最大流

王曦 2021192010

数学与统计学院

算法设计与分析

2023 年 06 月 13 日

1. 实验内容

实验内容

1. 有 m 篇论文和 n 个评审, 每篇论文需要安排 a 个评审, 每个评审最多评 b 篇论文. 请设计一个论文分配方案.
2. 要求应用最大流解决上述问题, 画出 $m = 10, n = 3$ 的流网络图并解释说明流网络图与论文评审问题的关系.
3. 编程实现所设计算法, 计算 a 和 b 取不同值情况下的分配方案, 如果没有可行方案则输出无解.

2. 实验环境与约定

实验环境与约定

- 实验环境: GNU C++17 (O2).
- 约定所有实际运行时间都不包含数据生成和输入的时间.

3. 构建流网络

构建流网络

- 为使得流网络结构更清晰, 采用分层的形式构建, 省略了一些重复的节点, 如下图所示:

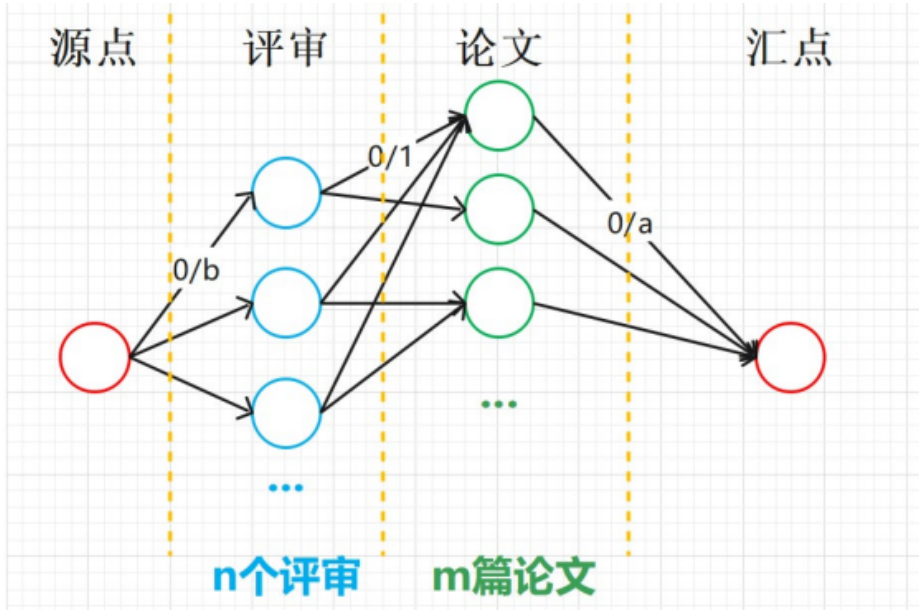


Figure 1: 流网络示意图

- 上图所示的流网络分为 4 层, 第 1 层只有源点, 第 4 层只有汇点, 第 2 层为 n 个评审, 第 3 层为 m 篇论文.

构建流网络

(1) 边的含义:

(i) 源点 $\rightarrow$ 评审的边的容量为每个评审最多评的论文数, 即 b.

