DOI:10.13878/j.cnki.jnuist.2017.05.012

华强胜[1,2]　艾明[1,2]　钱立祥[1,2]　于东晓[1,2]　石宣化[1,2]　金海[1,2]

大规模图中低复杂度分布式算法浅析

摘要

近年来大规模图分析问题在网络大数据领域发挥着重要作用.经典的图分析问题包括求图的直径、半径、围长、聚类系数、紧密中心度和介数中心度等.集中式算法求解这些图计算问题一般都需要问题规模的平方甚至立方以上复杂度,显然不适用于大规模图.本文旨在从分布式算法角度介绍对这些基本图计算问题具有最坏性能保证的低复杂度(线性时间)算法.此外,本文还将介绍如何通过通信复杂性理论证明分布式图计算问题的下界.

关键词

图分析;分布式算法;分布式复杂性;通信复杂性;拥塞模型

中图分类号 TP301.5;TP301.6;TP338.8

文献标志码 A

收稿日期 2017-07-01

资助项目 国家自然科学基金(61572216);中央高校基本科研业务费专项资金(0118210126)

作者简介

华强胜,男,博士,副教授,博士生导师,研究方向为网络分布式算法.qshua@hust.edu.cn

1 华中科技大学 计算机科学与技术学院,武汉,430074

2 华中科技大学 服务计算技术与系统教育部重点实验室,武汉,430074

0 大规模图分析基本问题介绍

大规模图分析在社交网络、交通网络、生物网络等复杂网络中均有重大的应用[1].典型的大规模图分析[1-3]包括求图的直径(Diameter)、半径(Radius)、围长(Girth)、中心度(Centralities)以及聚类系数(Clustering Coefficient)等.

给定n个顶点和m条边的无权无向图$G(V,E)$,图中2节点u和v的距离$d(u,v)$表示u和v之间最短路径长度;图的直径$D:=\max_{u,v\in V}d(u,v)$表示图中任意2点间距离的最大值;图的半径$R:=\min_{u\in V}\max_{v\in V}d(u,v)$表示图中某点到其他所有点最大距离的最小值,该点一般也称为图的中心点(Center);图的围长描绘了图中最短环路的长度;图的中心度用来描述图中每个点的重要性,主要有紧密中心度(Closeness Centrality)和介数中心度(Betweenness Centrality).图中节点v的紧密中心度$CC(v)$定义为点v到其他所有点的平均距离,即$CC(v)=\left(\sum_{u\in V}d(v,u)\right)/(n-1)$.$CC(v)$越小,则表明点$v$能较快达到图中其他节点.图中节点$v$的介数中心度$BC(v)$定义为点$v$居任意2点之间最短路径次数占所有最短路径之比例,可以用来判断图中是否有个重要节点作为整个图的桥梁.节点v介数的具体定义为$BC(v)=\sum_{s\neq v\neq t}\sigma_{st}(v)/\sigma_{st}$,其中点$s$和$t$都是图中异于$v$的点,$\sigma_{st}$是$s$、$t$之间最短路径的个数,$\sigma_{st}(v)$是$s$、$t$之间经过$v$的最短路径的个数.通过每次删除图中最大的$BC(v)$节点可以用来进行大图划分(如GN算法[4]),从而找到相互之间连接紧密的各个社团.基于随机游走的介数中心度是一种更一般的介数中心度,它考虑节点对之间的所有路径而不仅是最短路径.它的具体定义为$RWBC(v)=\sum_{s\neq v\neq t}g_{st}(v)/g_{st}$,其中$g_{st}$表示$s$、$t$之间随机游走的个数,$g_{st}(v)$表示$s$、$t$之间随机游走经过$v$的个数.图中节点$v$的聚类系数$C(v)$(Clustering Coefficient)衡量的是与一个节点有边相连的2个节点之间有边相连的可能性,其具体定义为$C(v)=2t/(k(k-1))$,其中k表示节点v有k个相邻节点,这k个节点之间最多可能存在的边数为$k(k-1)/2$(这种情况发生在这k个节点都相互连接时),t则表示这些节点间实际存在的边数也即和点v相连的三角形的数量.较高的聚类系数说明了“物以类聚”的特性.整个网络的聚类系数等于所有节点聚类系数的平均值.

1 相关集中式算法介绍

1.1 集中式算法求解图的直径和半径

为了求解图的直径和半径,最简单的方法是先计算图中所有点对之间最短路径(All Pairs Shortest Paths, APSP),然后根据图的直径和半径定义,可以很容易求解这2个基本图计算问题.针对无权图的APSP,只需要在 n 个点上进行宽度优先搜索(Breadth First Search, BFS),其时间复杂度为 $O(mn)$.通过利用快速矩阵相乘[5],文献[6]对无向图提出了一个时间复杂度为 $O(n^{\omega}\log n)$ 时间的集中式精确算法.这里 ω 为当前快速矩阵相乘算法所需时间的指数因子.根据文献[5],当前快速矩阵相乘最小的指数因子 $\omega < 2.372\ 864$.

1.2 集中式算法求解图的围长

集中式算法求解图的围长可以通过对图中每个点分别运行一次BFS算法实现,其时间复杂度为 $O(mn)$,因此最坏时间为 $O(n^3)$.基于此,文献[7]提出了一个平均时间为 $O(n^2)$ 时间的方法.如何对图的围长提出一个最坏时间为 $O(n^{3-\varepsilon})$ (ε 为0到1的常数)的集中式算法仍然为开放性问题[8].如果不是求最小环路长度(围长),而是在无权图中找到给定长度环路,则可以找到较快的集中式精确算法.譬如,如果环路长度为偶数,针对无向图,通过在所有点上各运行一次宽度优先搜索算法,文献[9]给出了一个 $O(n^2)$ 时间的偶数长度环路精确求解算法.如果给定环路长度为 k,当 k 为偶数时,文献[10]给出了一个 $O(m^{2-2/k})$ 时间的 k 环路求解算法;当 k 为奇数时,该文也给出了一个 $O(m^{2-2/(k+1)})$ 时间的 k 环路求解算法.

1.3 集中式算法求解图的紧密中心度和介数中心度

对精确求解所有节点紧密中心度,只需要对每个节点执行单源最短路径算法SSSP(Single Source Shortest Paths).因此对无权图其时间复杂度为 $O(mn)$(对 n 个点分别运行一次BFS算法).

对于精确求解所有节点介数中心度,一般做法是第1步对每个节点对(共 n^2 个节点对)求最短路径的长度和个数,其时间复杂度为 $O(mn)$;第2步是对每个节点 v 求 $BC(v)$.因为每个 $BC(v)$ 的求解都需要 $O(n^2)$ 个求和操作,因此该方法的时间复杂度为 $O(n^3)$.Brandes[11] 提出了一个快速计算介数中心度的算法,其时间复杂度为 $O(mn)$.对于稠密图,该时间复杂度和一般方法相同.

