

# 基于双连通分量覆盖图的稀疏大图 最大流并行加速方法

刘扬, 魏蔚, 许贺洋

(河南工业大学 信息科学与工程学院, 河南 郑州 450001)

**摘要:**最大流问题是图论中重要的基础性问题,大规模网络中的最大流加速已成为重要研究方向,已有工作包括并行计算加速和图缩减加速2种思路,但仍有较大改进空间:①图缩减和并行计算2种加速思路并未充分融合,导致各自加速效果受限;②已有加速算法对常见的多次最大流求解支持不足,导致多次计算间存在大量冗余工作;③已有加速算法往往需涉及出入度和边容量等多个条件,计算复杂度偏高。针对上述问题,提出了一种基于优化子图的最大流并行加速方法,通过识别原始大图的双连通分量并建立覆盖图,可将任意最大流问题分解为独立的子问题,并行求解快速获取最大流精确解;覆盖图的构建仅涉及节点之间连接关系,具较低的时间复杂度。在基准图上的测试结果表明,算法可显著缩短稀疏大图中最大流计算时间。

**关键词:**计算复杂度;图理论;最大流问题;稀疏图计算;双联通分量;覆盖图;并行计算

**中图分类号:** TP301

**文献标志码:** A

**文章编号:** 1000-2758(2018)05-0955-08

作为图数据之上一类经典的组合优化问题,最大流问题寻找通过一个流通网络的最大流量,是网络流理论体系中重要的研究领域<sup>[1]</sup>。最大流可辅助求解一些重要的基础理论问题,如双边匹配等线性规划问题,以及一些重要的应用问题,如网络规划<sup>[2]</sup>,网络和信息安全<sup>[3]</sup>和各类复杂系统中的资源调度<sup>[4]</sup>等。经典的最大流算法主要包括增广路<sup>[5]</sup>和预流推进<sup>[6]</sup>2种方法,并在2种方法基础上衍生出了很多变种以降低计算复杂度。已知最大流加速方法包括针对有向平面图的快速算法<sup>[7]</sup>,针对计算机视图数据的快速计算方法<sup>[8]</sup>,以及基于线性代数或电路流的最大流近似计算方法等<sup>[9-10]</sup>。但随着各行业数据呈爆炸性增长,传统加速方法已无法应对大规模图上的最大流问题。针对大规模图中最大流加速,已有方法主要集中在图缩减和并行计算2个方面:

1) 图缩减思路:通过化简原始网络,降低问题规模并减少后续重复计算。如针对单源单汇最大流问题,文献[11]提出基于图缩减机制的两阶段最大流方法,针对几种局部特殊拓扑结构,将满足给定代数关系的子图缩为节点,在简化的图上进行快速计算。文献[12]通过多次最大流计算,构建原始图的最小割树,以丢失路径信息的代价,快速求解任意2个节点之间的最大流值。文献[13-14]检测满足缩减标准的网络社区结构,将其压缩为一点,在简化的图上获取最大流近似解。文献[15]寻找合并后不影响外部最大流的节点对,通过多次迭代合并图中的节点对缩小图的规模。文献[16]利用商空间理论建立问题求解的保真和保假原理,将图中所有节点对之间最大流计算量从 $n(n-1)/2$ 次降低为 $n-1$ 次左右。

2) 并行计算思路:基于底层的多种并行架构加

**收稿日期:** 2017-09-06

**基金项目:** 国家自然科学基金(61472460, U1504607, 61702162)、河南省高校科技创新团队支持计划(17IRTSTHN011)、河南省教育厅科学技术研究重点项目(17A520004)、粮食信息处理与控制教育部重点实验室开放基金课题(KFJJ-2016-104)、河南省科技厅科技攻关项目(172102110013)、河南工业大学高层次人才基金(2017025)、河南工业大学骨干教师(2012012)及河南工业大学校基金(2018RCJH07, 2018QNJH26)资助

**作者简介:** 刘扬(1978—),女,河南工业大学副教授、博士,主要从事分布式计算及图计算研究。

速求解过程。早期工作主要集中在实现多处理器环境中的并行算法<sup>[17-18]</sup>。随着云计算技术的发展,出现了基于多种通用和专用并行计算模型的加速方法。其中,文献[19]实现了基于 Spark 框架的 Edmonds-Karp 增广路算法,在算法各步骤采取不同计算模型,利用框架中内置的 pregel 和 mapreduce 等多种计算接口提升算法并行度。文献[20]针对包含上亿节点和数十亿条边的社交网络数据,在算法和 mapreduce 框架层进行跨层优化,将最大流计算时间缩短至 10 分钟左右。

