USTC

Chapter 10
Network Flow

王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn/

学习要点

- 理解网络与网络流的基本概念
- 掌握网络最大流的增广路算法
- 掌握网络最大流的预流推进算法
- 掌握网络最小费用流的消圈算法
- 掌握网络最小费用流的最小费用路算法
- 掌握网络最小费用流的网络单纯形算法

基本概念

基本概念和术语

- **网络**
  - G 是一个简单**有向**图，$G=(V, E)$，$V=\{1, 2, \ldots, n\}$
  - 在 V 中指定一个顶点 s，称为**源**和另一个顶点 t，称为**汇**
  - 有向图 G 的每一条边 $(v,w)\in E$，对应有一个值 $cap(v,w)\geq 0$，称为边的**容量**
  - 这样的有向图 G 称作一个**网络**
- **网络流**
  - 网络上的**流**是定义在网络的边集合 E 上的一个非负函数 $flow=\{flow(v,w)\}$
  - 并称 $flow(v,w)$ 为边 (v,w) 上的**流量**

基本概念和术语

❖ 可行流

- 满足下述条件的流 flow 称为**可行流**：
  - **(3.1)容量约束**：对每一条边 (v,w)∈E，0≤flow(v,w)≤cap(v,w)
  - **(3.2)平衡约束**：
- 对于中间顶点：流出量=流入量。即对每个 v∈V(v≠s,t) 有：顶点 v 的流出量－顶点 v 的流入量=0，即 $\sum_{(v,w)\in E} flow(v,w) - \sum_{(w,v)\in E} flow(w,v) = 0$
- 对于源 s：s 的流出量－s 的流入量=源的净输出量 f，即 $\sum_{(s,v)\in E} flow(s,v) - \sum_{(v,s)\in E} flow(v,s) = f$
- 对于汇 t：t 的流入量－t 的流出量的=汇的净输入量 f，即 $\sum_{(v,t)\in E} flow(v,t) - \sum_{(t,v)\in E} flow(t,v) = f$
  - 式中 f 称为这个可行流的流量，即源的净输出量(或汇的净输入量)
  - 可行流总是存在的，例如，让所有边的流量 flow(v,w)=0，就得到一个其流量 f=0 的可行流(称为 0 流)

基本概念和术语

❖ 边流

- 对于网络 G 的一个给定的可行流 flow，将网络中满足 flow(v,w)=cap(v,w) 的边称为**饱和边**；flow(v,w)<cap(v,w)的边称为**非饱和边**；flow(v,w)=0 的边称为**零流边**；flow(v,w)>0的边称为**非零流边**；当边(v,w)既不是一条零流边也不是一条饱和边时，称为**弱流边**

❖ 最大流

- 最大流问题即求网络 G 的一个可行流 flow，使其流量 f 达到最大，即 flow 满足：
  - $0 \le flow(v,w) \le cap(v,w)$，$(v,w) \in E$；且

$$\sum flow(v,w) - \sum flow(w,v) = \begin{cases} f & v = s \\ 0 & v \ne s,t \\ -f & v = t \end{cases}$$

❖ 流的费用

- 在实际应用中，与网络流有关的问题，不仅涉及流量，而且还有费用的因素。此时网络的每一条边 (v,w) 除了给定容量 cap(v,w) 外，还定义了一个单位流量费用 cost(v,w)
- 对于网络中一个给定的流flow，其费用定义为：

$$cost(flow) = \sum_{(v,w) \in E} cost(v,w) \times flow(v,w)$$

基本概念和术语

❖ 残流网络

- 对于给定的一个流网络 G 及其上的一个流 flow，网络 G 关于流 flow 的残流网络 G* 与 G 有相同的顶点集 V，而网络 G 中的每一条边对应于 G* 中的 1条边或 2 条边
- 设(v,w)是G的一条边
  - 当 flow(v,w)>0 时，（w,v）是 G* 中的一条边，该边的容量为 cap*(w,v)=flow(v,w)
  - 当 flow(v,w)<cap(v,w) 时，(v,w) 是 G* 中的一条边，该边的容量为 cap*(v,w)=cap(v,w)-flow(v,w)
- 按照残流网络的定义，当原网络 G 中的边 (v,w) 是一条零流边时，残流网络 G* 中有唯一的一条边 (v,w) 与之对应，且该边的容量为 cap(v,w)
- 当原网络 G 中的边 (v,w) 是一条饱和边时，残流网络 G* 中有唯一的一条边 (w,v) 与之对应，且该边的容量为 cap(v,w)
- 当原网络 G 中的边 (v,w) 是一条弱流边时，残流网络 G* 中有 2 条边 (v,w) 和 (w,v) 与之对应，这 2 条边的容量分别为 cap(v,w) -flow(v,w) 和 flow(v,w)
- 残流网络是设计与网络流有关算法的重要工具