1.4 集中式算法求解图的聚类系数

根据聚类系数定义,对每个点 v 求解其聚类系数 $C(v)$ 最重要的是求解和 v 相邻的三角形的个数.对于三角形检测或者计数,一个最简单的算法就是对每个节点所有邻居节点的每2个节点查看是否存在一条边,如果存在一条边则有一个三角形.令 $\deg(v)$ 表示节点 v 的邻居个数,则该算法的时间复杂度为 $\sum_{v\in V}\deg(v)^2 \leqslant 2mn = O(mn)$.该算法对稀疏图譬如 $m = O(n)$ 时的时间复杂度为 $O(n^2)$,但是对于稠密图譬如 $m = O(n^2)$,则时间复杂度 $O(n^3)$.为了降低稠密图的算法复杂度,可以采用快速矩阵相乘法.令 A 为图的邻接矩阵且 $A[i,j] = 1$ 如果节点 i 和节点 j 间有边相连,否则 $A[i,j] = 0$.注意到 $A^2[i,j] = \sum_k (i,k)\cdot(k,j) > 0$ 当且仅当存在一个点 k 使得图中存在2条边 (i,k) 和 (k,j),那么如果图中再存在边 (i,j) 则说明 (i,j,k) 构成一个三角形.所以该算法首先进行矩阵相乘计算 A^2,然后再用 $O(n^2)$ 时间查看对每个大于0的 $A^2[i,j]$ 是否 $A[i,j]$ 也大于0,如果这2个值均大于0,则邻结于节点 i 的三角形数量为 $(\sum_j A^2(i,j))/2$(这里的分母2是因为这些计数的三角形里面每个三角形被计入了2次).目前最快矩阵相乘计算的时间复杂度为 $O(n^{2.372\,864})$[5],所以整个算法的时间复杂度为 $O(n^{2.372\,864})$.

对于稀疏图,通过采用组合算法,还可以得到更快的时间复杂度为 $O(m^{1.41})$ 的集中式算法[10].该算法首先把图中节点按照节点度数分为"low-degree"节点(节点度数<t)和"high-degree"节点(节点度数≥t).对于low-degree节点,我们运行前面描述的对每个节点所有邻居节点的每2个节点进行检查的方法,其时间复杂度为 $O(mt)$,然后再对图中剩下的至多 $O(2m/t)$ 的high-degree节点采用前面所述的快速矩阵相乘算法,其时间复杂度为 $O((2m/t)^{2.372\,864})$.所以算法总的时间复杂度为 $O((2m/t)^{2.372\,864}+mt)$.当 $t = m^{0.407}$ 时,该组合算法的时间复杂度最小,为 $O(m^{1.407})$.

2 分布式算法模型

2.1 为什么采用分布式算法求解图分析问题

从以上分析可以看出,集中式算法求解这些基本的图分析问题都需要平方甚至立方的时间复杂度,即使采用近似算法,其时间复杂度也一般都为 $O(n^2)$ 或者 $O(n^{2-\varepsilon})$ 时间(依近似比而定).这对频繁

变化的大规模图显然不可接受.另外一方面,虽然机器内存越来越大,从而很多大规模图都能够单机内存计算,但是由于分布式算法中各个处理机可以同时运行算法,可以大大降低算法的时间复杂度.

2.2 分布式算法模型

分布式系统是一个有 n 个顶点和 m 条边的无向图.每个顶点表示一个计算实体(节点或进程),每条边对应一个双向通信信道(链路).每个节点是一个状态机,根据其当前状态和收到的消息改变自身状态,并传递消息至邻居节点.分布式算法就是对系统中所有节点对应状态进行部署,所有节点协同改变状态,最终达到预期配置.每个节点都有唯一的标识符,并且知道其邻居节点的标识符.这些信息被编码在每个节点初始状态中.每个节点以同步的方式执行算法流程.系统的全局执行过程按照 $t=0,1,2,\cdots$ 的时间节拍即轮数进行.在每一轮,所有节点向邻居节点发送消息(同时从邻居节点接收消息),并执行本地计算.每轮中计算和消息传递过程需要在本轮内完成.所有消息的大小都有限制.每轮中每条链路上至多能传递 $O(\log n)$ 比特位.系统是可靠的,没有节点宕机,在所有消息传递过程中,没有消息丢失或损坏.

由于该系统模型特别限定了每条链路上能够传递最大消息的大小,该模型被称为拥塞模型(CONGEST model)[12].因为通信延迟为制约分布式计算的主要因素,所以分布式算法代价主要取决于通信开销而忽略各个节点的局部计算时间.分布式算法衡量目标为完成整个计算任务需要的轮数(rounds),也称为分布式算法的时间复杂度.

因为大图分布式算法运行时间由消息传递时间主导(忽略各个节点局部计算时间),所以为了尽可能降低算法的时间复杂度,需要尽可能让更多节点发出的消息并行传输.但是由于网络带宽的限制,如果多条消息经过同一条链路(边),则消息会进行拥塞.因此如何协调每个时刻最大化各个节点并行化操作以及降低链路拥塞是设计低复杂度分布式算法的最大挑战.

3 分布式算法求解图分析问题

3.1 分布式 BFS 算法

从定义可以看出,求解图的直径、半径、围长以及紧密中心度问题都和求解图的 APSP 问题密切相关.在无权无向图中求解 SSSP 可以通过构造 BFS 树实现,因此求解 APSP 可以通过对所有节点均运行一次 BFS 算法完成.在拥塞模型下,文献[2-3,13]均提出了时间复杂度 $O(n)$ 的分布式 APSP 求解算法.其中文献[2-3]的基本思想是运行 n 次 BFS 算法,而文献[13]则提出了更一般的算法.单个分布式 BFS 算法如图 1 所示.

算法 1 Distributed BFS Algorithm

```
1:Initially the root r_0 sets L(r_0) := 0,and all other nodes v set L(v) := ∞
2:The root r_0 sends out the message Layer(0) to all its neighbors
3:while There is a node v which receives a message Layer(d) from a
  neighbor w do
4:    if d+1<L(v) then
5:       parent(v) = w;
6:       L(v) = d+1;
7:       Send Layer(d+1) to all neighbors except w
8:    end if
9:end while
```

图 1 分布式 BFS 算法
Fig. 1 Distributed BFS algorithm

分布式 BFS 算法的描述如下:

1)对于给出的根节点 r_0,令到其距离 $L(r_0):=0$,令其他所有节点 v 到 r_0 的距离为无穷大.(算法 1 第 1 行)

2)根节点 r_0 发送消息 $Layer(0)$ 给其所有的邻居节点.(算法 1 第 2 行)

3)当节点 v 接收到来自其邻居 w 的消息 $Layer(d)$ 后,判断 $d+1$ 是否小于 $L(v)$.如果 $d+1$ 小于 $L(v)$,则将 w 设为其父亲节点,将 $L(v)$ 设置为 $d+1$,并且发送消息 $Layer(d+1)$ 给其所有的除 w 以外的邻居.(算法 1 第 3—9 行)