已有工作中从多个角度对最大流问题进行了探讨,但仍存在一些问题:①图缩减和并行计算 2 种加速思路并未充分融合,基于单个思路的加速效果受限;②未考虑常见的单个图中多次最大流求解场景,多次计算间存在大量冗余工作;③图缩减方法需要涉及出入度和边容量等多个条件,计算复杂度偏高,甚至会抵消并行计算加速效果。针对上述问题,我们尝试利用双连通分量构建覆盖图,提出了稀疏大图基于双连通分量覆盖图 (BCR: bi-connected component overlay) 的并行加速方法。对原始图中任意两点间的最大流问题,基于覆盖图分解成多个子图内最大流计算问题,并行求解这些子问题。覆盖图仅涉及连接关系,对局部拓扑结构没有任何要求,计算复杂度较低。在基准网络拓扑上的实验结果表明,算法可显著缩短计算时间,大幅提升求解效率。

## 1 模型

### 1.1 试验方法

算法主要处理无向图中的最大流问题,首先引入连通分量和双连通分量的定义。

**定义 1** 连通分量:在无向图中,若顶点 1 和顶点 2 之间存在路径,则称顶点 1 和 2 连通。若某个子图中任意 2 个顶点之间都连通,则称该图为连通图,其中极大连通子图称为连通分量。

**定义 2** 双连通分量:若任意图中任意 2 点至少存在 2 条不存在重复点的路径,则称该图为双连通图;对一个无向图,双连通的极大子图称为双连通分量。

图 1 示例中,左侧虚线包含的子图为双连通分量中。双连通分量和割点有紧密关系,下面引入割点和双连通分量界点的定义。

**定义 3** 割点:如在图 G 中去掉一个顶点,并同

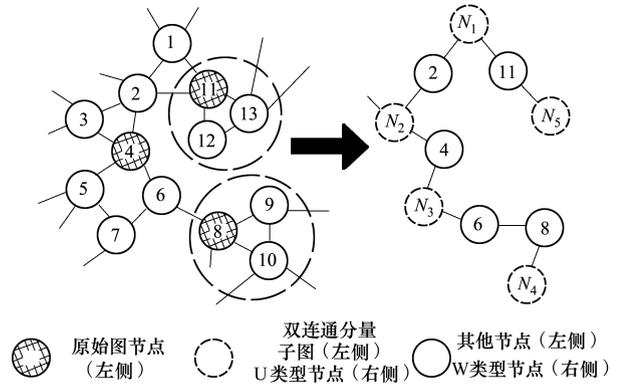


图 1 算法模型图

时去掉与该顶点相关联的所有边后,该图的连通分量数增加,则称该顶点为图 G 的割点。

**定义 4** 双连通分量的界点:若某连通分量中节点和外部节点存在边,称该节点为双连通分量的界点。

割点的示例见图 1 中左侧黑色节点,其中节点 8 和 11 也是界点。双连通分量和割点存在如下关系:

**引理 1** 连通图 G 的某个双连通分量所有界点必然是割点。

**证明** 设双连通分量为  $v_U$ , 则有:

1) 若只有一个界点  $a$ , 则去除  $a$  后,  $v_U$  和外部不相连,  $a$  即为割点。

2) 假设有  $\geq 2$  个界点, 且界点  $b$  不是割点。设界点  $b$  和外部节点  $b_2$  相连, 则对于  $v_U$  中任何一个节点  $a$ , 存在路径  $b_2 \rightarrow b \rightarrow a$ ; 因  $b$  不是割点, 则去除  $b$  后,  $b_2$  和节点  $a$  间仍存在路径, 设该路径上最早访问到的  $v_U$  内节点是界点  $c$  且有  $c \neq b$ , 则节点  $b_2$  到  $c$  存在  $b_2 \rightarrow b \rightarrow c$  和  $b_2 \rightarrow c$  2 条路径; 因路径  $b \rightarrow c$  存在只通过  $v_U$  内节点的路径, 而  $b_2 \rightarrow c$  不通过任何  $v_U$  内节点, 即  $b_2$  到  $c$  存在 2 条路径无重复节点的路径。

3) 因为  $c$  和  $a$  均为  $v_U$  内节点, 存在 2 条无重复节点路径, 进而  $b_2$  和内部任何节点  $a$ , 都存在  $b_2 \rightarrow b \rightarrow c \rightarrow a$  和  $b_2 \rightarrow c \rightarrow a$  2 条无重复节点路径, 即包含  $v_U$  和  $b_2$  的分量也是双连通分量, 这和  $v_U$  是双连通极大子图相矛盾。

4) 由于上述证明并未对界点  $b$  外的任何其他节点的性质做出假设, 即证明任何一个界点如是割点都会导出矛盾, 可得出在存在  $\geq 2$  个界点情况下, 每个界点都必须是割点。