最大网络流问题

增广路算法

❖ 算法基本思想

- 设 P 是网络 G 中联结源 s 和汇 t 的一条路，定义路的方向是从 s 到 t
- 将路 P 上的边分成 2 类：
  - 一类边的方向与路的方向一致，称为**向前边**，向前边的全体记为P+
  - 另一类边的方向与路的方向相反，称为**向后边**，向后边的全体记为P-
- 设 flow 是一个可行流，P 是从 s 到 t 的一条路，若 P 满足下列条件：

  （1）在 P 的所有向前边 (v,w) 上，flow(v,w)<cap(v,w)，即 P+ 中的每一条边都是非饱和边

  （2）在 P 的所有向后边 (v,w) 上，flow(v,w)>0，即 P- 中的每一条边都是非零流边

  则称 P 为关于可行流 flow 的一条**可增广路**
- 可增广路是残流网络中一条容量大于 0 的路
- 将具有上述特征的路 P 称为可增广路是因为可以通过修正路 P 上所有边流量 flow(v,w) 将当前可行流改进成一个流值更大的可行流

增广路算法

- 增流的具体做法是：
  - 不属于可增广路 P 的边 (v,w) 上的流量保持不变
  - 可增广路 P 上的所有边 (v,w) 上的流量按下述规则变化：
    - 在向前边(v,w)上，flow(v,w)+d
    - 在向后边(v,w)上，flow(v,w)-d
  - 按下面的公式修改当前的流 $$flow(v,w)=\begin{cases} flow(v,w)+d & (v,w)\in P^{+} \\ flow(v,w)-d & (v,w)\in P^{-} \\ flow(v,w) & (v,w)\notin P \end{cases}$$
  - 其中 d 称为可增广量，可按下述原则确定：
    - d 取得尽量大，又要使变化后的流仍为可行流
  - 按照这个原则，d 既不能超过每条向前边 (v,w) 的 cap(v,w)-flow(v,w)，也不能超过每条向后边 (v,w) 的 flow(v,w)
  - 因此 d 应该等于向前边上的 cap(v,w)-flow(v,w) 与向后边上的 flow(v,w) 的最小值，也就是残流网络中 P 的最大容量
- **增广路定理**：设 flow 是网络 G 的一个可行流，如果不存在从 s 到 t 关于 flow 的可增广路 P，则 flow 是 G 的一个最大流

增广路算法

算法描述

- 最大流的增广路算法如右
- 该算法也常称作 Ford Fulkerson 算法

```
template <class Graph, class Edge> class MAXFLOW
{
    const Graph &G;
    int s, t,maxf;
    vector<int> wt;
    vector<Edge *> st;
    int ST(int v) const  { return st[v]->other(v); }
    void augment(int s, int t)
    {
        int d = st[t]->capRto(t);
        for (int v = ST(t); v != s; v = ST(v))
            if (st[v]->capRto(v) < d)  d = st[v]->capRto(v);
        st[t]->addflowRto(t, d);
        maxf+=d;
        for ( v = ST(t); v != s; v = ST(v))  st[v]->addflowRto(v, d);
    }
    bool pfs();
public:
     MAXFLOW(const Graph &G, int s, int t,int &maxflow) :
        G(G), s(s), t(t), st(G.V()), wt(G.V()),maxf(0)
     { while (pfs()) augment(s, t); maxflow+=maxf;}
};
```