以图 2 为例说明算法 1 的执行过程.首先选取节点 A 作为图中的根节点,并令 $L(A):=0$,令 $L(B)$、$L(C)$、$L(D)$、$L(E)$、$L(F)$ 均为 ∞.接着根节点 A 向他的邻居 B、C 发送消息 $Layer(0)$.B、C 接收到消息 $Layer(0)$ 后与自己保存的 $L(B)$ 与 $L(C)$ 比较,由于 $L(B)$ 与 $L(C)$ 均为 ∞,大于 $1+0=1$,因此 B 更新自

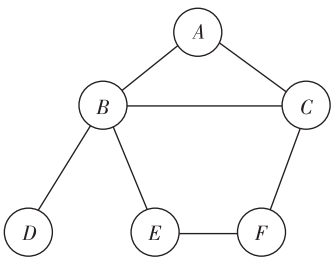

图 2 一个图例
Fig. 2 An illustrationof distributed BFS algorithm

己的父亲节点为 A,$L(B)$ 为 1,C 也更新自己的父亲节点为 A,$L(C)$ 为 1. 之后,B 与 C 同时分别发送 $Layer(1)$ 到自己的邻居节点.D 点与 E 点会接收到来自 B 点的 $Layer(1)$,由于 $L(D)$ 与 $L(E)$ 均为 ∞,大于 $1+1=2$,因此 D 点与 E 点同时将自己的父亲节点设为 B,且更新 $L(D)=2$,$L(E)=2$.F 点会接收到来自 C 点的 $Layer(1)$,由于 $L(F)$ 为 ∞,大于 $1+1=2$,因此 F 点将自己的父亲节点设为 C,且更新 $L(F)=2$.虽然 E、F 之后均会发送消息 $Layer(2)$ 给自己的邻居,但由于每个接收到消息的节点 v 的 $L(v)$ 均小于 $2+1=3$,因此图中没有节点更新列表,也不会再有节点发送消息,至此算法结束,并且算法 1 已构建出一棵以 A 点为根节点的 BFS 树,且每个点都计算出其到根节点 A 的距离.

3.2 分布式 APSP 算法

分析算法 1 可知,算法 1 的时间复杂度为 BFS 树的深度.由于图中构造的 BFS 树的深度不大于图的直径,因此分布式 BFS 算法的时间复杂度为 $O(D)$(D 为图的直径).为了求解 APSP,可以通过顺序运行 n 次 BFS 算法实现,其时间复杂度为 $O(nD)$.文献[2-3]发现了一种可以加快在分布式环境下运行 n 次 BFS 算法的方法.其基本思想不是顺序执行 n 个 BFS 算法,取而代之的是通过合适调度算法来并行执行各个节点上分布式 BFS 算法.本文主要介绍文献[2]的算法,算法过程如图 3 所示.

算法 2 Distributed BFS Algorithm by Parallel BFS

```
1: Choose a node as a root, and build a BFS tree T root on it
2: Send a pebble P from root, and pebble traverses T by DFS
3: while P traverses T do
4:     if P visits a node v for the first time then
5:         Wait one time slot.
6:         Start a BFS_v from node v
7:     end if
8: end while
```

图 3 分布式 APSP 算法[2]

Fig. 3 Distributed APSP algorithm[2]

分布式 APSP 算法[2](算法 2)的描述如下:

1) 在图中随机取一个点作为根节点,以其为根节点建立 BFS 树 T.(算法 2 第 1 行)

2)从根节点发出一个信号 P,P 对树 T 做深度优先搜索(Depth First Search,DFS).(算法 2 第 2 行)

3)当 P 到达一个其未经过的节点 v 时停留一轮,之后以 v 为根节点建立 BFS 树 BFS_v.直到 P 经过图中所有节点时,算法结束.(算法 2 第 3—8 行)

以图 4 为例说明算法 2 的执行过程.首先选取节点 A 作为根节点构建 BFS 树 T.根据算法 2,构建的 BFS 树 T 如图 4 所示.从节点 A 发出信号 P,P 对 T 做深度优先搜索.P 在第 1 轮到达节点 B,由于 P 第 1 次访问节点 B,P 将在 B 点停留 1 轮,之后节点 B 以其为根做 BFS.P 继续做深度优先搜索,并且在第 3 轮到达节点 D.同样,由于节点 D 第 1 次被访问,P 将在 D 点停留 1 轮,之后节点 D 以其为根做 BFS.P 继续做深度优先搜索,并且在第 5 轮到达点 B,由于 B 已经被访问过了,P 不会再在 B 停留,而是继续做深度优先搜索,并且在第 6 轮到达点 E,由于 E 点被 P 第 1 次访问,P 在 E 点停留 1 轮,之后 E 以其为根节点做 BFS.直到 P 访问到点 F,且点 F 做完 BFS 后,由于图中每个点均已被 P 访问,且每个点均做完了 BFS,算法 2 结束.

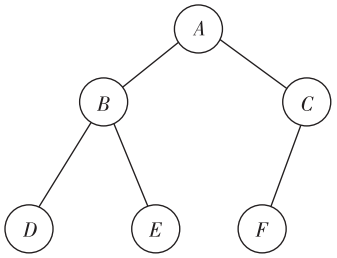

图 4 以 A 为根节点构建的 BFS 树

Fig. 4 A BFS tree rooted on A

分析算法 2 可知,由于 P 通过深度优先搜索访问树 T 的时间复杂度为 $O(n)$,BFS 的时间复杂度为 $O(D)$,因此最后一个点执行完 BFS 的时间为 $O(n+D)=O(n)$.下面我们证明算法 2 的正确性,即证明该算法为什么能满足 CONGEST 模型的限制条件:每一轮中每条边上不会发生消息冲突.更确切的,我们只需要证明图中任意一个点 w,它不可能同时接收任意 2 点 u 和 v 为根的 BFS 树 BFS_v 和 BFS_u 的消息.其原因为,如果一个点接收到了多个 BFS 树访问的消息,根据算法 2 的步骤,下一轮它将发送数目为其接收到 BFS 树访问消息的消息给其邻居.如果这些消息的数目大于 $O(\log n)$,那么将违背 CONGEST 模型的限制,造成拥塞.下面给出具体证明:

假设在算法中,P 先在时间 t_v 访问 v,之后在时间 t_u 访问 u,则 $t_v<t_u$.令 v 与 w 之间的距离为 $d(v,w)$,u 与 w 之间的距离为 $d(u,w)$,u 与 v 之间的距离为 $d(u,v)$,BFS_v 中消息到达 w 的时间为 $t_v(w)$,BFS_u 中消息到达 w 的时间为 $t_u(w)$.根据算法 2,可知 $t_v(w)=t_v+d(v,w)$,$t_u(w)=t_u+d(u,w)$.又 P 是以

深度优先搜索的顺序遍历 T,则 $t_u \geqslant t_v+d(u,v)+1>t_v+d(u,v)$(在步骤3)中,$P$ 到达未经过的节点时要停留1轮时间).结合上面的不等式,我们得到 $t_u(w)=t_u+d(u,w)>t_v+d(u,v)+d(u,w)\geqslant t_v+d(v,w)$(第1个大于基于 $t_u>t_v+d(u,v)$,第2个大于等于基于三角不等式,参见图5),即在 BFS_u 中消息到达 w 的时间恒大于在 BFS_v 中消息到达 w 的时间.也就是说,2棵不同的BFS树到达同一个点的时间不同,因此每条边上不会出现消息冲突,该算法满足CONGEST模型.

