的.因此,本文对动态网络中节点介数中心度的更新进行研究,尝试寻找一种快速更新的方法,提高介数中心度精确计算的效率.

除了节点和边,网络图中通常还存在社区的概念.社区[9]是复杂网络的普遍特征,可以认为一个网络是由许多社区组成的.社区本身还没有公认的形式化定义.一般认为,社区就是由一些节点组成的聚合体,其中,同一社区内的节点关系相对密切,而不同社区间的节点关系相对而言比较松散.

社区发现(community detection)对于研究网络的特性具有重要意义.找到那些属于同一社区的相似节点,可以更加细化地进行用户的特征挖掘、喜好推荐等工作.因此,近些年许多学者投入到复杂网络中社区的发现与研究中,并提出了不少社区发现的算法,社区发现的研究成果也被成功应用于好友推荐、个性化商品推荐、蛋白质功能预测、舆情分析与处理等众多领域[10].受社区概念的启发,我们考虑借助社区自身的特性,即社区间、社区内节点的联系,尝试在动态图上快速、精准地进行节点介数中心度的更新计算.

针对前述问题,本文提出一种基于社区的节点介数中心度的快速更新算法,能够有效减少节点介数中心度的计算量,提高节点介数中心度的更新效率.

本文的主要贡献包括:

(1) 提出了社区与社区间最短距离集合、社区与节点的最短距离集合的概念,并基于这些概念快速找到图结构变化过程中最短路径改变的点对;

(2) 提出了一种基于社区间最短距离集合的节点介数中心度的快速更新算法 CBU(community based node betweenness centrality updating).利用上述概念,只需要计算那些最短路径改变的点对,即可提高节点介数中心度的更新效率;

(3) 在真实数据集和合成数据集上进行了大量实验.实验结果表明,本文所提算法具有良好的加速效果.

本文第 1 节介绍节点介数中心度计算和社区发现领域的相关研究工作.第 2 节给出 CBU 算法所用到的基本概念.第 3 节给出最短路径数量的计算方法和 CBU 算法的社区过滤策略.第 4 节提出基于社区的介数中心度更新算法 CBU,并分析其时间和空间复杂度.第 5 节给出实验测试结果及分析.第 6 节总结全文.

1 相关工作

1.1 介数中心度

介数中心度是网络图中节点中心度度量的一项重要指标,它受图中所有点对的最短路径影响.在图中常用的计算所有点对最短路径的 Floyd-Warshall 算法的时间复杂度为 $O(n^3)$,这导致基于 Floyd-Warshall 算法进行节点介数中心度计算的方法时间复杂度均不低于 $O(n^3)$.显然,这样的计算速度太慢,无法满足动态网络的需求.因此,如何减少介数中心度所依赖的最短路径计算量,从而提高介数中心度的计算效率,也就成为了研究的重点.具体地,有如下两种基本的加速思路:

第 1 种,减少冗余的计算量.Brandes 等人[3]提出了一种基于改进的广度优先搜索和点对依赖的算法,减少了最短路径不必要的计算,使无权图上的时间复杂度降低为 $O(mn)$,有权图上的时间复杂度也降为 $O(mn+n^2\log n)$.这也是目前最常用的介数中心度计算方法.然而,Brandes 算法本身仍旧依赖于所有点对间最短路径的计算,效率依然有限.如何进一步减小介数中心度对于点对最短路径的依赖性,成为加快介数中心度更新的首要策略.Sariyüce 等人[4]提出了一种将图拆分压缩的策略,通过将复杂的网络图拆分成多个简单的子图,并将许多作用等价的节点合并,大大简化原有的网络,从而加快介数中心度的计算.Lee 等人[5,6]提出了一种利用无向图中的环路的方法,过滤介数中心度不受边增减操作影响的节点,从而提高了更新效率.

第 2 种,计算介数中心度的近似值,而不追求其精确值.Brandes 等人[7]提出了一种近似算法,通过选取一些支点,计算有限数量的单源最短路径,进而对所有节点的介数中心度进行无偏估计.Bader 等人[2]提出了一种基于自适应的采样技术,显著降低了具有较高中心度的节点单源最短路径计算量.Geisberger 等人[8]改进并推广了 Brandes 的算法,给出了一个计算框架,提高了原有的近似比.

然而,上述两种加速思路都没有充分考虑到网络图中固有的社区特性.

1.2 社区发现

社区的概念最早是由 Girvan 等人[9]于 2002 年提出,他们认为:复杂网络除了具有小世界和无标度的特征之外,还具有社区这种特殊结构.此后,社区与社区发现逐渐成为热门的研究话题,来自不同领域的学者们从各自的角度提出了大量的有效方法.

文献[9]提出了社区发现的基本算法 G-N,通过移除具有最大边介数的边来划分社区.该算法将原来的网络直接进行分割,自上而下形成社区.在此基础上衍生出了许多算法,例如, Radicchi[11]和 Newman[12]分别用边聚类系数和 modularity 矩阵的特征值代替最大边介数来进行网络分割.整体上看,这类算法精度很高,但是速度较慢.例如,G-N 的时间复杂度为 $O(n^3)$.

与上述依据分割思想的方法相对应,自底向上凝聚的方法通过逐步扩展的方式形成最终的社区.Newman[13]提出了一种快速算法 NFGA,其精度比 G-N 略低,但时间复杂度减小到了 $O((m+n)n)$.在此基础上,Clauset 等人[14]又给出了改进算法 CNM,时间复杂度仅为 $O(n(\log n)^2)$.

另一种主流的社区发现方法通过标签的传播形成社区.Raghavan 等人[15]于 2007 年提出了 LPA 算法,首先给每个节点指定唯一的标签,而后通过异步更新的方式将每个节点的标签赋值为其邻接节点中出现次数最多的标签,最终将标签一致的节点划为一个社区.在此基础上,Leung 等人[16]于 2009 年提出了 HANP 算法,进一步引入标签传播能力衰减的概念和节点获取新标签偏好的概念,使得社区发现的结果更加稳定与可靠.上述两个算法的时间复杂度均为 $O(T(m+n))$(其中,T 为迭代次数),效率相对较高.

除此之外,还有矩阵分块(DSGE)[17]、脉络图聚类(gCluSkeleton)[18]、图嵌入(Deepwalk)[19]和深度稀疏自动编码器(CoDDA)[20]等不同方法.

结合介数中心度和社区发现的研究目前尚不多见.与本文最相近的工作是文献[21],该文提出了一种基于社区的最短路径近似算法,利用社区代替社区内部的具体节点,维护社区间的最短路径,近似计算节点间的最短路径.但是该文仅将社区作为一类超节点,并未考虑网络动态变化对社区结构带来的变更,不能有效支持动态网络.同时,该方法得到的只是近似的最短路径,而非精确的最短路径,于是也就无法支持节点介数中心度的有效计算.

2 基本概念

本节对 CBU 算法涉及到的概念术语进行定义.

给定复杂网络图 $G=(V,E)$,$V=\{v_1,v_2,\ldots,v_n\}$是节点集合,$E=\{e_1,e_2,\ldots,e_m\}$是边集合.如果没有特别声明,$n=|V|$表示节点的个数,$m=|E|$表示边的个数,$\omega(v,w)$表示节点 v 到节点 w 的边权值(无权重默认为 1).$c=\{v_1,v_2,\ldots,v_{|c|}\}$表示由$|c|$个节点组成的社区.两个节点之间的距离 $d_{vv}(s,t)$指从节点 s 出发到达节点 t 的最短路径的长度,简记为 $d(s,t)$.$\sigma(s,t)$表示节点 s 到节点 t 的最短路径的数量.$\sigma(s,t|v)$则表示节点 s 到节点 t 的经过节点 v 的最短路径

的数量.于是,节点 v 的介数中心度为 $c_B(v)=\sum_{s\neq t\neq v,s,t\in V}\frac{\sigma(s,t\mid v)}{\sigma(s,t)}$.