5) 根据前述证明, 对于存在  $\geq 1$  个界点情况

下,每个界点必然是割点,证毕。

由引理 1 可知,在一个连通图中,所有双连通分量必然通过割点与外部相连。

**引理 2** 若在原图  $G$  的一个连通子图中,每个界点在分量外的邻居节点只能通过该界点连到连通分量内节点,且该连通子图中没有割点,则该连通子图为双连通分量。

**证明** 取原始图  $G$  中任意 2 个节点及之间的边构成一个小的特殊的双连通图  $v_U$ ,对  $v_U$  中节点  $a$  在分量外的邻居  $b$ ,只要  $b$  不通过  $a$  也能连到  $v_U$  内节点,即  $b$  能通过第二个界点  $a_2$  连到  $v_U$  内,则  $b$  至少能连到 2 个不同的界点,可知对  $v_U$  内任意节点  $c$ ,节点  $b$  都有 2 条无重复节点路径  $b \rightarrow a \rightarrow c$  和  $b \rightarrow a_2 \rightarrow c$ ,因此  $b$  也能加入到  $v_U$  中并组成更大的双连通图。因此,直到每一个界点在双连通图外邻居只能通过该界点连到连通图内,这种扩张过程才能停住,此时获得的是极大双连通图,即双连通分量。证毕。

基于引理 2,可先找到所有割点,识别被割点分隔的并以割点为界点的连通分量,保证分量外邻居节点只能通过界点连到分量内,即得到双连通分量,在获取所有双连通分量过程中,可构建双连通分量覆盖图,定义如下:

**定义 5** 双连通分量覆盖图:设原始图  $G$  对应的双连通分量覆盖图为  $G'$ ,对原始图  $G$  中每个双连通分量,在  $G'$  中对应一个类型为  $U$  的节点;其每个割点,在  $G'$  中对应一个类型为  $W$  的节点  $v_W$ ,若割点  $v$  属于某个双连通分量,则在该双连通分量在图  $G'$  中的  $U$  类节点和  $v_W$  间建立一条边;对原图中与割点  $v$  不属于同一个双连通分量的所有邻居节点,用他们对应的  $G'$  中节点分别和  $v$  建立一条边。

图 1 中右侧即为双连通覆盖图。关于双连通分量覆盖图  $G'$  的拓扑结构,给出下述定理:

**定理 3** 双连通分量覆盖图  $G'$  是树形拓扑。

**证明** 反证。因不存在环路的图即为树,假设双连通分量覆盖图  $G'$  中存在环路,依据环路定义,环路中任意 2 个节点在  $G'$  中存在无重复点的  $\geq 2$  条路径,即  $G'$  中环路上的类型为  $W$  的节点对应的原图  $G$  中任意两个节点也存在无重复点的  $\geq 2$  条路径,即  $G'$  中环路上的点所对应的点在原图  $G$  中属于同一个双连通分量,而根据生成规则,同一个双连通分量中所有非割点对应一个  $U$  类节点,每个割点对应的  $W$  节点和该  $U$  类节点间存在一条边,同属于一个  $U$  类节点的这些  $W$  节点之间不会存在边,即

$G'$  中这些节点只会组成局部星形结构,这和组成环路相矛盾。证毕。

基于定理 3,我们也把双连通分量覆盖图  $G'$  写为双连通分量覆盖树  $T'$ 。

## 2 算 法

算法的基本思想如图 2 所示:①首先,识别原始图中的双连通分量并缩成点以构建覆盖图,建立原始图到覆盖图中节点的单射关系。②对原始图中节点对  $s$  和  $t$ ,寻找覆盖图中的对应点  $S$  和  $T$ 。③覆盖图中  $S$  和  $T$  之间路径上每个  $U$  类节点对应原图中的一个双连通分量, $U$  类节点在路径上前后两个节点是双连通分量的界点,则可并行求解每个双连通分量的 2 个界点之间的最大流。④合并子问题结果得到原始图中节点对  $s$  和  $t$  最大流的解。