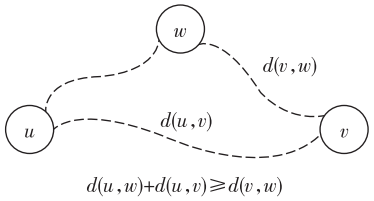

图5 证明中的三角不等式

Fig. 5 The triangle inequality in proof

文献[3]的主要思想与文献[2]类似,先对图做一次DFS,每个节点记录自己被DFS访问的时间,每个点开始BFS的时间即为它们第1次被DFS访问到的2倍.该算法的时间复杂度为 $O(2n+D)=O(n)$.基于文献[2]的思想,我们提出了一种更快计算APSP的算法,其时间复杂度为 $O(n/\log n+D)$[14].该时间复杂度也匹配了无权无向图上分布式计算APSP问题的下界[15],所以为最优算法.

文献[13]提出了一种更普遍的算法,该算法主要解决源检测问题(Source Detection Problem):给出一个特定节点集合,要求求出图中所有其他节点到该集合中每个节点的距离.该算法不仅能够计算APSP,还能够计算多源点最短路径问题(Multi-Source Shortest Paths,MSSP).文献[13]的算法如图6所示.

分布式APSP算法[13](算法3)的描述如下:

1)每个节点 v 初始化一个空列表 L_v,对于列表中的每个元素(每个元素是一个键值对 (d_s,s),其中 s 表示源点,d_s 表示到 s 的距离),设置一个变量 $sent_v$.(算法3第1—2行)

2)对于在源集合 S 中的点 v,初始化 L_v 为((0,v)),其中 $(0,v)$ 表示距离 v 的距离为0,并且将 $sent_v(0,v)$ 设置为FALSE,表示没有发送过键值对 $(0,v)$.(算法3第3—6行)

3)对于每个节点,如果在其保存的 L_v 中存在一个没有被发送过的键值对 (d_s,s),则令 (d_s,s) 为其保存的 L_v 中最小的没被发送过的键值对.之后将该键值对发送给其所有邻居节点,并且设置参数 $sent_v(d_s,s)$ 为TRUE,表示该键值对已被发送.(算法3第7—12行)

算法3 Distributed APSP Algorithm by Source Detection

```
Input: Source set S
Output: Every node in S knows the distances from the other nodes
1: L_v := ( )
2: sent_v : L_v → {TRUE, FALSE}
3: if v ∈ S then
4:     L_v := ((0,v))
5:     sent_v(0,v) := FALSE
6: end if
7: while TRUE do
8:     if ∃(d_s,s) ∈ sent(d_s,s) = FALSE then
9:         (d_s,s) := argmin{(d'_s,s') ∈ L_v | sent_v(d'_s,s') = FALSE}
10:        send (d_s,s) to all neighbors
11:        send_v(d_s,s) := TRUE
12:    end if
13:    for received (d_s,s) from some neighbor do
14:        if ∃(d'_s,s): d'_s ≤ d_s+1 then
15:            L_v := L_v \ {(·,s)}
16:            L_v := L_v ∪ {(d_s+1,s)}
17:            sent_v(d_s+1,s) := FALSE
18:        end if
19:    end for
20: end while
```

图6 分布式APSP算法[13]

Fig. 6 Distributed APSP algorithm[13]

4)如果一个点从其邻居节点收到消息 (d_s,s),则判断保存列表中是否有一个键值对 (d'_s,s) 满足 $d'_s\leqslant d_s+1$.如果不存在这样的 (d'_s,s),则将列表中所有以 s 点为源点的键值对 $(\cdot,s)$ 删除,之后将键值对 (d_s+1,s) 放入列表中,并且设置参数 $sent_v(d_s+1,s)$ 为FALSE.(算法3第13—20行)

以图2为例说明算法3的执行过程.假设给定源点集 $S=\{A,D,F\}$,即利用算法求图中所有节点到点 A、D、F 的距离,算法3的执行过程如下:A、D、F 分别初始化其列表 $L_A=((0,A))$、$L_D=((0,D))$、$L_F=((0,F))$,同时将 $sent_A(0,A)$、$sent_D(0,D)$、$sent_F(0,F)$ 均设置为FALSE.由于节点 A、D、F 均有 $sent_v(d_s,s)$=FALSE,因此节点 A 将 $(0,A)$ 发送给其邻居 B、C,节点 D 将 $(0,D)$ 发送给其邻居 B,节点 F 将 $(0,F)$ 发送给其邻居 C、E,并且 A、D、F 分别将 $sent_A(0,A)$、$sent_D(0,D)$、$sent_F(0,F)$ 设置为TRUE.由于 B 接收到2条消息 $(0,A)$ 与 $(0,D)$,且 L_B 中不存在源为 A 或 D 的键值对,因此节点 B 将 $(1,A)$ 与 $(1,D)$ 放入 L_B 中,此时 $L_B=((1,A),(1,D))$ 且设置 $sent_B(1,A)$ 与

$sent_B(1,D)$ 为FALSE.同样的,节点 C 将 $(1,A)$ 与 $(1,F)$ 放入 L_C,此时 $L_C = ((1,A),(1,F))$,同时设置 $sent_C(1,A)$ 与 $sent_C(1,F)$ 为FALSE,节点 E 将 $(1,F)$ 放入 L_E,此时 $L_E = ((1,F))$ 同时设置 $sent_E(1,F)$ 为FALSE.由于 B、C、E 中有 $sent_v(d_s,s)$ = FALSE,因此节点 B 从 L_B 中选择最小的且未被发送过的键值对 $(1,A)$ 发送给其邻居 A、C、D,节点 C 将 $(1,A)$ 发送给其邻居 A、B、F,节点 E 将 $(1,F)$ 发送给其邻居 B、F.B 虽然接收到来自点 C、E 的消息 $(1,A)$,但是在 L_B 中已存在 $(0,A)$ 同时 $0 < 1 + 1$,因此 B 忽略这2条消息,同时 B 还收到来自 E 的消息 $(1,F)$,由于 B 中不存在关于 F 的键值对,因此节点 B 将 $(2,F)$ 放入 L_B 中,且设置 $sent_B(2,F)$ 为FALSE.同理,节点 D 将 $(2,A)$ 放入到 L_D 中,设置 $sent_D(2,A)$ 为FALSE,节点 E 将 $(2,A)$ 放入到 L_E 中,设置 $sent_E(2,A)$ 为FALSE.节点 F 将 $(2,A)$ 放入到 L_F 中,设置 $sent_F(2,A)$ 为FALSE.重复同样的操作直到图中没有消息发送.最终,可以得到 $L_A = ((0,A),(2,D),(2,F))$,$L_B = ((1,A),(1,D),(2,F))$,$L_C = ((1,A),(1,F),(2,D))$,$L_D = ((0,D),(2,A),(3,F))$,$L_E = ((1,F),(2,A),(2,D))$,$L_F = ((0,F),(2,A),(3,D))$.由结果可知,每个点都得出其到源点集中每个点的距离.