定义 1(社区到社区的最短距离集合). 给定网络图 $G=(V,E)$,$c_1=\{v_1,v_2,\ldots,v_k\}$和 $c_2=\{u_1,u_2,\ldots,u_l\}$是 G 中的两个社区.c_1 到 c_2 的最短距离集合 $d_{cc}(c_1,c_2)$为 c_1 中所有节点到 c_2 中所有节点的最短距离的集合,即:

$$d_{cc}(c_1,c_2)=\{d(v,u)|v\in c_1,u\in c_2\} \tag{1}$$

鉴于整个最短距离集合所占据的存储空间较大,且在计算节点介数中心度的实际应用中,该集合的边界值比集合内大部分值的作用更大,于是,我们仅记录每个最短距离集合中的最小值 $d_{cc}(c_1,c_2)_{\min}$ 与最大值 $d_{cc}(c_1,c_2)_{\max}$.

定义 2(社区与节点的最短距离集合). 给定网络图 $G=(V,E)$,$c=\{v_1,v_2,\ldots,v_k\}$是 G 中一个社区,u 是图 G 中一个节点.社区 c 到节点 u 的最短距离集合 $d_{cv}(c,u)$为社区 c 中所有节点到节点 u 的最短距离的集合;同时,节点 u 到社区 c 的最短距离集合 $d_{vc}(u,c)$为节点 u 到社区 c 中所有节点的最短距离的集合,即:

$$d_{cv}(c,u)=\{d(v,u)|v\in c\} \tag{2}$$

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

显然,在无向图中,$d_{cv}(c,u)$和 $d_{vc}(u,c)$是等价的.而在有向图中,两者并不相等.同样为了减少存储开销,对于无向图,我们只记录 $d_{cv}(c,u)$的最小值 $d_{cv}(c,u)_{\min}$ 和最大值 $d_{cv}(c,u)_{\max}$;对于有向图,我们则记录 $d_{cv}(c,u)$的最小值 $d_{cv}(c,u)_{\min}$和最大值 $d_{cv}(c,u)_{\max}$ 以及 $d_{vc}(u,c)$的最小值 $d_{vc}(u,c)_{\min}$ 和最大值 $d_{vc}(u,c)_{\max}$.

例 1:如图 1 所示,无向图 G 中包含 16 个节点.假定依照虚线将 G 划分为 4 个社区.对于左上角的社区 c_1 和右下角的社区 c_4,因为 c_1 和 c_4 之间距离最近的点对为(v_5,v_{15}),其最短距离 $d(v_5,v_{15})=1$,距离最远的点对是(v_1,v_{16}),其最短距离 $d(v_1,v_{16})=3$,所以可以得到 $d_{cc}(c_1,c_4)_{\min}=1$,$d_{cc}(c_1,c_4)_{\max}=3$.同理,社区 c_1 与节点 v_{14} 的最短距离集合 $d_{cv}(c_1,v_{14})$的最小值 $d_{cv}(c_1,v_{14})_{\min}=3$,最大值 $d_{cv}(c_1,v_{14})_{\max}=4$.

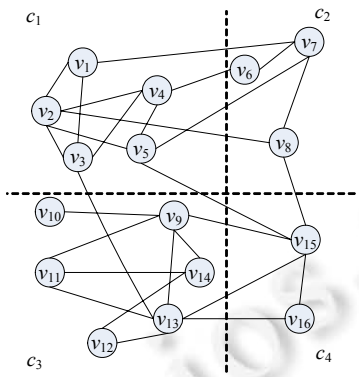

Fig.1 An example for the shortest path sets between communities

图 1 社区最短距离集合示意

3 点对过滤

本节给出 CBU 算法中点对间最短路径数量的计算方法以及基于社区的点对过滤策略,它们是下一节中动态网络节点介数中心度更新算法的基础.

3.1 最短路径

从节点介数中心度的定义可知,介数中心度依赖于点对间的最短路径数量.因此,我们在本节给出最短路径数量的有效计算方法.

如算法 1 和算法 2 所示:本文的最短路径数量计算方法采取深度优先搜索的遍历方式,对给定点对依次访问源节点所有的邻居节点,由最短距离判断邻居节点是否出现在给定点对的最短路径中.若出现,则将给定点对

的最短路径数量转化为其邻居节点到目标节点的最短路径数目之和.通过上述方法,即可得到我们所需点对的最短路径数量.注意到更新两点间最短路径数$\sigma(s,t)$时,会计算一些额外的点对间最短路径数(算法 2 第 7 行,计算出了$\sigma(v,t)$),因此,我们会在具体实现中维护一个标志位矩阵,记录哪些点对间的最短路径数已经算出,以避免重复计算.

算法 1. 所有点对间最短路径数计算 CountPath.

输入:$G=(V,E)$,d;

输出:σ.

1. **for** $(s,t)\in V\times V$ **do**
2. $\sigma(s,t)\leftarrow UpdatePath(G,d,s,t)$
3. **return** σ

算法 2. 两点间最短路径数更新 UpdatePath.

输入:$G=(V,E)$,d,s,t;

输出:$\sigma(s,t)$.

1. **if** $s==t$
2. $\sigma(s,t)\leftarrow 1$
3. **else**
4. $\sigma(s,t)\leftarrow 0$
5. **for** $(s,v)\in E$ **do**
6. **if** $d(s,t)==d(s,v)+d(v,t)$ **then**
7. $\sigma(v,t)\leftarrow UpdatePath(G,d,v,t)$
8. $\sigma(s,t)\leftarrow\sigma(s,t)+\sigma(v,t)$
9. **return** $\sigma(s,t)$

例 2:以图 1 中节点对(v_{11},v_{15})为例,用算法 2 计算从 v_{11} 到 v_{15} 的最短路径数$\sigma(v_{11},v_{15})$.因为 v_{11} 和 v_{15} 不是同一个节点,所以$\sigma(v_{11},v_{15})$初始化为 0.注意到节点 v_{11} 有邻居节点 v_9、v_{13} 和 v_{14}.对于节点 v_9,因为 $d(v_{11},v_{15})=2=d(v_{11},v_9)+d(v_9,v_{15})=1+1$,所以存在经过节点 v_9 的从节点 v_{11} 到节点 v_{15} 的最短路径,用 $UpdatePath(G,d,v_9,v_{15})$计算出$\sigma(v_9,v_{15})=1$,因此$\sigma(v_{11},v_{15})=\sigma(v_{11},v_{15})+\sigma(v_9,v_{15})=0+1=1$.对于节点 v_{13},因为 $d(v_{11},v_{15})=2=d(v_{11},v_{13})+d(v_{13},v_{15})=1+1$,所以也存在经过节点 v_{13} 的从节点 v_{11} 到节点 v_{15} 的最短路径,用 UpdatePat(G,d,v_{13},v_{15})计算出$\sigma(v_{13},v_{15})=1$,因此$\sigma(v_{11},v_{15})=\sigma(v_{11},v_{15})+\sigma(v_{13},v_{15})=1+1=2$.而对于节点 v_{14},$d(v_{11},v_{15})=2<d(v_{11},v_{14})+d(v_{14},v_{15})=1+2$,所以不存在经过节点 v_{14} 的从节点 v_{11} 到节点 v_{15} 的最短路径,$\sigma(v_{11},v_{15})$不变.综上所述,从节点 v_{11} 到节点 v_{15} 的最短路径数$\sigma(v_{11},v_{15})=2$.