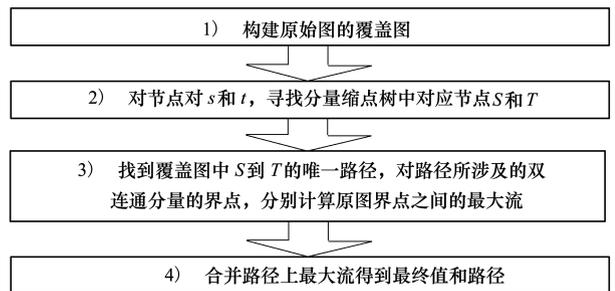


图 2 最大流加速方法整体流程

### 2.1 覆盖图构建算法

覆盖图构建对应图 2 的步骤 (1),原理如图 3 所示。

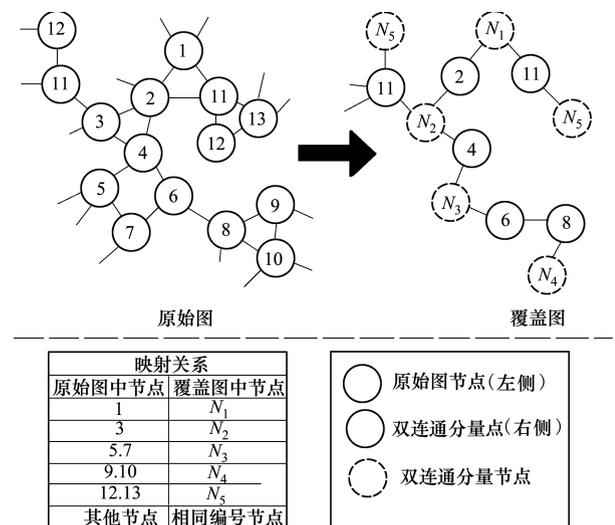


图 3 双连通分量覆盖图构建示意图

左侧原始图可转换为右侧覆盖图,覆盖图中的实线节点对应原图中相同编号的割点,覆盖图中虚线节点对应左侧原图中的双连通分量;其中,原始图到覆盖图的映射关系在图中下方的表格中。对于原图中节点 1 和 10 的最大流,由于对应的  $N_1$  和  $N_4$  之间的唯一路径包含界点 2/4/6/8,则原图中节点 1 和节点 10 最大流可分解为(1,2),(2,4)(4,8)(8,10)4 个并行的最大流计算,从而大幅降低求解时间。注意此时各个界点间的计算仅涉及对应的双连通分量内的节点,即此时原图中节点 1 和 10 之间的所有最大流路径,不可能包含覆盖图中路径上之外的节点,例如此时  $N_2$  的另外一个界点为 11,且根据定义必为割点,则最大流路径不可能涉及节点 12 所在的双连通分量除 11 之外的节点,否则节点 11 必然在路径中出现 2 次,违反了最大流路径为简单路径的定义。这也意味着,基于覆盖图可排除部分节点,这也可降低计算量。

根据引理 2,我们给出了递归的覆盖图构建算法,如表 1 所示。

表 1 双连通分量覆盖图构建算法

input: 当前节点 $node_i$ , 节点深度 $depth_i$ , 邻居节点信息
output: 设置节点 low 值 $low_i$ , 更新的双连通分量覆盖树 $T'$
step1: 初始化当前节点 low 值 $low_i = INF$ , 状态值 $state_i = visiting$ , 初始化双连通分量覆盖树 $T'$ , 初始化全局堆栈 $stack_o$
step2: For $node_i$ 的每个邻居节点 $node_k$
step2.1: if $node_k$ 是 $node_i$ 在遍历过程中的父节点, continue
step2.2: if $state_k = unvisited$
step2.2.1: 设置 $node_k$ 的深度 $depth_k = depth_i + 1$
step2.2.2: 将边 $(node_i, node_k)$ 压到堆栈 $stack_o$ 中
step2.2.3: 以 $node_k$ 为参数调用当前过程 $proc\_shink$
step2.2.4: $temp_k = low_k$
step2.3: elseif $state_k = traversing$ , 设置临时变量 $temp_k = depth_k$
step2.4: else 设置临时变量 $temp_k = low_k$
step2.5: If $low_i > temp_k$ , $low_i = temp_k$
step2.6: If $state_k = visited$ and $low_k > = depth_i$
step2.6.1: if $node_i$ 的割点标识 is $Cut_i = false$
step2.6.1.1: 设置当前节点 $node_i$ 的割点标识 is $Cut_i = true$
step2.6.1.2: 在 $T'$ 中添加原图割点对应的 W 类节点, 记录在 $T'$ 中的 id 为 $node_w$
step2.6.2: 在 $T'$ 中添加原图双连通分量对应的 U 类节点, 记录在 $T'$ 中的 id 为 $node_u$

续表 1

step2.6.3: 从 $stack_o$ 中循环弹出边, 直到遇到边 $(node_i, node_k)$
step2.6.4: 标记包括 $(node_i, node_k)$ 在内的所有已弹出边中非割点对应节点 $node_u$
step2.6.5: 在 $T'$ 中节点 $node_u$ 和 $node_w$ 间建立一条边

整个构建过程首先确定一个根节点,按照深度优先的顺序在每个节点上递归调用固定的算法,在单个节点上调用的递归算法。算法识别各个双连通分量,同时更新对应的双连通分量覆盖树  $T'$ ,并建立双连通分量覆盖树中节点到原图节点的映射关系。算法 1 在运行时,表 1 中递归调用结束时 step1 会被调用  $N$  次,step2 会被调用  $2M$  次,而 step2 中包含循环的步骤为 step2.6.3 和 step2.6.4,由于对每条边只会调用 1 次,全局共被调用  $M$  次,则算法 1 的复杂度为  $O(N + M)$ ,在稀疏大图中远小于最大流算法复杂度下限  $O(NM)$ <sup>[1]</sup>。这说明构建算法复杂度较低,和图的规模呈线性关系。