文献[13]分析了该算法的时间复杂度,假设给出的源点集的大小为 k,则该算法的时间复杂度为 $O(k + D)$.该证明的主要思想是证明在第 $O(k + D)$ 轮之后,每个点保存的列表不会发生变化,具体证明可参见文献[13].

运用这些分布式APSP算法,我们可以对图的一些性质进行进行分析.例如,当求解完APSP之后,每个节点均很容易求出其紧密中心度.因此,分布式求出每个点紧密中心度算法的时间复杂度等于求解APSP问题的时间复杂度 $O(n)$.

以分布式APSP算法为基础,还可以分布式求解图的直径、半径、围长和三角形计数等相关图分析问题.下面分别予以介绍.

3.3 分布式求解图的直径和半径

求解图的直径以及半径的算法如图7所示,该算法的具体步骤如下:

1) 分布式计算图的APSP.(算法4第1行)

2) 每个点 v 计算 $ecc(v)$,其中 $ecc(v)$ 代表其他点到 v 点的最大距离.(算法4第2行)

3) 从图中随机选取点 r,以 r 为根建立BFS树.(算法4第3行)

4) 每个点 v 初始化列表 L_v 为 $ecc(v)$.(算法4第4行)

5) 对于每个点 v,当 L_v 不为空集时,任意选择 L_v 中的元素 p,将 p 发送给 v 在BFS树中的父亲节点,同时从 L_v 中删除 p.(算法4第6—9行)

6) 如果 v 从邻居接收到消息 p,v 将 p 添加到 L_v 中.(算法4第10—12行)

7) 根节点 r 将直径 D 设置为 L_r 中最大的元素,将半径 R 设置为 L_r 中最小的元素.(算法4第14行)

8) 利用BFS树广播直径 D 与半径 R.(算法4第15行)

算法4 Distributed Computing Diameter or Radius

```
1: Compute APSP of a graph.
2: Each node v computes ecc(v) := max d(u,v) where u ≠ v ∈ V
3: Randomly choose a node r, and build a BFS tree rooted on r
4: Each node v initializes L_v := (ecc(v))
5: for each node v do
6:     while L_v ≠ ∅ do
7:         Choose p ∈ L_v, and send it to v's parent
8:         L_v := L_v \ {p}
9:     end while
10:    if v receives p from a neighbor then
11:        L_v := L_v ∪ {p}
12:    end if
13: end for
14: Node r computes the diameter as max{D | D ∈ L_r} and computes
    the radius as min{R | R ∈ L_r}
15: Broadcast R and D by BFS tree
```

图7 分布式计算图直径与半径

Fig. 7 Distributively computing graph diameter and radius

该算法第1)步需要 $O(n)$ 轮,第2)、4)、7)步需要 $O(1)$ 轮,第3)、8)步需要 $O(D)$ 轮,第5)—6)步需要 $O(n)$ 轮,因此总时间复杂度为 $O(n)$.

3.4 分布式求解图的围长

分布式求解图围长算法如图8所示,该算法的具体步骤如下:

1)每个节点 v 初始化一个空列表 L_v,对于列表中的每个元素(每个元素是一个键值对 (d_s,s),其中 s 表示源点,d_s 表示到 s 的距离),设置一个变量 $sent_v$.(算法5第1—2行)

2) 对于在源集合 S 中的点 v,初始化 L_v 为 $((0,v))$,其中 $(0,v)$ 表示距离 v 的距离为0,并且将 $sent_v(0,v)$ 设置为FALSE,表示没有发送过键值对 $(0,v)$,初始化关于节点 s 的父节点集合 $parent_s(v)$.(算法5第3—7行)

3) 对于每个节点,如果在其保存的 L_v 中存在一

个没有被发送过的键值对(d_s,s),则令(d_s,s)为其保存的L_v中最小的没被发送过的键值对.之后将该键值对发送给其所有除在$parent_s(v)$里的邻居节点,并且设置参数$sent_v(d_s,s)$为TRUE,表示该键值对已被发送.(算法5第8—13行)

4)当节点v从邻居u接收到消息(d_s,s)时,如果L_v中已经存在键值对(d'_s,s),则初始化$c_s(v)=d'_s+d_s$;否则判断保存列表中是否有一个键值对(d'_s,s)满足$d'_s \leqslant d_s+1$,如果不存在这样的(d'_s,s),则将列表中所有以s点为源点的键值对$(\cdot,s)$删除,之后将键值对(d_s+1,s)放入列表中,设置参数$sent_v(d_s+1,s)$为FALSE,并且将u加入到关于s点的父节点集合$parent_s(v)$中.(算法5第14—25行)

5)每个节点分布式计算出$g_v=\min c_s(v)$.(算法5第26行)

6)运用算法2,每个点并行式广播g_v.(算法5第27行)

7)每个点分布式计算出$g=\min g_v$.(算法5第28行)

分析算法5可知,算法5第1)—4)步需要$O(n)$轮,第5)步需要$O(1)$轮,第6)步需要$O(n)$轮,第7)步需要$O(1)$轮.因此算法5的时间复杂度为$O(n)$.

3.5 分布式求解三角形个数

分布式计算三角形个数如图9所示,该算法的具体步骤如下:

1)每个节点将其连接的邻居信息传给其所有邻居.(算法6第1行)

2)每个节点v初始化变量$count_v$为0.(算法6第2行)

3)对于每个节点v,如果存在邻居节点u、w满足u、w之间有一条边相连,则将$count_v$增加1.(算法6第3—7行)

4)随机选取一个节点r,以r为根构建BFS树.(算法6第8行)

5)如果一个节点v是BFS树叶子节点,则将$count_v$发送给BFS树中的父亲节点.(算法6第9—11行)

6)当一个节点u接收到来自其孩子节点v的消息$count_v$时,u将其保存的$count_u$加上$count_v$,如果节点u已经接收到所有其孩子发送的消息,u将$count_u$发送给其在BFS树中的父亲节点.(算法6第12—17行)

算法5 Distributively Computing Girth

```
1: L_v := ( )
2: sent_v : L_v → {TRUE, FALSE}
3: if v ∈ V then
4:     L_v := ((0, v))
5:     sent_v(0, v) := FALSE
6:     parent_s(v) = ( )
7: end if
8: while TRUE do
9:     if ∃(d_s, s) ∈ L_v; sent(d_s, s) = FALSE then
10:        (d_s, s) := argmin{(d'_s, s') ∈ L_v | sent_v(d'_s, s') = FALSE}
11:        send(d_s, s) to all neighbors except for nodes in parent_s(v)
12:        sent_v(d_s, s) := TRUE
13:    end if
14:    for v received (d_s, s) from some neighbor u do
15:        if ∃(d'_s, s) then
16:            c_s(v) := d_s + d'_s
17:        end if
18:        if ∄(d'_s, s): d'_s ≤ d_s + 1 then
19:            L_v := L_v \ {(·, s)}
20:            L_v := L_v ∪ {(d_s + 1, s)}
21:            sent_v(d_s + 1, s) := FALSE
22:            parent_s(v) = parent_s(v) ∪ {u}
23:        end if
24:    end for
25: end while
26: g_v = min c_s(v)
27: Using Algorithm 2 to broadcast g_v
28: Each node computes girth g = min g_v where v ∈ V
```