以上计算了给定网络图中两点间的最短路径数量$\sigma(s,t)$,接下来考虑如何计算经过任意节点的最短路径数量$\sigma(s,t|v)$.最朴素的方法是记录每一条最短路径经过的所有节点,进而逐个节点进行统计.但是这样需要维护所有最短路径的信息,不仅占用的空间巨大,而且时间开销也较长.因此,我们考虑通过定理 1 将其转化为更容易计算的形式:

定理 1[22]. 给定图 $G=(V,E)$,若图上节点 v 至少在节点 s 到节点 t 的一条最短路径上出现,则 s 到 t 的经过 v 的最短路径个数$\sigma(s,t|v)$,s 到 v 的最短路径个数$\sigma(s,v)$,v 到 t 的最短路径个数$\sigma(v,t)$,三者满足性质:

$$\sigma(s,t|v)=\sigma(s,v)\times\sigma(v,t).$$

基于定理 1,可将经过某点的最短路径数量转化成对应的最短路径数量的乘积,提高计算效率.

3.2 社区过滤

引理 1. 给定有向网络图 $G=(V,E)$,s,t,u,v 为图 G 中的 4 个节点.令 $u\rightarrow v$ 是在节点 u 和 v 之间添加的一条边,权重为ω.s 到 t 的最短路径发生改变,当且仅当 $d(s,t)\geqslant d(s,u)+d(v,t)+\omega(u,v)$.

证明:

($\Leftarrow$必要性):假设 $d(s,t)\geqslant d(s,u)+d(v,t)+\omega$.设 s 到 u 的一条最短路径为$(s\ldots u)$,v 到 t 的一条最短路径为$(v\ldots t)$,

则 s 到 t 的新路径($s\ldots u,v\ldots t$)的长度为 $d(s,u)+d(v,t)+\omega$.根据假设有 $d(s,u)+d(v,t)+\omega\leqslant s$ 到 t 的原最短路径的长度 $d(s,t)$,因此($s\ldots u,v\ldots t$)成为了新的最短路径,即 s 到 t 的最短路径发生改变.

($\Rightarrow$充分性):反证法.假设 $d(s,t)<d(s,u)+d(v,t)+\omega$.若最短路径 s 到 t 发生改变,则新的最短路径 P 必经过添加的边(u,v).于是,P 的长度 $d(s,u)+d(v,t)+\omega\leqslant s$ 到 t 的原最短路径的长度 $d(s,t)$,这与原假设矛盾.于是,$d(s,t)\geqslant d(s,u)+d(v,t)+\omega$ (u,v). □

在引理 1 的基础上,可以得到定理 2.

定理 2. 给定有向网络图 $G=(V,E)$,$c_1=\{s_1,s_2,\ldots,s_k\}$和 $c_2=\{t_1,t_2,\ldots,t_l\}$是 G 中的两个社区,u 和 v 为 G 中两个节点.令 $u\to v$ 是在节点 u 和 v 之间添加的一条权重为ω的边.当 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均不变;当 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均改变.

证明:

- 当 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega$ 时,对于 $\forall s\in c_1,\forall t\in c_2,d(s,t)\leqslant d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega\leqslant d(s,u)+d(v,t)+\omega$.由引理 1,$s$ 到 t 的最短路径不变;
- 当 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega$ 时,对于 $\forall s\in c_1,\forall t\in c_2,d(s,t)\geqslant d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega\geqslant d(s,u)+d(v,t)+\omega$.由引理 1,$s$ 到 t 的最短路径均改变.

证毕. □

同理,我们可以得到定理 3~定理 5.

定理 3. 给定有向网络图 $G=(V,E)$,$c_1=\{s_1,s_2,\ldots,s_k\}$和 $c_2=\{t_1,t_2,\ldots,t_l\}$是 G 中的两个社区,u 和 v 为 G 中两个节点.令 $u\to v$ 是在节点 u 和 v 之间删除的一条权重为ω的边.当 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均不变;当 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均改变.

定理 4. 给定无向网络图 $G=(V,E)$,$c_1=\{s_1,s_2,\ldots,s_k\}$和 $c_2=\{t_1,t_2,\ldots,t_l\}$是 G 中的两个社区,u 和 v 为 G 中两个节点.令(u,v)是在节点 u 和 v 之间添加的一条权重为ω的边.当 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{cv}(v,c_2)_{\min}+\omega$且 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,v)_{\min}+d_{cv}(u,c_2)_{\min}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均不变;当 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{cv}(v,c_2)_{\max}+\omega$或 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,v)_{\max}+d_{cv}(u,c_2)_{\max}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均改变.

定理 5. 给定无向网络图 $G=(V,E)$,$c_1=\{s_1,s_2,\ldots,s_k\}$和 $c_2=\{t_1,t_2,\ldots,t_l\}$是 G 中的两个社区,u 和 v 为 G 中两个节点.令(u,v)是在节点 u 和 v 之间删除的一条权重为ω的边.当 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{cv}(v,c_2)_{\min}+\omega$且 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,v)_{\min}+d_{cv}(u,c_2)_{\min}+\omega$时,社区 c_1 内的点到社区 c_2 内点的最短路径均不变;当 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{cv}(v,c_2)_{\max}+\omega$或 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,v)_{\max}+d_{cv}(u,c_2)_{\max}+\omega$时,社区 c_1 内的点到社区 c_2 内的点的最短路径均改变.

基于上述定理,我们可以对更新操作后的不同情况进行具体分析.

在有向图上,增删一条边后有如下 3 种情况.

(1) 如果 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega(u,v)$,由定理 2 和定理 3 可知,这两个社区间的点对最短路径均不发生变化,可以全部忽略;

(2) 如果 $d_{cc}(c_1,c_2)_{\min}>d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega(u,v)$,由定理 2 和定理 3 可知,这两个社区间的点对最短路径均发生变化;

(3) 如果 $d_{cc}(c_1,c_2)_{\max}\geqslant d_{cv}(c_1,u)_{\min}+d_{vc}(v,c_2)_{\min}+\omega(u,v)$且 $d_{cc}(c_1,c_2)_{\min}\leqslant d_{cv}(c_1,u)_{\max}+d_{vc}(v,c_2)_{\max}+\omega(u,v)$,我们无法从这两个社区整体的最短距离数据上得到结果,需要对其点对进行逐一判断,从而确定每组点对最短路径的变化情况.

在无向图上,增删一条边后有如下两种情况.

(1) 如果 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,u)_{\min}+d_{cv}(v,c_2)_{\min}+\omega(u,v)$且 $d_{cc}(c_1,c_2)_{\max}<d_{cv}(c_1,v)_{\min}+d_{cv}(u,c_2)_{\min}+\omega(u,v)$,由定理 4 和定理 5 可知,这两个社区间的点对最短路径均不发生变化,全部忽略;

(2) 反之,我们也无法从这两个社区整体的最短距离数据上得到结果,需要对其点对进行逐一判断,从而

确定每组点对最短路径的变化情况.

综合以上分析,通过社区的筛选,我们可以从整体上过滤掉最短路径不受增加或删除操作所影响的点对,而后对剩余的点对再进行详细分析,减少最短路径的计算量,从而提高介数中心度的更新效率.