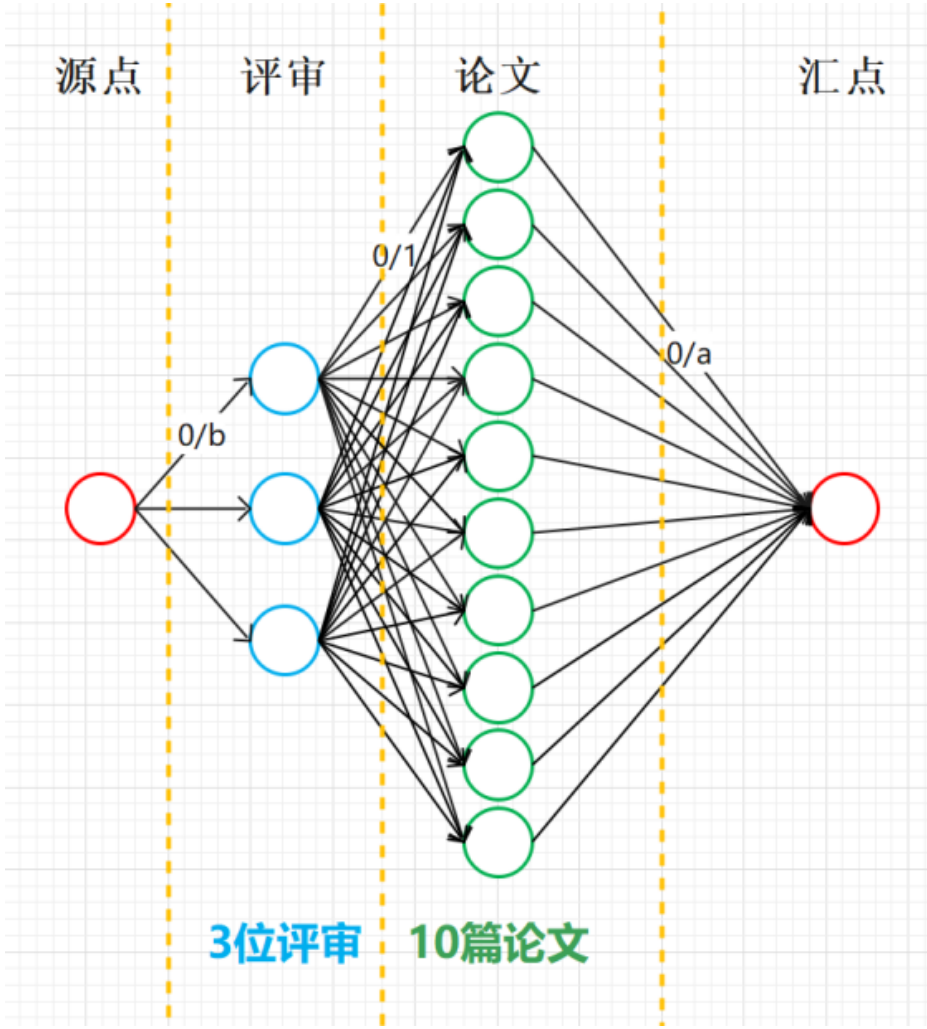
(ii) 评审 $\rightarrow$ 论文的边的容量为 1, 因一个评审对一篇论文, 要么未评阅 (流量为 0), 要么以评阅 (流量为 1).

(iii) 论文 $\rightarrow$ 汇点的边的容量为每篇论文需安排的评审数, 即 a. 若论文你节点到汇点的流量为 f, 则该论文被 f 个评审评过了.

(2) 第 1 层到第 2 层的边的容量保证了每位评审的最多评阅的论文数, 第 2 层到第 3 层的边表示每位评审是否评阅每篇论文, 第 3 层到第 4 层的容量保证了每篇论文所需的评审数.

构建流网络

- $n = 3, m = 10$, 即 3 位评审、10 篇论文时, 流网络如下图所示:



构建流网络

- 流网络的最大流与论文评审问题的解的关系

(1) 当最大流不等于论文数与每篇论文所需的评审数之积 $(m \times a)$ 时, 论文评审问题无解, 因为存在论文未被 a 位评审评阅; 否则论文评审问题有解.

(2) 有解时, 若评审 A 指向论文 B 的边有流量, 则评审 A 评了论文 B, 否则未评论文 B.

(3) 有解时, 源点指向评审的边满流, 论文指向汇点的边满流, 此时流量为最大流.

4. Ford-Fulkerson 增广

Ford-Fulkerson 增广

- Ford-Fulkerson 增广是求最大流的一类方法, 用贪心的思想, 通过寻找增广路更新流量.
- (见实验报告) 定义: 剩余容量、残量网络、增广路、增广.
- 退流的 “抵消” 效果使得无需关心选择增广路的顺序.
- 若残量网络 G_f 中存在增广路, 则可对其增广使得流量增加; 否则当前流量已是最大流.
- 在整数流量的网络 $G = (V, E)$ 上, Ford-Fulkerson 增广的时间复杂度的一个上界是 $O(|E| \cdot |f|)$, 其中 f 是 G 上的最大流. 这是因为每轮增广的时间复杂度为 $O(|E|)$, 增广会导致总流量增加, 则增广轮数不超过 $|f|$.
- Ford-Fulkerson 增广有不同的实现, 时间复杂度各不相同, 其中主流实现有 Edmonds-Karp 算法、Dinic 算法、SAP 算法、ISAP 算法等.