图8 分布式求解围长
Fig. 8 Distributively computing girth

7)图中总三角形个数为$count_r/3$.(算法6第18行)

算法6 Distributively Counting the Number of Triangles

```
1: Each node tranfers which node it connects to its neighbors
2: Each node v intializes count_v := 0
3: for each node v do
4:     if (v, u) ∈ E, (v, w) ∈ E and (u, w) ∈ E then
5:         count_v := count_v + 1
6:     end if
7: end for
8: Randomly choose a node r, and build a BFS tree root on r
9: if Node v is the leaf of BFS tree then
10:    v sends count_v to its parent
11: end if
12: if Node u receives count_v from its child v then
13:    count_u := count_u + count_v
14:    if Node u receives all its children's messages then
15:        u sends count_u to its parent
16:    end if
17: end if
18: The total number of triangles is count_r/3
```

图9 分布式求解三角形个数
Fig. 9 Distributively counting the number of triangles

该算法第1)步需要 $O(n/\log n)$ 轮,因为每个点可将其邻居信息视为一个 n 位的数(其中每一位代表图中的一个点,如果该点与其相连,则将该位置为"1",否则置为"0"),而每一轮每条边只可传递 $O(\log n)$ 位数据,因此第1)步需要 $O(n/\log n)$ 轮.该算法第2)、3)、7)步需要 $O(1)$ 轮,第4)、5)、6)步需要 $O(D)$ 轮,因此,该算法总时间复杂度为 $O(n/\log n+D)$ 轮.

3.6 分布式求解介数中心度

文献[16]提出了首个分布式计算介数中心度的方法,该方法可以看成文献[11]提出的 Brandes 算法的分布式版本.通过解决分布式过程中最短路径信息聚合问题和最短路径条数可能为指数问题,文献[16]的分布式算法时间复杂度为 $O(n+D)$.文献[16]还证明了分布式计算介数中心度的下界为 $\Omega(D+\frac{n}{\log n})$,因此所提算法为近似最优.此外,文献[17]还对随机游走介数中心度问题提出了首个分布式近似算法(近似比为 $(1-\varepsilon)$,ε 为小常数).该方法主要基于矩阵分析理论,时间复杂度为 $O(n\log n+D)$.通过该文证明的分布式计算随机游走介数中心度的下界 $\Omega(D+\frac{n}{\log n})$,该文所提分布式算法也为近似最优.

4 分布式图计算下界

4.1 分布式下界求解概述

分布式计算图问题需要通过消息传递来了解其他点的状态,并且很多图分析问题的计算甚至需要了解图中最远节点的信息.因此 $\Omega(D)$ 轮的时间显得不可避免.如果消息的大小不受限制,那么任何图问题都可以在 $O(D)$ 时间下解决.一个自然的问题是,在拥塞模型下,解决这些图计算问题最少需要多少时间?这也就是分布式图计算下界问题.文献[18]说明验证一个图是否存在某个性质可以用于证明计算这个图性质所需的下界.例如文献[15]需要回答这样一个问题的下界"一个图的直径是多少".作者给出了问题"这个图的直径是否大于4"的下界,该验证问题的下界同时表明了原计算问题的下界.因此,对于一个图计算问题,我们可以设计相应的验证问题来转化下界的求解.而对于一个验证问题的下界求解,可以归约为通信复杂度的下界问题:通过建立图问题和通信复杂度问题之间的联系,将图问题归约为通信复杂度问题,从而使用通信复杂度问题的下界去界定图问题的下界.

4.2 通信复杂度

通信复杂度由姚期智先生在文献[19]中首次提出.本文我们主要以双方通信复杂度模型为例来阐述一个基本图计算问题的分布式下界.

在双方通信复杂度模型中,有2个无限计算能力的个体,称为 Alice 和 Bob.每个人分别拥有一个 k 比特的字符串 $a\in\{0,1\}^k$ 和 $b\in\{0,1\}^k$.Alice 和 Bob 可以互相通信用于计算出一个函数 $h(a,b):\{0,1\}^k\times\{0,1\}^k\to\{0,1\}$,但是每次只能发送1 bit 的消息.通信复杂度是用于界定 Alice 和 Bob 计算出函数 h 所需要的比特数.假设 Alice 和 Bob 都采用公共的随机策略(public randomness),记为 pub.设 A_ε 为错误概率为 ε 的算法集合,$A\in A_\varepsilon$ 是 Alice 和 Bob 计算 h 的算法.则其在输入 a 和 b 的情况下,计算 h 的通信复杂度定义为:

$$R_\varepsilon^{\text{cc-pub}}(h)=\min_{A\in A_\varepsilon}\max_{a,b\in\{0,1\}^k}R_\varepsilon^{\text{cc-pub}}(A(a,b)).$$

在通信复杂度中有大量已经被证明的下界问题,其中最常用的是集合不相交问题(set disjointness problem).这个问题是判断2个大小为 k 的集合是否相交,即字符串 a 和 b 是否有某个相同的位都是1.它的标准定义为

$$Disj_k(a,b)=\begin{cases}0, & \exists i\in[0,k-1]\text{ such that }a[i]=b[i]=1,\\ 1, & \text{otherwise.}\end{cases}$$

对于这样一个通信复杂度问题,参考文献[20]中给出了下界:对于一个足够小的 $\varepsilon>0$,$R_\varepsilon^{\text{cc-pub}}(Disj_k)\geqslant k$,即求解集合不相交问题最少需要 k 比特的消息.

在图算法中,时间复杂度是用轮数来衡量的,而图计算问题和通信复杂度问题看起来又相去甚远,如何建立两者之间的联系需要对不同的图计算问题进行细致的考量和分析.

4.3 分布式图计算下界求解的一般框架

本节将以文献[15]的分布式计算图的直径问题为例,介绍该文提出的一个分布式计算图问题下界的框架.最近诸多图计算问题的分布式下界证明基本都采用了此框架,譬如文献[16-17]等.

对于双方通信复杂度模型,存在2个计算能力无限的个体 Alice 和 Bob,因此,在图问题中,需要将图 G 表示为2个不相交的部分 G_a 和 G_b,分别代表这2个个体.将连接 G_a 和 G_b 的边称为割边,其构成的集合称为割集 C_k.根据这种分割方法,可以定义

待求解的分布式图计算问题 f 和通信复杂度问题 f' 的关系如下:

$f'((G_a,C_k),(G_b,C_k))=f(G)$.