4 介数中心度更新算法

本节首先给出基于社区的节点介数中心度更新算法 CBU(由算法 3 和算法 4 组成)的具体步骤,然后对算法复杂度进行分析.为了便于读者理解,本节余下部分以无向图为例进行讨论,通过删减相关步骤易得有向图的对应版本.

4.1 边添加操作后的更新算法

我们先讨论边的添加操作后的介数中心度更新情况,见算法 3.该算法分为 4 个步骤:首先,通过社区与社区、社区与节点的最短距离集合进行过滤,利用定理 4 快速找到那些最短路径发生变化的点对,即待更新的点对,并计算出其新的最短距离和最短路径数(第 1 行~第 15 行);其次,利用定理 1 计算出待更新点对在更新前的最短路径对介数中心度的影响,并删去这些影响(第 16 行~第 19 行);然后,更新最短距离和最短路径数(第 20 行、第 21 行);最后,利用定理 1 计算待更新点对间新的最短路径对节点介数中心度的影响,并更新社区与社区、社区与节点的最短距离集合(第 22 行~第 27 行).

算法 3. 插入边操作下的介数中心度更新算法.

输入:$G=(V,E),d,d_{cv},d_{cc},\sigma,c_B,(u,v,\omega),C$;

输出:新的 c_B.

1. **for** $(c_i,c_j)\in C\times C$ **do**
2. **if** $d_{cc}(c_i,c_j)_{\max}\geqslant d_{cv}(c_i,u)_{\min}+d_{cv}(v,c_j)_{\min}+\omega$ **or** $d_{cc}(c_i,c_j)_{\max}\geqslant d_{cv}(c_i,v)_{\min}+d_{cv}(u,c_j)_{\min}+\omega$ **then**
3. **for** $(s,t)\in c_i\times c_j$ **do**
4. **if** $d(s,t)<d(s,u)+d(v,t)+\omega$ **and** $d(s,t)<d(s,v)+d(u,t)+\omega$ **then**
5. **continue**
6. **else**
7. 将点对(s,t)加入集合 *NodePair*,所属的社区对(c_i, c_j)加入集合 *ComPair*
8. **if** $d(s,t)=d(s,u)+d(v,t)+\omega$ **then**
9. $\sigma'(s,t)\leftarrow\sigma(s,t)+\sigma(s,u)\times\sigma(v,t),d'(s,t)\leftarrow d(s,t)$
10. **else if** $d(s,t)=d(s,v)+d(u,t)+\omega$ **then**
11. $\sigma'(s,t)\leftarrow\sigma(s,t)+\sigma(s,v)\times\sigma(u,t),d'(s,t)\leftarrow d(s,t)$
12. **else if** $d(s,u)+d(v,t)<d(s,v)+d(u,t)$ **then**
13. $\sigma'(s,t)\leftarrow\sigma(s,u)\times\sigma(v,t),d'(s,t)\leftarrow d(s,u)+d(v,t)+\omega$
14. **else then**
15. $\sigma'(s,t)\leftarrow\sigma(s,v)\times\sigma(u,t),d'(s,t)\leftarrow d(s,v)+d(u,t)+\omega$
16. **for** $(s,t)\in NodePair$ **do**
17. **for** $r\in V$ **do**
18. **if** $d(s,t)=d(s,r)+d(r,t)$ **then**
19. $c_B(r)\leftarrow c_B(r)-\sigma(s,r)\times\sigma(r,t)\div\sigma(s,t)$
20. **for** $(s,t)\in NodePair$ **do**
21. $\sigma(s,t)\leftarrow\sigma'(s,t),d(s,t)\leftarrow d'(s,t)$
22. **for** $(s,t)\in NodePair$ **do**
23. **for** $r\in V$ **do**
24. **if** $d(s,t)=d(s,r)+d(r,t)$ **then**

25. $c_B(r)\leftarrow c_B(r)+\sigma(s,r)\times\sigma(r,t)\div\sigma(s,t)$

26. 基于 *ComPair* 更新 d_{cv},d_{cc}

27. **return** c_B

例 3:在图 2 中,我们插入一条边 v_3v_{10},在更新节点介数中心度时,有些社区对可以直接过滤,而有的社区对不能过滤掉.对于社区 c_2,c_4 来说:$d_{cc}(c_2,c_4)_{\max}=4<d_{cv}(c_2,v_3)_{\min}+d_{cv}(v_{10},c_4)_{\min}+1=2+2+1=5$,$d_{cc}(c_2,c_4)_{\max}<d_{cv}(c_2,v_{10})_{\min}+d_{cv}(v_3,c_4)_{\min}+1=2+3+1=6$.所以社区 c_2,c_4 间节点构成的点对最短路径不变,直接过滤,不用进一步计算.而对于社区 c_1,c_3,则无法通过社区层面直接过滤,需对每一组点对依次过滤,比如:点对 v_2,v_{13},$d(v_2,v_{13})=2<d(v_2,v_3)+d(v_{10},v_{13})+1=1+2+1=4$,$d(v_2,v_{13})=2<d(v_2,v_{10})+d(v_3,v_{13})+1=4+1+1=6$,$v_2,v_{13}$ 的最短路径不变,不用进一步计算.而点对 v_1,v_9,$d(v_1,v_9)=3=d(v_1,v_3)+d(v_{10},v_9)+1=1+1+1=3$,其最短路径距离不变,最短路径数量增多,需要更新这些新增最短路径对节点介数中心度的影响.我们考虑 v_1,v_9 的最短路径对于节点 v_{13} 的介数中心度的影响.在插入边 v_3v_{10} 之前,v_1,v_9 的最短路径只有 $v_1v_3v_{13}v_9$,经过节点 v_{13},v_1,v_9 的最短路径对于节点 v_{13} 的介数中心度的影响为 1;在插入边 v_3v_{10} 之后,v_1,v_9 的最短路径增加了 $v_1v_3v_{10}v_9$,不经过节点 v_{13},v_1,v_9 的最短路径对于节点 v_{13} 的介数中心度的影响变成了 1/2.因此更新之后,点对 v_1,v_9 的最短路径的变化使得节点 v_{13} 的介数中心度减少了 1−1/2=1/2.其他社区对也可以照上述方法分析,这里不再赘述.

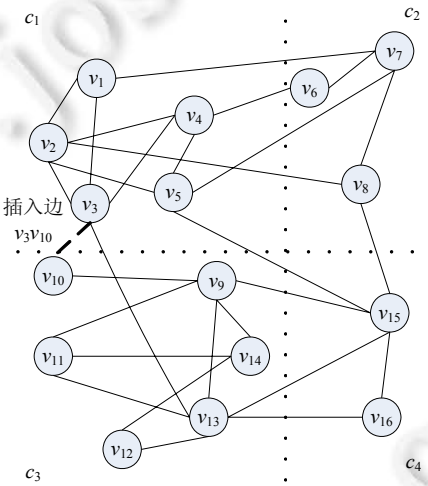

Fig.2 Anexample for adding an edge

图 2 添加边操作示意图

4.2 边删除操作后的更新算法

如算法 4 所示,删除边后的介数中心度更新算法也分为 4 个步骤:首先,通过社区与社区、社区与节点的最短距离集合进行过滤,利用定理 5 快速找到那些最短路径发生变化的点对,即待更新的点对(第 1 行~第 5 行);其次,利用定理 1 计算出待更新点对在更新前的最短路径对介数中心度的影响,并删去这些影响(第 6 行~第 9 行);然后,利用算法 2 和算法 5 计算待更新点对新的最短路径和最短距离(第 10 行),若更新前点对有最短路径不经过被删除的边,则这些路径仍是最短路径,该点对的最短距离不变,最短路径数也可直接算出;最后,利用定理 1 计算待更新点对间新的最短路径对节点介数中心度的影响,并更新社区与社区、社区与节点的最短距离集合(第 11 行~第 16 行).