2.2 基于覆盖图的最大流加速方法

基于覆盖图的加速方法包含图 2 流程图的所有步骤,其示例如图 4 所示。

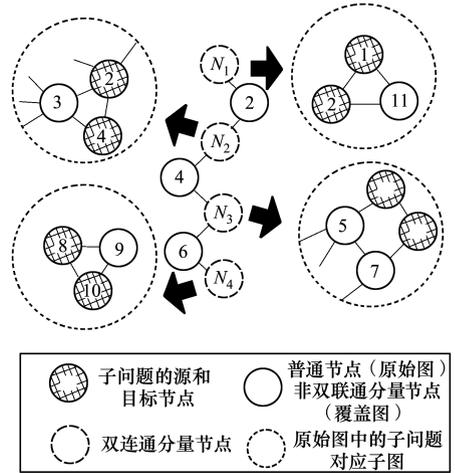


图 4 基于双连通分量覆盖图的最大流加速示意图

针对图 3 中原始图中节点 1 到节点 10 的最大流问题,基于映射表查询得到图 4 的覆盖图中节点  $N_1$  和  $N_4$ 。在双连通分量缩点树  $T'$  中搜索并发现如图 4 中  $N_1$  至  $N_4$  的唯一路径;路径上每个双连通分量节点对应一个子问题(如图 4 中左右两侧的虚线圆圈),通过并行求解子问题可进一步提高计算速度。

对应的最大流加速算法 2 如表 2 所示。

表 2 基于双连通分量覆盖图的最大流加速算法

input:	给定原始图中的节点 $s, t$
output:	最大流值和对应的多路径
step1:	If 原始图未构建双连通分量覆盖图
step1.1	选择原始图中某个节点为根节点
step1.2	在根节点调用算法 1, 递归构建双连通分量覆盖树
step2:	查询 $s$ 和 $t$ 在双连通分量覆盖树中的对应节点 $V_s$ 和 $V_t$
step3:	If $V_s$ 和 $V_t$ 不是双连通分量覆盖数中同一个节点
step3.1	搜索 $V_s$ 和 $V_t$ 之间的唯一路径 $(V_s, \dots, W_1, W_2, \dots, W_L, \dots, V_t)$ , 其中 $W_1, W_2, \dots, W_L$ 是路径上的 $W$ 类节点
step3.2	分别在对应的双连通分量内并行计算 $(s, n_1^w)$ , $(n_1^w, n_2^w), \dots, (n_L^w, t)$ 多个节点对之间的最大流, 其中路径上 $W$ 类节点 $W_k (1 \leq k \leq L)$ 对应的原图割点为 $n_k^w$ , 直到所有节点对计算结束
step3.3	取所有最大流值的最小值作为全局最大流值, 每个最大流路径连接起来作为结果路径,
step3.4	对每个节点对之间最大流路径上所有边的流量值, 按对应最大流和全局最大流最小值的比值等比降低.
step4:	elseif $V_s$ 和 $V_t$ 是双连通分量覆盖数中同一个节点
step4.1	直接在对应的双连通分量中计算 $(s, t)$ 之间的最大流

首先会根据需要构建覆盖图, 之后在双连通分量覆盖树  $T'$  中找到源和目标对应的节点, 获取  $T'$  中的唯一路径, 将路径拆分为若干子路径, 同时计算并汇总得到最终结果, 其中 step1 为覆盖图构建过程, 可单独调用或在第一次计算时调用, 对给定图只调用一次; step2~4 是针对单个源和目标节点对的计算步骤。算法 2 中 step1 即调用算法 1, 其复杂度如上分析为  $O(N + M)$ ; 通过在算法 1 中建立 Hash 表存储映射关系, step2 复杂度可低至  $O(1)$ , 相对其他步骤可忽略不计; step3.1 是一个树中的搜索过程, 在最坏情况下 (树是一个线性序列) 复杂度也仅为  $O(N)$ ; step3.2 的复杂度取决于所有子问题中复杂度最高的那个子问题, 设为  $O(\hat{N}\hat{M})$ , 其中  $\hat{N}$  和  $\hat{M}$  是该子问题的节点和边数量; step3.3 和 3.4 整体的复杂度等于最大流路径涉及的边的数量, 必小于等于  $O(M)$ 。则算法 2 整体的复杂度下限为  $O(N + M + \hat{N}\hat{M})$ 。由于无法直接简单计算 step3.2 复杂度, 我们也可考察间接的影响因素。设单次计算中覆盖图路径上所有节点对应的原图中所有双连通分量的节点

总数量为  $N'$ , 定义缩减倍数为  $N/N'$ , 则缩减倍数决定了算法由于排除不相关节点引入的加速效果, 子问题数量决定了在充分并行情况下的算法并行度, 上述 2 个指标均可间接反映算法加速效果。