因此,对于任意的图 G,当计算 $f'((G_a,C_k),(G_b,C_k))$ 时,Alice 得到输入 (G_a,C_k),Bob 得到输入 (G_b,C_k).假设 Alice 和 Bob 模拟算法 A,$M^a(r)$ 是 Alice 使用算法 A 时需要在第 r 轮发送的消息,$M^b(r)$ 是 Bob 使用算法 A 时需要在第 r 轮发送的消息.Alice 和 Bob 在图 G 上计算 f' 的过程如下:

1)Alice 在 G_a 上执行算法 A,产生需要发送给 Bob 的消息集合 $M^a(r)$,然后通过 C_k 将集合在第 r 轮发送给 Bob.

2)Bob 在 G_b 上执行算法 A,产生需要发送给 Alice 的消息集合 $M^b(r)$,然后通过 C_k 将集合在第 r 轮发送给 Alice.

对于一个通信复杂度问题 f' 中需要计算的函数 $h:\{0,1\}^k\times\{0,1\}^k\to\{0,1\}$,可以建立下面的归约:

$\mathrm{Red}:\{\mathrm{Alice},\mathrm{Bob}\}\times\{0,1\}^k\to\{(G_a,G_b,C_k)\}$.

因此,对于函数 h 的输入 a、b,我们得到 $h(a,b)=f'(\mathrm{Red}(\mathrm{Alice},a),\mathrm{Red}(\mathrm{Bob},b))$.结合 f' 到 f 的归约,可以通过 f' 建立函数 h 到图计算问题 f 上的联系.设 $R_\varepsilon^{\mathrm{dc\text{-}pub}}(f)$ 为分布式图计算问题 f 所需要的时间复杂度,$c_k=|C_k|$,B 为每一轮每条边最多可以传输的比特数,则可得到:

$$\frac{R_\varepsilon^{\mathrm{cc\text{-}pub}}(f')}{2c_kB}\leqslant R_\varepsilon^{\mathrm{dc\text{-}pub}}(f). \tag{1}$$

这个不等式表明,对于一个图计算问题 f,如果可以将其归约到通信复杂度问题 f',那么就可以建立起时间复杂度和通信复杂度之间的联系,从而计算出图计算问题的下界.这个框架可以用图 10 说明.

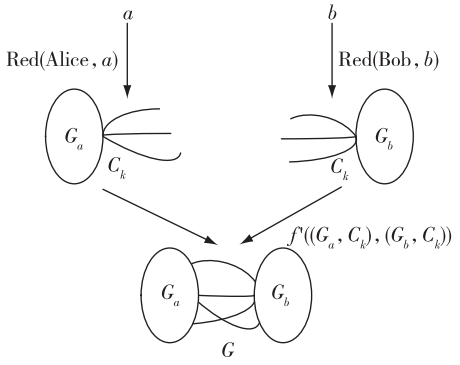

图 10 图问题和通信复杂度问题归约的一般框架

Fig. 10 A reduction framework between distributed graph computing and communication complexity

4.4 分布式计算图直径下界

根据图 10 所给的下界求解框架,这一节会具体描述如何证明分布式计算图直径的下界.

对于图直径计算问题,首先考虑如何将该问题表述为可以归约的形式.从前文描述中可知,通信复杂度需要计算一个确定的函数 h,而图计算并没有这样的形式.我们无法事先得知计算出来的直径到底是多少.但是对于这样的问题"确定一个图的直径是否小于 3"却是可以表示的.通过文献[18]中的结论可知,一个验证问题的下界可以表示为图计算问题的下界.这个可以理解为,确定图的直径是否小于 3 的时间小于计算图直径的时间,从而其下界可以用于表示计算图直径的下界.通过这样的转化,我们只需要求解"确定一个图直径是否小于 3"这个问题的下界.定义这个问题为 $Diam_3(G)$,则其形式化的定义为:

$$Diam_3(G)=\begin{cases}1, & Diam(G)<3,\\ 0, & \text{else.}\end{cases}$$

接下来设计 f 到 f' 的归约.如前文所述,这里使用通信复杂度里的经典问题"集合不相交问题"作为通信复杂度 f' 计算的函数 h,即需要将 $Diam_3(G)$ 归约为 $Disj_k(a,b)$.我们使用文献[15]中的归约方法:构建一个直径为 3 的图,然后通过在图 G_a 和 G_b 上加边的方式改变其直径.而边的连接方式由字符串 a 和 b 控制,这样将直径的判断和 $Disj_k(a,b)$ 联系起来.图的构造方式如下:

1)构建 2 个子图 G_a 和 G_b,其中 G_a 由点 $l_0\cdots l_{k-1}$,$l_k\cdots l_{2k-1}$ 组成.G_b 由点 $r_0\cdots r_{k-1}$,$r_k\cdots r_{2k-1}$ 组成.连接 (l_i,l_j),$0\leqslant i<j\leqslant k-1$,使得 $l_0\cdots l_{k-1}$ 组成一个团(两两之间有边相连);同样也将 $l_k\cdots l_{2k-1}$ 连成一个团.对 G_b 中的点也做相应的连边操作.

2)连接每对 (l_i,r_i),使 G_a 和 G_b 构成一个图 G.

3)新建点 c_L,c_R,连接 (c_L,c_R),(c_L,l_i),(c_R,r_i) 使图 G 变为一个连通图.

在这个构建的图 G 中,直径为 3.接着通过 a 和 b 来加边,使图的直径发生改变.具体步骤如下:

对于字符串 $a\in\{0,1\}^{2k}$ 和 $b\in\{0,1\}^{2k}$,如果 $a[i]=0$,则连接边 $(l_{i\bmod k},l_{k+\lfloor i/k\rfloor})$,如果 $b[i]=0$,则连接边 $(r_{i\bmod k},r_{k+\lfloor i/k\rfloor})$.

如果 $a[i]=0$ 或者 $b[i]=0$,导致点 $l_{i\bmod k}$ 到 $r_{k+\lfloor i/k\rfloor}$ 的距离从 3 变为 2.即如果没有 $a[i]=b[i]=$

1,那么图的直径将会由 3 变为 2.图 11 给出了一个简单的示例.

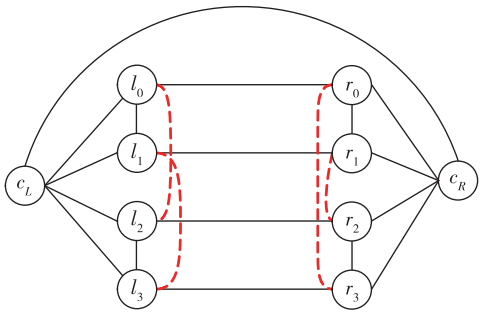

图 11　此图 $k=2$, $a=[0,1,1,0]$, $b=[1,0,0,1]$. 红色的虚线为根据 a 和 b 添加的边,图中不存在 $a[i]=b[i]=1$,因此图的直径由 3 变为 2

Fig. 11　An illustrating reduction example between distributed diameter computation and set disjointness