5. Edmonds-Karp 算法

Edmonds-Karp 算法

- Edmonds-Karp 算法用 BFS 求增广路, 过程如下:

(1) 若残量网络 G_f 中可从源点 s 出发 BFS 到汇点 t, 则已求得增广路.

(2) 对增广路 p, 求其所经过的边的剩余容量的最小值 $\Delta = \min_{(u,v)\in p} c_f(u,v)$, 给 p 的每条正向边都加上 Δ 的流量, 给每条反向边都退掉 Δ 的流量, 此时最大流增大 Δ.

(3) 修改流量后得到新的 G_f, 重复上述过程直至找不到增广路.

Edmonds-Karp 算法

- 单次 BFS 的时间复杂度为 $O(|E|)$, 增广轮数的上界为 $O(|V| \cdot |E|)$, 故 Edmonds-Karp 算法的总时间复杂度为 $O(|V| \cdot |E|^2)$.
- 证明增广轮数的上界时用到了如下引理, 详细的证明见实验报告:

[**最短路非递减引理**] 设 $d_f(u)$ 为残量网络 G_f 中源点 s 到节点 u 的最短路的长度. 对某轮增广, 设增广前、后的流分别为 f 和 f', 则对任意节点 u, 都有 $d_{f'}(u) \geq d_f(u)$.

6. Dinic 算法

Dinic 算法

- 增广前先对残量网络 G_f 做 BFS 分层, 即根据节点 u 与源点 s 的距离 $d(u)$ 将图分为若干层.
- (见实验报告) 定义: 层次图、阻塞流.
- Dinic 算法的过程:

(1) 在 G_f 上 BFS 出层次图 G_L.
(2) 在 G_L 上 DFS 出阻塞流 f_b.
(3) 将 f_b 并入原来的流 f 中.
(4) 重复上述过程直至不存在源点 s 到汇点 t 的路径, 此时的 f 为最大流.

Dinic 算法

- 当前弧优化:

(1) 对 G_L 做 DFS 时, 若节点 u 同时有大量的入边和出边, 且 u 每次接受来自入边的流量时都需遍历出边表以决定将流量传递给哪条出边, 则局部时间复杂度为 $O(|E|^2)$.

(2) 考虑优化. 若某时刻已知边 (u, v) 已增广至极限, 即已无剩余容量或 v 的后侧已增广至阻塞, 则 u 的流量没必要尝试再流向出边 (u, v). 对每个节点 u, 维护 u 的出边表中第一条还有必要尝试的出边的编号, 称该指针为当前弧, 称该优化为当前弧优化.

Dinic 算法

- 多路增广:

(1) 若在层次图上找到一条从源点 s 到汇点 t 的增广路 p, 则后续未必需重新从 s 出发找下一条增广路, 可从 p 上最后一个仍有剩余容量的位置出发寻找一条岔路进行增广, 该优化称为多路增广, 可在 DFS 回溯时实现.

(2) 当前弧优化是保证 Dinic 算法的时间复杂度的一部分, 但多路增广只减小常数.

- 单轮增广中 DFS 求阻塞流的时间复杂度为 $O(|V| \cdot |E|)$ (证明见实验报告). 因层次图的层数不超过 $|V|$, 论文 (https://people.orie.cornell.edu/dpw/orie633/LectureNotes/lecture9.pdf) 中证明了层次图的层数在增广过程中严格单增, 则 Dinic 算法的增广轮数为 $O(|V|)$, 进而总时间复杂度为 $O(|V|^2 \cdot |E|)$.