算法 4. 删除边操作下的介数中心度更新算法.

输入:$G=(V,E),d,d_{cv},d_{cc},\sigma,c_B,(u,v,\omega),C$;

输出:新的 c_B.

1. **for** $(c_i,c_j)\in C\times C$ **do**

2. **if** $d_{cc}(c_i,c_j)_{\max}\geqslant d_{cv}(c_i,u)_{\min}+d_{cv}(v,c_j)_{\min}+\omega$ **or** $d_{cc}(c_i,c_j)_{\max}\geqslant d_{cv}(c_i,v)_{\min}+d_{cv}(u,c_j)_{\min}+\omega$ **then**
3. **for** $(s,t)\in c_i\times c_j$ **do**
4. **if** $d(s,t)\geqslant d(s,u)+d(v,t)+\omega$ **or** $d(s,t)\geqslant d(s,v)+d(u,t)+\omega$ **then**
5. 将点对(s,t)加入集合 *NodePair*,所属的社区对(c_i,c_j)加入集合 *compare*
6. **for** $(s,t)\in NodePair$ **do**
7. **for** $r\in V$ **do**
8. **if** $d(s,t)=d(s,r)+d(r,t)$ **then**
9. $c_B(r)\leftarrow c_B(r)-\sigma(s,r)\times\sigma(r,t)\div\sigma(s,t)$
10. 利用后文算法 5 更新 $d(s,t)$,利用算法 2 重新计算$\sigma(s,t)$.
11. **for** $(s,t)\in NodePair$ **do**
12. **for** $r\in V$ **do**
13. **if** $d(s,t)=d(s,r)+d(r,t)$ **then**
14. $c_B(r)\leftarrow c_B(r)+\sigma(s,r)\times\sigma(r,t)\div\sigma(s,t)$
15. 基于 *compare* 更新 d_{cv},d_{cc}
16. **return** c_B

需要注意的是:在计算新的最短路径时,删除边的操作要比添加边的操作复杂.对于某些点对,它们原有最短路径经过被删除的边,这些点对的最短路径数和最短距离都需重新计算(算法 4 第 10 行).其中,点对间的最短路径数量可由算法 2 得到.因此,需要解决的问题是:在已知图上部分节点的最短距离的情况下,如何获取其他点对的最短距离.虽然我们可以使用朴素的 Floyd 算法,但是这样做耗时巨大.因此,我们对 Floyd 算法进行一些修改.算法 5 中,我们改进的 Floyd 算法仍然采用动态规划的思路,但是通过避免无需改动点对的最短距离的冗余计算,实现了降低部分点对最短距离更新的时间开销.

算法 5. 改进的 Floyd-Warshall 算法.

输入:$G=(V,E)$,d,$NodePair\in V\times V$;

输出:新的 d.

1. **for** $(s,t)\in NodePair$ **do**
2. $d(s,t)\leftarrow\min(INT\_MAX,\omega(s,t))$
3. **for** $u\in V$ **do**
4. **for** $(s,t)\in NodePair$ **do**
5. $d(s,t)\leftarrow\min(d(s,u)+d(u,t),d(s,t))$
6. **return** d

例 4:在图 3 中,我们删除边 $v_{11}v_{13}$,在更新节点介数中心度时,有些社区对可以直接过滤,而有的社区对不能过滤掉.对于社区 c_2,c_4,$d_{cc}(c_2,c_4)_{\max}=4<d_{cv}(c_2,v_{11})_{\min}+d_{cv}(v_{13},c_4)_{\min}+1=3+1+1=5$,$d_{cc}(c_2,c_4)_{\max}<d_{cv}(c_2,v_{13})_{\min}+d_{cv}(v_{11},c_4)_{\min}+1=2+2+1=5$.所以社区 c_2,c_4 间节点构成的点对最短路径不变,直接过滤,不用进一步计算;而对于社区 c_1,c_3,则无法通过社区层面直接过滤,需对每一组点对依次过滤,比如:点对 v_2,v_{12},$d(v_2,v_{12})=3<d(v_2,v_{11})+d(v_{13},v_{12})+1=3+1+1=5$,$d(v_2,v_{12})=3<d(v_2,v_{13})+d(v_{11},v_{12})+1=2+2+1=5$,$v_2,v_{12}$ 的最短路径不变,不用进一步计算;而点对 v_2,v_{14},$d(v_2,v_{14})=4=d(v_2,v_{13})+d(v_{11},v_{14})+1=2+1+1=4$,其最短路径减少,需要更新这些减少的最短路径对节点介数中心度的影响.我们考虑 v_2,v_{14} 的最短路径对于节点 v_{15} 的介数中心度的影响.在删除边 $v_{11}v_{13}$ 之前,v_2,v_{14} 的最短路径有 $v_2v_3v_{13}v_9v_{14}$,$v_2v_3v_{13}v_{11}v_{14}$,$v_2v_3v_{13}v_{12}v_{14}$,$v_2v_5v_{15}v_9v_{14}$ 和 $v_2v_8v_{15}v_9v_{14}$,其中:有 2 条经过节点 v_{15},v_2,v_{14} 的最短路径对于节点 v_{15} 的介数中心度的影响为 2/5;在删除边 $v_{11}v_{13}$ 之后,v_2,v_{14} 中最短路径减少了 $v_2v_3v_{13}v_{11}v_{14}$ 这一条,余下的 4 条中,有 2 条经过节点 v_{15},v_2,v_{14} 的最短路径对于节点 v_{15} 的介数中心度的影响变为 2/4.因此更新之后,点对 v_2,v_{14} 的最短路径的变化使得节点 v_{15} 的介数中心度增加了 2/4−2/5=1/10.其他社区对也可以按照上述方法分析,这里不再赘述.

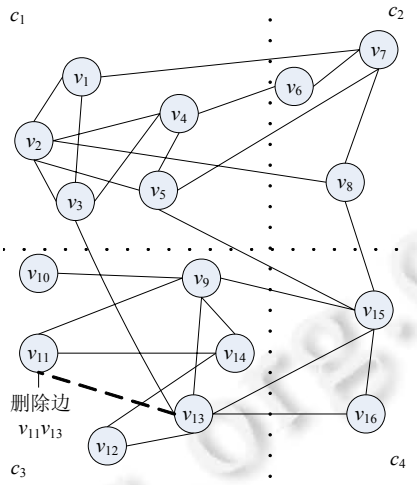

Fig.3 Anexample for deleting an edge

图 3 删除边操作示意图

4.3 复杂度分析

由介数中心度的定义 $c_B(v)=\sum_{s\neq t\neq v,s,t\in V}\frac{\sigma(s,t\mid v)}{\sigma(s,t)}$ 可知,介数中心度是由每组点对的依赖值 $\frac{\sigma(s,t\mid v)}{\sigma(s,t)}$ 累加得到.由定理 2~定理 5 可知:节点介数中心度更新算法 CBU 考虑到了更新过程中所有最短路径变化的点对,重新计算这些点对的依赖值 $\frac{\sigma(s,t\mid v)}{\sigma(s,t)}$, 得到最终的介数中心度.因此,本文所提出的算法是正确的.

在空间复杂度方面,CBU 算法自身需要维护的数据包括网络图、节点最短距离 d、节点最短路径数σ、社区间最短距离 d_{cc} 等相关数据.于是,其空间复杂度为 $O(|V|+|E|+2\times|V|^2+|C|^2+2\times|C||V|)=O(|V|^2)$.