构造这样的一个图 G 后,就建立了如框架中所述的 $f'((G_a,C_k),(G_b,C_k))$.因为图中边的连接由 a 和 b 确定,我们可以构造如下的归约:Alice 的输入为 a,Bob 的输入为 b.Alice 和 Bob 通过互相发送消息来确定 a 和 b 是否有某个相同位置都为 1.即 $Disj(a,b)=f'(\mathrm{Red}(\mathrm{Alice},a),\mathrm{Red}(\mathrm{Bob},b))=f'((G_a,C_k),(G_b,C_k))$.而 a 和 b 是否相交可以推导出图的直径是否小于 3,更确切的表示为:

$$Diam_3(G)=\begin{cases}1, & Disj(a,b)=1,\\ 0, & \text{else.}\end{cases}$$

至此,我们建立了 $Disj_{k^2}(a,b)$ 到 $Diam_3(G)$ 之间的联系.根据式(1),可以得到 $\frac{R_\varepsilon^{\text{cc-pub}}(Disj_{k^2}(a,b))}{2c_kB}\leqslant R_\varepsilon^{\text{dc-pub}}(Diam_3(G))$.根据文献[15],有 $R_\varepsilon^{\text{cc-pub}}(Disj_{k^2}(a,b))\geqslant k^2$, $c_k=k$, $B=\log n$,从而 $R_\varepsilon^{\text{dc-pub}}(Diam_3(G))\geqslant k/\log n$.因为 $k=O(n)$,所以问题 $Diam_3(G)$ 的下界为 $\Omega(D+\frac{n}{\log n})$.

如前文所述,确定一个图的直径是否小于 3 的难度小于计算图的直径,因此其下界也要小于计算图直径的下界.可以用 $R_\varepsilon^{\text{dc-pub}}(Diam_3(G))$ 来表示计算图直径的下界得到最终结论:对于采用拥塞模型的任意分布式算法,计算图直径至少需要 $\Omega(D+\frac{n}{\log n})$ 轮的时间.

5　未来工作

本文所提分布式算法均基于无权无向图,并不能直接应用到有向图或者有权图中.在有向图中,由于边存在方向,因此图中 2 点 u、v 之间的距离可能不对称,即点 u 到点 v 的距离可能不等于点 v 到点 u 的距离,可能出现 u 到 v 可达而 v 到 u 不可达的情况,从而导致问题发生变化.在有权图中,边存在权重,而每条边的权重不尽相同,假设 2 点 u、v 之间存在 2 条路径,则可能出现其中一条路径的总权重相对较小,而边的条数则较多,而另一条路径则恰好相反的情况.假设按照无权图建立 BFS 树操作在有权图上计算 u、v 之间的距离,如果以 u 为根建立 BFS 树,则可能存在先到达 v 点的路径权值反而更大,导致结果错误.因此,分布式图算法的未来挑战性的工作将是对有向图与有权图中的图计算问题提出低复杂度分布式算法.

本文介绍的主要是精确计算这些图分析问题.如何在算法复杂度和求解结果之间进行折中,即分布式近似计算这些图问题也是一个有意义的工作.

分布式图精确计算或近似计算下界问题的研究目前还处于起步阶段,如何对分布式图计算提出更好的下界证明技术和方法将是分布式复杂性领域的重要挑战.

参考文献

References

[1] 汪小帆,李翔,陈关荣.网络科学导论[M].北京:高等教育出版社,2012
WANG Xiaofan, LI Xiang, CHEN Guanrong. Network science: An introduction [M]. Beijing: Higher Education Press, 2012

[2] Holzer S, Wattenhofer R. Optimal distributed all pairs shortest paths and applications [C] // ACM Symposium on Principles of Distributed Computing, 2012: 355-364

[3] Peleg D, Roditty L, Tal E. Distributed algorithms for network diameter and girth [C] // Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming, 2012: 660-672

[4] Girvan M, Newman M E J. Community structure in social and biological networks [J]. Proceedings of the National Academy of Sciences, 2002, 99(12): 7821-7826

[5] Le Gall F. Powers of tensors and fast matrix multiplication [C] // Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, 2014: 296-303

[6] Seidel R. On the all-pairs-shortest-path problem in unweighted undirected graphs [J]. Journal of Computer and System Sciences, 1995, 51(3): 400-403

[7] Itai A, Rodeh M. Finding a minimum circuit in a graph [J]. SIAM Journal on Computing, 1978, 7(4): 413-423

[8] Roditty L, Tov R. Approximating the girth [J]. ACM-SIAM Symposium on Discrete Algorithms, 2011, 9(2): 1446-1454

[9] Yuster R,Zwick U.Finding even cycles even faster[C]// Proceedings of the 21st International Colloquium on Automata,Languages and Programming,1994:532-543
[10] Alon N,Yuster R,Zwick U.Finding and counting given length cycles[J].Algorithmica,1997,17(3):209-223
[11] Brandes U.A faster algorithm for betweenness centrality [J].Journal of Mathematical Sociology,2001,25(2): 163-177
[12] Peleg D.Distributed computing:A locality-sensitive approach[M].Society for Industrial and Applied Mathematics,2000
[13] Lenzen C,Peleg D.Efficient distributed source detection with limited bandwidth[C]// Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing,2013:375-382
[14] Hua Q S,Fan H,Qian L,et al.Brief announcement:A tight distributed algorithm for all pairs shortest paths and applications[C]// Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, 2016:439-441
[15] Frischknecht S,Holzer S,Wattenhofer R.Networks cannot compute their diameter in sublinear time[C]// Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms,2012:1150-1162
[16] Hua Q S,Fan H,Ai M,et al.Nearly optimal distributed algorithm for computing betweenness centrality[C]// IEEE 36th International Conference on Distributed Computing Systems,2016:271-280
[17] Hua Q S,Ai M,Jin H,et al.Distributively computing random walk betweenness centrality in linear time[C]// IEEE 37th International Conference on Distributed Computing Systems,2017:764-774
[18] Sarma A D,Holzer S,Kor L,et al.Distributed verification and hardness of distributed approximation[J].SIAM Journal on Computing,2012,41(5):1235-1265
[19] Yao A C C.Some complexity questions related to distributive computing (preliminary report)[C]// Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing,1979:209-213
[20] Kushilevitz E.Communication complexity[J].Advances in Computers,1997,44:331-360

Low-complexity distributed algorithms on large-scale graphs:A brief review

HUA Qiangsheng[1,2] AI Ming[1,2] QIAN Lixiang[1,2] YU Dongxiao[1,2] SHI Xuanhua[1,2] JIN Hai[1,2]
1 School of Computer Science & Technology,Huazhong University of Science & Technology,Wuhan 430074
2 Services Computing Technology and System Lab,Huazhong University of Science & Technology,Wuhan 430074

Abstract In recent years,large-scale graph analysis has played an important role in big data computing.The classical graph analysis problems include computing the graph diameter,the radius,the girth,the clustering coefficient and various centrality indices.To solve these problems,centralized algorithms generally require square or even cubic time complexity,which is obviously not applicable to large-scale graphs.In this paper,we aim to briefly review some low complexity (linear time) algorithms for these basic graph problems from a perspective of distributed algorithms. In addition,this paper also shows how to prove the lower bound of distributed graph computing by utilizing the communication complexity theory.

Key words graph analysis;distributed algorithm;distributed complexity;communication complexity;CONGEST model