7. MPM 算法

MPM 算法

- MPM 算法求最大流有如下两种形式:

  (1) 用基于堆的优先队列, 时间复杂度为 $O(|V|^3 \log |V|)$.
  (2) 用 BFS, 时间复杂度为 $O(|V|^3)$, 下面介绍该算法.

- MPM 算法的整体结构类似于 Dinic 算法, 也是分阶段运行. 每个阶段, 在流网络 G 的残量网络的分层网络中求增广路. MPM 算法求增广路的方式与 Dinic 算法不同, MPM 算法用 $O(|V|^2)$ 的时间复杂度求增广路.
- (见实验报告) 定义: 节点的容量、参考节点.
- MPM 算法用 BFS 求增广路, 增广后满流的边可从 L 中删除, 因为它们不会在后续阶段被使用. 同理可删除除源点 s 和汇点 t 外的无出边和无入边的节点.

MPM 算法

- MPM 算法至多有 $|V|$ 轮迭代, 因为每次至少删除了所选的参考节点, 且在每次迭代中都删除了除最多 $|V|$ 以外经过的所有边, 则每个阶段的时间复杂度为 $O(|V|^2 + |E|) = O(|V|^2)$. 因阶段总数 $< |V|$ (下面将证明), 则 MPM 算法的总时间复杂度为 $O(|V|^3)$.
- 证明阶段总数 $< |V|$ 时用到下面两个引理, 详细的证明见实验报告:

[**引理 I**] 每轮迭代后, 源点 s 到其它节点的距离不减, 即对任意阶段 i 和节点 v, 都有 $level_{i+1}[v] \geq level_i[v]$.

[**引理 II**] 对任意阶段 i 和节点 t, 都有 $level_{i+1}[t] > level_i[t]$.

8. ISAP 算法

ISAP 算法

- Dinic 算法每次求完增广路都需跑 BFS 分层, ISAP 算法对此做优化.
- ISAP 算法在原图的反图上跑 BFS 分层, 即从汇点 t 到源点 s 做 BFS. 分层完成后, 用 DFS 求增广路, 增广的过程类似于 Dinic 算法, 只选择比当前节点层数少 1 的节点增广. 与 Dinic 算法不同的是, 不重新跑 BFS 来重分层, 而是在增广的过程中重分层.
- 同 Dinic 算法, ISAP 算法也有当前弧优化.
- GAP 优化: 设层数为 i 的节点数为 num_i. 每当将一个节点的层数从 x 更新到 y 时, 同时更新 $num[]$ 数组. 若更新后 $num_x = 0$, 则图存在断层, 此时不存在增广路, 直接终止算法.

9. Push-Relabel 预流推进算法

Push-Relabel 预流推进算法

- 预流推进算法求最大流时忽略流量守恒, 每次更新一个节点的信息, 直至无节点需更新.
- (见实验报告) 定义: 节点的超额流、节点的高度.
- 算法的正确性基于如下引理:

[**引理**] 设残量网络 G_f 中的高度函数为 h, 则对 $\forall u, v \in V$, 若 $h(u) > h(v) + 1$, 则 $(u, v) \notin G_f$.

- 上述引理表明: 预流推进算法只会在 $h(u) = h(v) + 1$ 的边 (u, v) 上推流.

Push-Relabel 预流推进算法

- 推送 (Push):

(1) 适用条件: 若节点 u 溢出, 且存在节点 v $s.t.$ $(u,v) \in E_f$, 且 $c(u,v) - f(u,v) > 0, h(u) = h(v) + 1$, 则对边 (u,v) 做 push 操作.

(2) 初始化见实验报告.

- 重贴标签 (Relabel):

(1) 适用条件: 若节点 u 溢出, 且对 $\forall (u,v) \in E_f$, 都有 $h(u) \leq h(v)$, 则对 u 用 relabel 操作, 将 $h(u)$ 更新为 $\min\limits_{(u,v)\in E_f} h(v) + 1$ 即可.

(2) 初始化见实验报告.

Push-Relabel 预流推进算法

- Push-Relabel 预流推进算法的过程:

(1) 每次扫描整个图, 只要存在节点 u 满足 push 或 relabel 操作的条件, 就做相应的操作.