### 3 实验

我们在基准评测图中分析和比较算法的加速效果, 每个图的参数主要包括节点数量  $N$  和无向边的数量  $M$ 。对每个  $(N, M)$  组合, 采用权威评测工具 GENRMF 生成 50 个基准图, 每个图选取 50 个节点对分别计算最大流。GENRMF 工具接收  $a, b, c_1$  和  $c_2 (> c_1)$  4 个参数, 首先会产生  $b$  个拥有  $a$  个节点的子图, 随机分配顺序号给每个子图中节点, 并将其与前后 2 个相邻编号的节点相连; 每个子图也随机获得一个顺序号, 与前后 2 个相邻编号的子图做节点一对一的映射并建立连接。每个图中, 边的容量在  $c_1$  和  $c_2$  之间随机分布, 节点数量  $N = ab$ , 无向边数量  $M = a(2b - 1)$ , 在此基础上随机增加边以得到对应的边数量  $M$ 。因  $M \geq 6N$  后并行度变化不大, 实验主要考察  $M \leq 5N$  的稀疏图中的变化情况。

首先考察多个图中多次计算时并行度的均值及最大和最小值变化趋势, 如图 5a) 所示, 其中稀疏度 (density) 定义为  $M/N$ 。图 5a) 表明并行度 (及最大/最小值) 均随节点数量  $N$  的增加或稀疏度的降低有缓慢增加, 不同稀疏度下的并行度均值随着  $N$  的增加逐渐趋向接近, 说明算法在稀疏大图中始终能保持合适的并行度。随着节点数量的增大, 并行度的区间有增大趋势, 稀疏度较高时区间范围随节点数量增大增加趋势更快, 例如在  $N = 10^7$  时所有稀疏度下的最小值也能达到接近 10 左右, 保证最差情况下的加速效果。

我们也考察缩减倍数均值及最大最小值的变化趋势, 如图 5b) 所示, 其中  $Y$  轴为指数坐标系。可发现, 缩减倍数均值和节点数量大致呈指数关系, 且均值 (及最大/最小值) 随节点数量  $N$  的增加或稀疏度的降低有缓慢的增加。随着节点数量的增加, 各种稀疏度下的缩减倍数均有指数级的增加, 并不随着稀疏度变化有明显降低。且随着节点数量的增大, 区间也有明显的变大趋势 (注:  $Y$  轴为指数坐标系)。另外缩减倍数最小值始终都有增大的趋势, 但是在稀疏度较高和节点数量较低时可能会很小, 如  $N = 10^5$  且稀疏度为 5 时为 1 左右, 此时整个原图即为

一个双连通分量,从而使得最大流计算需涉及原图中所有节点。

法<sup>[22]</sup>,因此在各类图缩减算法计算时均默认采用 highest push/reliable 计算缩减图中的最大流,比对的各类算法标记如下:

Clique:文献[10]基于局部子图缩减的最大流求解方法,检测子图特征时使用了最大团方法。

Community 方法:文献[10]中基于网络社区结构缩减的最大流求解方法。

VertexPair:文献[15]中基于节点对的缩减方法。

实验中针对 3 个不同的节点数量  $N$ ,从基准图中随机选择各种稠密度的 50 幅图,每幅基准图中选取 50 个不同节点对分别进行最大流计算,并记录各种不同方法中除去数据载入和输出的计算时间,如图 6 所示,其中  $X$  轴分别对应节点数量  $N=10^5, 10^6, 10^7$ , $Y$  轴为按对数坐标显示的计算时间。此时我们算法 BCR 对应  $X$  轴 3 个坐标点的计算时间均值分别为 0.3 s, 2.3 s, 13.2 s。可见仅  $N=10^5$  时 VertexPair 方法平均要稍快一些(约为我们方法的 0.9 倍左右时间),其他方法以及在更大的图中,我们方法都是最快的,而且随着图规模的增加,其他方法的计算时间呈指数级上升。检查数据发现,由于 Clique 和 Community 方法寻找最大团和社团等无尺寸限制的子图结构,而实际上由于算法限制,稀疏图中能找到的子图结构并不多,从而影响了加速效果;随着图规模的增加,相应的子图结构并没有同步变大,使得缩减效果进一步降低,导致加速效果变差,而 VertexPair 基于局部的 2 个子节点进行缩减,无论在何种图中都能充分找到可利用的子图结构,从而保证一定的加速效果。

我们也比较了所有方法的预处理时间,如图 7 所示。我们算法中对应  $X$  轴 3 个坐标点的预处理时间均值分别为 0.9 s, 8.7 s, 61.3 s。我们算法始终保持较低的预处理时间;且随着图规模的增大,在  $N=10^7$  时相对其他方法的优势更加明显;检查对应的预处理算法发现, Clique 和 Community 方法寻找无尺寸限制子图方法,在原图增大时计算复杂度呈指数级增加;而 VertexPair 方法优化后可维持线性的复杂度,但整体上我们的预处理时间仍然最低。前述实验可得到如下结论:

1) 相对其他方法,我们算法加速效果部分来自于排除作用,即排除计算不会涉及的节点,将计算相关节点缩减到一个很小的范围,在大图中甚至可缩减到原图节点的万分之一左右,大大加速计算速度;

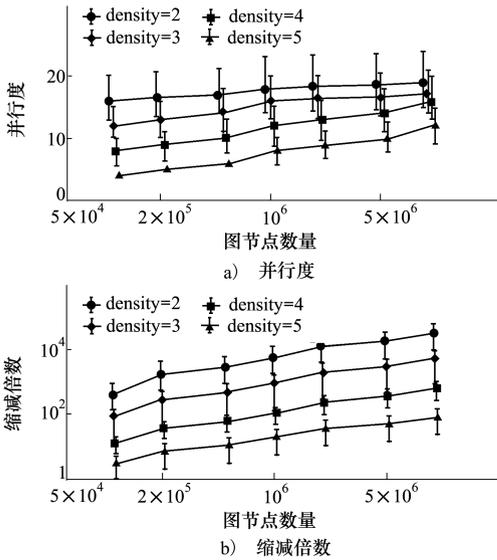


图 5 基准图中并行度和缩减倍数的变化趋势

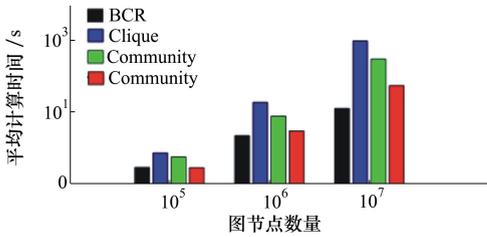


图 6 基准图中所有算法平均最大流计算时间对比

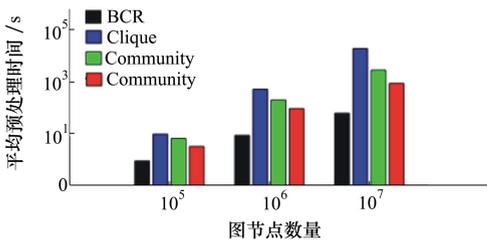


图 7 基准图中所有算法平均预处理时间对比

基于前述理论分析,我们在上述基准网络上比较我们的算法和经典算法的效果。硬件环境为 ThinkStation P310 工作站,拥有 2 颗 4 核 CPU 和 32GB 内存,CPU 主频 2.6GHz,操作系统采用 CentOS7.0。并行计算时开启 8 个线程,每线程对应一个 CPU 核;计算时将所有子问题组织成先进先出队列,依次将问题调度给空闲的线程。已知面向通用最大流求解的串行算法是基于 highest push/reliable 的方

加速效果的另外一个来源是并行计算的引入,通过将原始的最大流问题分隔成多个独立的可并行求解的子问题,可充分利用底层设施的并行性加速计算;且由于独立子问题数量不会太多,对基础设施要求也较低。

2) 相比基于大范围子图方法 Clique 和 Community, 我们的算法加速效果更明显,且前期处理时间也大幅领先;我们的算法对于 VertexPair 这种针对局部小型子图的方法也有优势。上述优势随着图尺寸的增加而增加,也说明了我们方法更加适用于大规模稀疏图。

## 4 结 论

本文提出基于双连通分量覆盖图的稀疏大图最大流并行加速方法,算法采用双连通分量建立覆盖图,将原始图中最大流问题分解为多个子图中的局部问题,基于并行计算加速求解。在大量基准图上的分析和实验表明,算法具极低的覆盖图构建代价,相比已知算法可达到2个数量级的加速效果。将来的工作可从多个角度展开,包括寻找稠密大图中的最大流并行加速方法,及如何在分布式计算框架上实现基于覆盖图的加速计算。