在时间复杂度方面,首先,CBU 算法需要使用社区进行过滤,找出需要计算的点对.这一步消耗的时间为 $O(|C|^2+\boldsymbol{\rho_1}\times|V|^2)$,其中,$\boldsymbol{\rho_1}$ 是社区过滤后仍需检测的点对数占总点对数量的比值;接下来,我们要对找到的点对重新计算最短路径,更新介数中心度,每个点对消耗的时间可以认为是 $O(|V|)$,因此,更新的时间复杂度为 $O(\boldsymbol{\rho_2}\times|V|^2\times|V|)=O(\boldsymbol{\rho_2}\times|V|^3)$,其中,$\boldsymbol{\rho_2}$ 是最终待更新的点对占总点对数量的比值.最终,一次更新操作的时间复杂度为 $O(|C|^2+\boldsymbol{\rho_1}\times|V|^2+\boldsymbol{\rho_2}\times|V|^3)=O(\boldsymbol{\rho_2}\times|V|^3)$.最坏情况下,$\boldsymbol{\rho_2=1}$,即:所有点对都需要重新进行计算,消耗时间 $O(|V|^3)$.而实际网络中,这种极端情况发生的概率比较小.通常情况下,社区内节点联系密切,社区间节点关系较弱,甚至不连通,可以认为$\boldsymbol{\rho_2}<<\boldsymbol{\rho_2}<\boldsymbol{1}$,因此,CBU 消耗的时间是可接受的.

5 实验结果与分析

本节使用真实数据集和合成数据集对所提算法进行实验验证和结果分析.首先介绍实验环境、数据集、评价标准,然后进行性能对比实验和参数敏感性实验.

5.1 实验准备

实验所用操作系统是 Windows Server 2008,CPU 是 Intel Xeon E5-2650,内存为 256GB.本文所提算法及实验代码都基于 C++语言和 VS 2012 实现.

本实验数据集分为合成数据集和真实数据集两部分,见表 1.其中,合成数据集包括:ER 随机图[23],实验中,ER_10000 平均度数为 10,ER_100000 平均度数 20;WS 小世界网络[24],实验中,WS_10000 平均度数为 10,重新连接的概率为 0.5,WS_100000 平均度数为 20,重新连接的概率为 0.5;BA 无标度网络[25],实验中,BA_10000 平均度数为 10,BA_100000 平均度数为 20;社区基准图 ben_10000[26],平均度数为 10;真实数据集[27]包括:来自社交网

络 Facebook 上的真实数据集 ego-Facebook;俄勒冈 2001 年路由网络的信息 Oregon;来自基于位置的社区网络的用户关系 gowalla;Wikipedia 在 2008 年 1 月前的投票关系图 wiki-vote;来自消费者评论网站 Epinions 的真实用户网络 soc-Epininos1;2002 年 8 月时,Gnutella 对等文件共享网络的一份快照数据 p2p-Gnutella25.其中,数据集合 ER_100000,WS_100000,BA_100000 和 gowalla 的数据规模(节点和边的总和)均达到百万.

Table 1 Datasets information

表 1 数据集的统计信息

数据集	节点数	边数	类型
ER_10000	10 000	50 029	无向
ER_100000	100 000	1 000 678	无向
WS_10000	10 000	50 000	无向
WS_100000	100 000	1 000 000	无向
BA_10000	10 000	49 975	无向
BA_100000	100 000	999 900	无向
ben_10000	10 000	49 883	无向
ego-Facebook	4 039	88 234	无向
Oregon	10 670	22 002	无向
gowalla	196 591	950 327	无向
wiki-vote	7 115	103 689	有向
soc-Epininos1	75 879	508 837	有向
p2p-Gnutella25	22 687	54 705	有向

在实验中,我们通过随机增减边的方式来实现网络的动态变化.每次随机增加或删除一条边,我们对得到的新网络计算节点介数中心度,并取计算的平均时间作为算法所消耗的时间.

5.2 评价标准

为了更好地评价本文所提出的 CBU 算法的有效性,针对有向图,我们选择目前最主流的 Brandes 算法[3]作为基准算法;针对无向图,我们选择 Brandes 算法[3]和 Lee 等人[6]提出的无向图上的算法(以下简记为 Lee)当作基准算法.

同时,本文考虑如下 3 个评价指标.

(1) 时间.CBU 算法的目标在于减小动态网络中节点介数中心度的更新计算开销,于是,计算所用时间 T 就成为第 1 个指标;

(2) 加速比.为了更直观地呈现本文所提算法与基准算法的差异,我们用加速比 Ra 的概念来刻画算法效率.加速比指基准算法所用时间与 CBU 算法所用时间的比值,值越大,意味着 CBU 算法的效果越好;

(3) 过滤比.CBU 算法的核心是:利用社区的整体性过滤掉那些不会因为边的增删而在最短路径方面受到影响的点对,从而最终提高介数中心度的计算效率.因此,我们基于定理 2~定理 5,使用过滤比 ρ 来衡量经过社区这层过滤后,所剩点对占总点对的比值.过滤比的值越小,表明算法的过滤效果越好.

5.3 对比实验

社区过滤算法 CBU 和基准算法在不同数据集上的效率对比情况见表 2,其中,Ra 为加速比,T 是计算时间开销(单位为 s).初始计算社区时,本部分采用的社区发现算法是 HANP.由于 Brandes 算法并未从流程上严格区分增加边和删除边,对应这两种操作的计算相同,所以其对应的时间开销也一致,故表 2 在 Brandes 算法的时间开销方面只列出一个值.对于 CBU 算法和 Lee 算法,我们分别给出其在增加与删除边操作下所用的更新时间.需要注意的是:因为 Lee 算法仅适用于无向图,所以在有向图上该算法对应的数据项为空.

Table 2 Efficiency of updating in different datasets

表 2 不同数据集更新效率

数据集	时间 *T*/s					加速比 *Ra*			
	Brandes	Lee(+)	Lee(–)	CBU(+)	CBU(–)	Brandes(+)	Brandes(–)	Lee(+)	Lee(–)
ER_10000	460	237	234	7.96	13.5	58	34	30	17
ER_100000	212 773	88 484	85 345	3 524	9 644	60	22	25	8.9
WS_10000	406	224	244	10.8	12.1	38	34	21	20
WS_100000	238 970	78 627	78 061	7 193	11 794	33	20	11	6.6
BA_10000	445	242	232	7.35	15.1	60	29	32	15
BA_100000	302 499	82 770	91 397	3 834	47 902	78	6.3	22	1.9
ben_10000	572	147	117	5.56	7.06	102	81	26	17
ego-Facebook	324	55.4	65.9	18.8	0.859	17	337	2.9	77
Oregon	241	36.7	51.5	9.86	25.7	24	9.4	3.7	2.0
gowalla	547 191	200 691	243 214	46 791	204 773	12	2.7	4.3	1.2
wiki-vote	107	–	–	1.41	3.3	75	32	–	–
soc-Epininos1	332	–	–	29.8	16.1	11	20	–	–
p2p-Gnutella25	223	–	–	61	84	3.7	2.7	–	–

注:表中(+)代表边添加操作,(–)代表边删除操作;
对于非百万规模数据集,时间为 50 次添加或删除操作的平均值.而百万数据集耗时较长,为 2 次操作的均值;
时间 *T*/s 下的 Lee 表示 Lee 算法所用时间,加速比 *Ra* 下的 Lee 表示 CBU 算法所用是时间与 Lee 算法的比值

表 2 中,CBU 算法显著提高了动态网络中节点介数中心度的更新效率.在合成数据集上,CBU 算法的效率大约是 Brandes 算法的 30 倍,大约是 Lee 算法的 10 倍.其中,添加一条边之后的更新效率略高于删除边之后的更新效率.在 ER 和 WS 中,节点与边的分布比较均匀;ben 对应的分布虽然不够均匀,整个网络是连通的,每个节点都至少与一定数量的边相关联.因此,在这 3 类数据集上添加或删除边所影响的点对数量是相对固定的,且差异不大,即,需要重新计算的点对数量大致相当.考虑到在添加边操作后,可以直接得到点对的新最短路径,耗时较短;而在删除边操作后,可能需要遍历整张图来获得点对的新最短路径,耗时较长.这样的差异导致添加边操作后的更新一般快于删除边操作后的更新.而对于 BA 数据集,整个网络类似一棵树,添加操作影响到的点对少于删除操作影响的点对,添加操作后的更新效率明显更高.