(2) 最后超额流的一部分流回 s, 且除源点和汇点外, 其他节点不溢出, 此时流函数 f 满足流守恒性, 为最大流, 流量为 $e(t)$.

(3) 论文 (Cherkassky B V, Goldberg A V. On implementing push-relabel method for the maximum flow problem[C]//International Conference on Integer Programming and Combinatorial Optimization. Springer, Berlin, Heidelberg, 1995: 157-171.) 指出: 只处理高度 $< n$ 的溢出节点也可求得正确的最大流值, 但这样结束时流函数不满足流守恒性, 不能确定每条边上的真实流量.

10. HLPP 算法

HLPP 算法

- 最高标号预流推进算法 (Highest Label Preflow Push) 在预流推进算法的基础上, 每次选择高度最高的溢出节点, 时间复杂度为 $O\left(n^2\sqrt{m}\right)$.
- HLPP 算法的过程:

(1) 初始化: 同预流推进算法.

(2) 选择溢出节点中高度最高的节点 u, 对其所有可推送的边进行推送.

(3) 若 u 仍溢出, 则对其重贴标签, 回到步骤 (2).

(4) 若无溢出的节点, 算法结束.

HLPP 算法

- 论文 (Ahuja R K, Kodialam M, Mishra A K, et al. Computational investigations of maximum flow algorithms[J]. European Journal of Operational Research, 1997, 97(3): 509-542.) 表明: 预流推进算法的时间瓶颈在重贴标签. 论文 (Derigs U, Meier W. Implementing Goldberg's max-flow-algorithm—A computational investigation[J]. Zeitschrift für Operations Research, 1989, 33(6): 383-403.) 介绍了如下两种能显著减小重贴标签次数的优化.

- BFS 优化:

  (1) 初始化 $h(u)$ 为节点 u 到汇点 t 的最短距离. 特别地, $h(s) = n$.
  (2) 在 BFS 的同时检查图的连通性, 排除无解的情况.

HLPP 算法

- GAP 优化:

(1) HLPP 推送的条件为 $h(u) = h(v) + 1$. 若在某时刻 $s.t.\ h(u) = t$ 的节点 u 的个数为 0, 则 $h(u) > t$ 的节点无法推送超额流到汇点 t, 故只能回到源点 s, 此时让它们的高度变为 $\geq n + 1$, 以尽快推送回 s, 减小重贴标签的操作.

(2) 论文 (Ahuja R K, Kodialam M, Mishra A K, et al. Computational investigations of maximum flow algorithms[J]. European Journal of Operational Research, 1997, 97(3): 509-542.) 中用 $(2|E| - 1)$ 个桶 $B[]$, 其中 $B[i]$ 记录当前所有高度为 i 的溢出节点. 加入以上两种优化, 且只处理高度 $< n$ 的溢出节点.

- HLPP 算法较抽象, 有实际运行过程的演示图.

11. 实验结果与分析

有解性

- 固定 $n = 3, m = 10$, 考察不同的 a, b 值时的实际运行效率和有解性.
- 经测试, 用上述算法的实际运行时间都为 0 ms, 有解性如下表所示:

a/b	1	2	3	4	5	6	7	8	9	10
1	F	F	F	**T**	**T**	**T**	**T**	**T**	**T**	**T**
2	F	F	F	F	F	F	**T**	**T**	**T**	**T**
3	F	F	F	F	F	F	F	F	F	**T**

有解性条件

- 论文评审问题有解 iff $n \cdot b \geq m \cdot a$.

  [证] 从源点发出的流量为 $b \cdot n$.
  (1) 论文评审问题有解时, 该网络的最大流 $\geq m \cdot a$,
  则 $n \cdot b \geq m \cdot a$.
  (2) $n \cdot b \geq m \cdot a$ 时, 该网络的最大流可达到 $m \cdot a$.
  取得最大流时, 可用如下方法构造论文评审问题的一组解:
  若评审 A 指向论文 B 的节点的边有流量, 则让 A 评阅 B.

谢　谢!

Thank you!