## 参考文献:

- [1] Goldberg A, Tarjan E. Efficient Maximum Flow Algorithms[J]. Communications of the ACM, 2014, 57(8):82-89
- [2] Liang C, Yu F R, Zhang X. Information-Centric Network Function Virtualization over 5g Mobile Wireless Networks[J]. IEEE Network, 2015, 29(3): 68-74
- [3] Kosut O. Max-Flow Min-Cut for Power System Security Index Computation[C]//IEEE Sensor Array and Multichannel Signal Processing Workshop, 2015:61-64
- [4] 陈晓旭, 吴恒, 吴悦文. 基于最小费用最大流的大规模资源调度方法[J]. 软件学报, 2017, 28(3): 598-610  
Chen Xiaoxu, Wu Heng, Wu Yuewen. Large-Scale Resource Scheduling Based on Minimum Cost Maximum Flow[J]. Journal of Software, 2017, 28(3): 598-610 (in Chinese)
- [5] Ford L R, Fulkerson D R. Notes on Linear Programming, Part II: Maximal Flow through a Network[M]. Santa Monica, RAND Corporation, 1977:16-31
- [6] Karzanov A V. Determining the Maximal Flow in a Network by the Method of Preflows[J]. Doklady Mathematics, 1974, 15(1): 434-437
- [7] Schiopu C, Ciurea E. The Maximum Flows in Planar Dynamic Networks[J]. International Journal of Computers Communications & Control, 2016, 11(2): 282-291
- [8] Goldberg AV, Hed S, Kaplan H, Tarjan RE, Werneck RF. Maximum Flows by Incremental Breadth-First Search[C]//European Symposium on Algorithms, 2011:457-468
- [9] Ghaffari M, Karrenbauer A, Kuhn F. Near-Optimal Distributed Maximum Flow: Extended Abstract[C]//ACM Symposium on Principles of Distributed Computing, 2015:81-90
- [10] Sherman J. Nearly Maximum Flows in Nearly Linear Time[C]//IEEE Symposium on Foundations of Computer Science, 2013: 263-269
- [11] Liers F, Pardella G. Simplifying Maximum Flow Computations: The Effect of Shrinking and Good Initial Flows[J]. Discrete Applied Mathematics, 2011, 159(17): 2187-2203
- [12] Dan G. Very Simple Methods for All Pairs Network Flow Analysis[J]. Siam Journal on Computing, 1990, 19(1): 143-155
- [13] Zhang Y, Xu X, Hua B. Contracting Community for Computing Maximum Flow[C]//IEEE International Conference on Granular Computing, 2012:651-656
- [14] Zhang Y P, Hua B, Jiang J, et al. Research on the Maximum Flow in Large-Scale Network[C]//International Conference on Computational Intelligence and Security(CIS), 2011:482-486
- [15] Scheuermann B, Rosenhahn B. Slimcuts: Graphcuts for High Resolution Images Using Graph Reduction[C]//International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, 2011: 219-232
- [16] Zhen C, Zhang L. The Computation of Maximum Flow in Network Analysis Based on Quotient Space Theory[J]. Chinese Journal of Computers, 2015, 38(8):1705-1712

- [17] Niklas B, Blueloch G, Shun J. Efficient Implementation of a Synchronous Parallel Push-Relabel Algorithm[C]//European Symposium on Algorithms, 2015:106-117
- [18] Soner S, Ozturan C. Experiences with Parallel Multi-Threaded Network Maximum Flow Algorithm[J]. Partnership for Advanced Computing in Europe, 2013, 2013(1):1-10
- [19] Benoit D, Dupont E, Zhang W. Distributed Max-Flow in Spark[EB/OL]. (2015-06-03)[2017-04-06]. [http://stanford.edu/~rezab/classes/cme323/S15/projects/distributed\\_max\\_flow\\_report.pdf](http://stanford.edu/~rezab/classes/cme323/S15/projects/distributed_max_flow_report.pdf)
- [20] Halim F, Yap R, Wu Y. A Mapreduce-Based Maximum-Flow Algorithm for Large Small-World Network Graphs[C]//International Conference on Distributed Computing Systems(ICDCS), 2011:192-202
- [21] Goldfarb D, Grigoriadis M D. A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow[J]. Annals of Operations Research, 1988, 13(1): 81-123
- [22] Goldberg, A V. Two-Level Push-Relabel Algorithm for the Maximum Flow Problem[C]//The Fifth Conference on Algorithmic Aspects in Information Management, 2009:212-225

## Engineering Bi-Connected Component Overlay for Maximum-Flow Parallel Acceleration in Large Sparse Graph

Liu Yang, Wei Wei, Xu Heyang

(College of Information Science and Engineering, Henan University of Technology, Zhengzhou 450001, China)

**Abstract:** Network maximum flow problem is important and basic in graph theory, and one of its research directions is maximum-flow acceleration in large-scale graph. Existing acceleration strategy includes graph contraction and parallel computation, where there is still room for improvement: (1) The existing two acceleration strategies are not fully integrated, leading to their limited acceleration effect; (2) There is no sufficient support for computing multiple maximum-flow in one graph, leading to a lot of redundant computation. (3) The existing preprocessing methods need to consider node degrees and capacity constraints, resulting in high computational complexity. To address above problems, we identify the bi-connected components in a given graph and build an overlay, which can help split the maximum-flow problem into several subproblems and then solve them in parallel. The algorithm only uses the connectivity in the graph and has low complexity. The analyses and experiments on benchmark graphs indicate that the method can significantly shorten the calculation time in large sparse graphs.

**Keywords:** computational complexity; graph theory; maximum flow problem; sparse graph computing; bi-connected component; overlay; parallel computing