在真实数据集上,CBU 算法在效率上也有明显提高.添加操作和删除操作后,更新效率没有明显的规律.这主要是因为真实数据集中,节点与边的分布不均匀.对于 ego-Facebook 和 soc-Epininos1 数据集,节点平均度数比较高,且聚集系数不低,网络中存在若干规模较大的紧密社区.因此在进行边的添加操作时,容易将这些大的社区连接,使得受影响的点对较多.与此相比,删除操作更容易发生在社区内部,其影响范围有限.这使得添加边操作后的更新效率低于删除边操作后的更新效率.对于 p2p-Gnutella25 和 Oregon 数据集,节点平均度数较低,同时,网络中的边集中于有限的节点上,这使得图中存在大量孤立节点.因此在添加边操作时,容易处理到孤立节点,受影响点对较少,加速效果明显.而删除边的操作则基本出现在网络的核心区域,节点之间联系紧密,于是,删除操作所影响的点对较多,更新效率也随之变慢.对于 wiki-Vote 数据集,有向边的目标节点较为集中,大量节点不存在入边,只有少量出边.于是在添加边操作时,易处理到这些度数较低的节点,受影响的点对较少;而删除边操作会涉及到那些入度较高的节点,受影响的点对偏多,因此,添加边操作的效率高于删除边操作.对于 gowalla 数据集,因为删除边操作后需要遍历整张图来获得受影响点对的新最短路径,同时,gowalla 节点数较大,受影响的点对较多,遍历整张图的成本也就较高,所以删除边操作耗时较增加边操作明显增多.

图 4 则是在数据集 ben_10000 上进行 200 次插入操作后的更新效率.我们将 200 次操作分为 10 个区间,纵坐标为每个区间内更新节点介数中心度所消耗的平均时间,横坐标为更新次数.从图中可以看出,CBU 算法的更新效率比较稳定且明显优于 Brandes 算法和 Lee 算法.

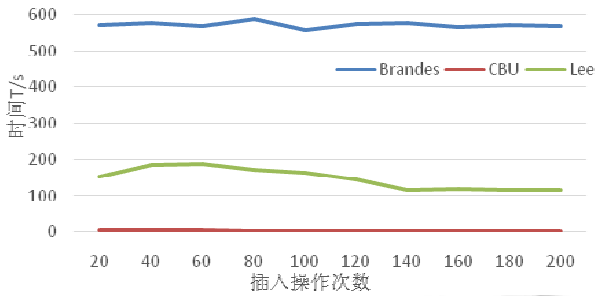

Fig.4 Comparison of the speed after adding edgeson the dataset ben_10000

图 4 在 ben_10000 数据集上插入边效率比较

接下来,我们探究社区过滤效果对于介数中心度更新计算的影响.表 3 展示了 CBU 算法的过滤能力.无论是合成数据集还是真实数据集,社区过滤效果均在 0.25 以下.在合成数据集上,删除操作的社区过滤效果略好,但是无法弥补删除操作后寻找最短路径过程所消耗的时间.在真实数据集 ego-Facebook 上,边添加操作后的社区过滤比高于边删除操作后的社区过滤比;在真实数据集 p2p-Gnutella25 上,边删除操作后的社区过滤比高于边添加操作后的社区过滤比.这与我们之前对于这两个数据集更新操作花费时间的分析是一致的.在真实数据集 gowalla 上,社区过滤效果比较突出,但是由于数据集节点数较大,受影响的点对仍然很多,更新速度的提高实际有限.整体上看,CBU 算法提高了介数中心度的更新速度,其采用的社区过滤策略是有效的.

Table 3 Filtration ratio of community in different datasets

表 3 不同数据集社区过滤比

数据集	ρ(CBU+)	ρ(CBU−)
ER_100000	0.21	0.14
WS_100000	0.13	0.13
BA_100000	0.16	0.06
ben_10000	0.10	0.05
ego-Facebook	0.10	0.05
p2p-Gnutella25	0.07	0.19
gowalla	0.000 3	0.000 7

5.4 参数实验

本节分析节点数量 n 以及社区发现的精度对实验结果的影响.

(1) 节点数量 n

针对 ER 合成图,设置节点的平均度数为 10,分析不同节点数量下,CBU 算法的加速效率.

从图 5 和图 6 可以看出:随着节点数的增加,3 种算法所消耗的时间也在不断增大.但是对于添加操作和删除操作,CBU 算法的时间开销均明显少于 Brandes 算法和 Lee 算法,效率至少提升了一个数量级.这表明,CBU 算法可以很好地提高节点介数中心度的更新效率.

(2) 社区精度的影响

针对表 1 中的合成图 ben_10000,采用不同的社区发现算法,计算出社区结果的精度.然后,在这些不同的社区结果上运行 CBU 算法,分析社区精度对于 CBU 算法效率的影响.

对于社区精度,我们采用的指标是 Modularity,这是 Newman[28]于 2004 年提出的一个定义在[−0.5,1)区间内的指标,由网络图中每个社区内连边数与期待值之差决定.Modularity 越大,社区内部实际连边越是高于随机期望,说明节点越有集中在某些社区内的趋势,即网络的模块化结构越明显,社区划分的精度越高.

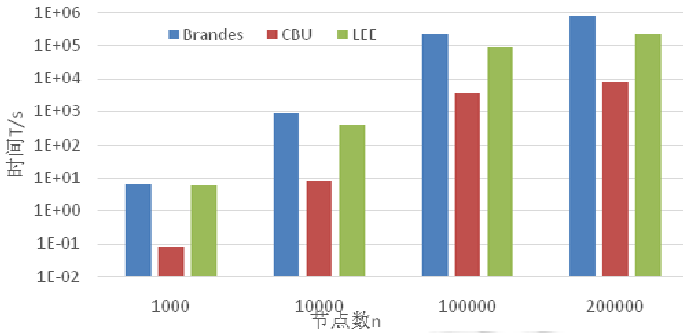

Fig.5 Comparison of the speed after adding an edge

图 5 插入边效率比较

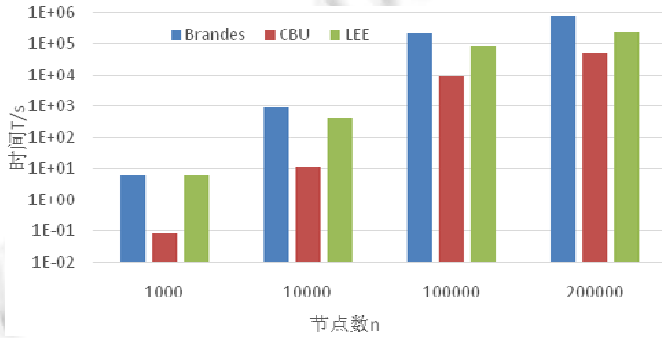

Fig.6 Comparison of the speed after deletingan edge

图 6 删除边效率比较

从表 4 可以看出:随着社区精度的提高,CBU 算法的效率越高,耗时越少.当 Modularity 较大时,提升更为明显,在 CNM 算法得到的社区上进行更新的耗时只有 HANP 的一半.总的来说,社区发现的精度越高,CBU 基于社区的过滤算法效果越好.

Table 4 Efficiency of updating with different communities

表 4 不同社区下的更新效率

社区发现算法	Modularity	时间 *T*/s	
		CBU(+)/s	CBU(−)/s
DSGE[17]	0.14	8.54	11.82
LPA[15]	0.25	7.54	9.30
HANP[16]	0.55	5.56	7.06
CNM[14]	0.61	3.40	3.13

6 结束语

为了解决在实际应用场景中节点介数中心度更新缓慢的问题,本文提出了基于社区的节点介数中心度更新算法 CBU,通过维护社区间最短距离集合、社区与节点的最短距离集合,快速过滤掉在网络更新过程中最短路径不变的点对,大大提高了节点介数中心度的更新效率.未来,我们将尝试利用社区的性质,对节点介数中心度进行近似计算.

References:

[1] Merrer EL, Tredan G. Centralities: Capturing the fuzzy notion of importance in social graphs. In: Proc. of the European Conf. on Computer Systems. 2009. 33−38. [doi: 10.1145/1578002.1578008]

[2] Bader DA, Kintali S, Madduri K, Mihail M. Approximating betweenness centrality. In: Proc. of the Workshop on Algorithms and Models for the Web Graph. 2007. 124−137. [doi: 10.1007/978-3-540-77004-6_10]

[3] Brandes U. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology, 2001,25(2):163−177. [doi: 10.1080/0022250X.2001.9990249]

[4] Sariyüce AE, Saule E, Kaya K, Çatalyürek ÜV. Shattering and compressing networks for centrality analysis. In: Proc. of the Computer Science. 2012..

[5] Lee MJ, Lee J, Park JY, Choi RH, Chung CW. QUBE: A quick algorithm for updating betweenness centrality. In: Proc. of the Int'l Conf. on World Wide Web. ACM Press, 2012. 351−360. [doi: 10.1145/2187836.2187884]

[6] Lee MJ, Choi S, Chung CW. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. In: Proc. of the Information Sciences. 2016. 278−296. [doi: 10.1016/j.ins.2015.07.053]

[7] Brandes U, Pich C. Centrality estimation in large networks. Int'l Journal of Bifurcation and Chaos, 2007,17(07):2303−2318. [doi: 10.1142/S0218127407018403]

[8] Geisberger R, Sanders P, Schultes D. Better approximation of betweenness centrality. In: Proc. of the Algorithm Engineering and Experimentation. 2008. 90−100. [doi: 10.1137/1.9781611972887.9]

[9] Girvan M, Newman ME. Community structure in social and biological networks. Proc. of the National Academy of Sciences of the United States of America, 2002,99(12):7821−7826. [doi: 10.1073/pnas.122653799]

[10] Huang FL, Zhang SC, Zhu XF. Discovering network community based on multi-objective optimization. Ruan Jian Xue Bao/Journal of Software, 2013,24(9):2062−2077 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4400.htm [doi: 10.3724/SP.J.1001.2013.04400]

[11] Radicchi F, Castellano C, Cecconi F, Loreto V, Parisi D. Defining and identifying communities in networks. Proc. of the National Academy of Sciences of the United States of America, 2004,101(9):2658. [doi: 10.1073/pnas.0400054101]

[12] Newman ME. Modularity and community structure in networks. Proc. of the National Academy of Sciences of the United States of America, 2006,103(23):8577−8382. [doi: 10.1073/pnas.0601602103]

[13] Newman ME. Fast algorithm for detecting community structure in networks. Physical Review E, 2004,69(6). [doi: 10.1103/PhysRevE.69.066133]

[14] Clauset A, Newman ME, Moore C. Finding community structure in very large networks. Physical Review E, 2004,70(6). [doi: 10.1103/PhysRevE.70.066111]

[15] Raghavan UN, Albert R, Kumara SR. Near linear time algorithm to detect community structures in large-scale networks. Physical Review E, 2007,76(3). [doi: 10.1103/PhysRevE.76.036106]

[16] Leung IX, Hui P, Lio P, Crowcroft J. Towards real-time community detection in large networks. Physical Review E, 2009,79(6). [doi: 10.1103/PhysRevE.79.066107]

[17] Chen J, Saad Y. Dense subgraph extraction with application to community detection. IEEE Trans. on Knowledge and Data Engineering, 2012,24(7):1216−1230. [doi: 10.1109/TKDE.2010.271]

[18] Huang J, Sun H, Song Q, Deng H, Han J. Revealing density-based clustering structure from the core-connected tree of a network. IEEE Trans. on Knowledge and Data Engineering, 2013,25(8):1876−1889. [doi: 10.1109/TKDE.2012.100]

[19] Perozzi B, Alrfou R, Skiena S. DeepWalk: Online learning of social representations. In: Proc. of the Knowledge Discovery and Data Mining. 2014. 701−710. [doi: 10.1145/2623330.2623732]

[20] Shang JW, Wang CK, Xin X, Ying X. Community detection algorithm based on deep sparse autoencoder. Ruan Jian Xue Bao/Journal of Software, 2017,28(3):648−662 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5165.htm [doi: 10.13328/j.cnki.jos.005165]

[21] Gong M, Li G, Wang Z, Ma L, Tian D. An efficient shortest path approach for social networks based on community structure. CAAI Trans. on Intelligence Technology, 2016,1(1):114−123. [doi: 10.1016/j.trit.2016.03.011]

[22] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. The MIT Press, 2001.

[23] Erdős P, Rényi A. On random graphs I. Publicationes Mathematicae, 1959,6:290−297.

[24] Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. Nature, 1998,393(6684):440−442. [doi: 10.1038/30918]

[25] Barabasi A, Albert R. Emergence of scaling in random networks. Science, 1999,286(5439):509−512. [doi: 10.1126/science.286.5439.509]

[26] Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection algorithms. Physical Review E, 2008, 78(4). [doi: 10.1103/PhysRevE.78.046110]

[27] SNAP datasets. http://snap.stanford.edu/data

[28] Newman ME, Girvan M. Finding and evaluating community structure in networks. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2004,69(2):26−113. [doi: 10.1103/PhysRevE.78.046110]

附中文参考文献:

[10] 黄发良,张师超,朱晓峰.基于多目标优化的网络社区发现方法.软件学报,2013,24(9):2062−2077. http://www.jos.org.cn/1000-9825/4400.htm [doi: 10.3724/SP.J.1001.2013.04400]

[20] 尚敬文,王朝坤,辛欣,应翔.基于深度稀疏自动编码器的社区发现算法.软件学报,2017,28(3):648−662. http://www.jos.org.cn/1000-9825/5165.htm [doi: 10.13328/j.cnki.jos.005165]

钱珺(1993－),男,江苏泰州人,硕士生,主要研究领域为社交网络,社区发现.

郭高扬(1994－),男,博士生,主要研究领域为社交网络,深度学习.

王朝坤(1976－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为图和社交数据管理,音乐计算,大数据系统.