增广路算法

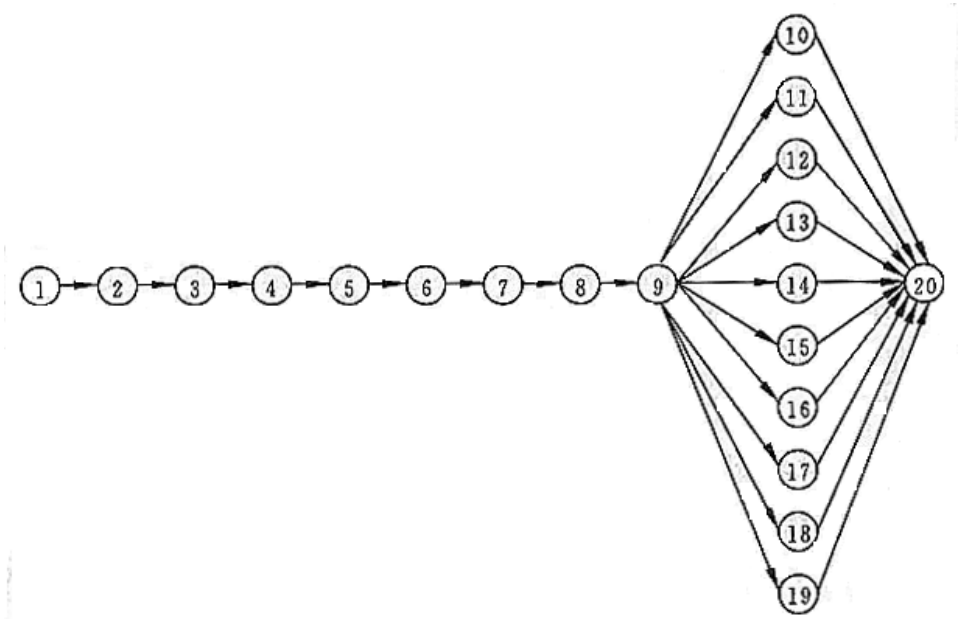
算法的计算复杂性

- 增广路算法的效率由下面 2 个因素所确定
  （1）整个算法找增广路的次数
  （2）每次找增广路所需的时间
- 给定的网络中有 n 个顶点和 m 条边，且每条边的容量不超过 M
- 可以证明，在一般情况下，增广路算法中找增广路的次数不超过 nM 次
- 最短增广路算法在最坏情况下找增广路的次数不超过 nm/2 次
- 找 1 次增广路最多需要 $O(m)$ 计算时间
- 因此，在最坏情况下最短增广路算法所需的计算时间为 $O(nm^2)$
- 当给定的网络是稀疏网络，即 $m=O(n)$ 时，最短增广路算法所需的计算时间为 $O(n^3)$
- 最大容量增广路算法在最坏情况下找增广路的次数不超过 2mlogM 次
- 由于使用堆来存储优先队列，找1次增广路最多需要 O(nlogn) 计算时间
- 因此，在最坏情况下最大容量增广路算法所需的计算时间为 $O(mn\log n\log M)$
- 当给定的网络是稀疏网络时，最大容量增广路算法所需的计算时间为 $O(n^2\log n\log M)$

预流推进算法

算法基本思想

- 增广路算法的特点是找到增广路后，立即沿增广路对网络流进行增广
- 每一次增广可能需要对最多 n-1 条边进行操作
- 最坏情况下，每一次增广需要 $O(n)$ 计算时间
- 有些情况下，这个代价是很高的，下面是一个极端的例子



预流推进算法

- 无论用哪种增广路算法，都会找到 10 条增广路，每条路长为 10，容量为 1
- 共需要 10 次增广，每次增广需要对 10 条边进行操作，每条边增广 1 个单位流量
- 10 条增广路中的前 9 个顶点（前 8 条边）是完全一样的
- 如果直接将前 8 条边的流量增广 10 个单位，而只对后面长为 2 的不同的有向路单独操作，就可以节省许多计算时间
- 这就是预流推进（preflow push)算法的基本思想
- 预流推进算法注重对每一条边的增流，而不必每次一定对一条增广路增流
- 通常将沿一条边增流的运算称为一次推进（push）
- 在算法的推进过程中，网络流满足容量约束，但一般不满足流量平衡约束
- 从每个顶点（除 s 和 t 外）流出的流量之和总是小于等于流入该顶点的流量之和
- 这种流称为预流（preflow），这也是这类算法被称为预流推进算法的原因
- 下面先给出预流的严格定义：
  - 给定网络 G=(V,E)一个预流是定义在 G 的边集 E 上的一个正边流函数
  - 该函数满足容量约束，即对 G 的每一条边 (v,w)∈E，满足 0≤flow(v,w)≤cap(v,w)

预流推进算法

- G 的每一中间顶点满足流出量小于或等于流入量，即对每个 v∈V(v≠s,t)，有：

$$\sum_{(v,w)\in E} flow(v,w) \le \sum_{(w,v)\in E} flow(w,v)$$

- 满足条件 $\sum_{(v,w)\in E} flow(v,w) < \sum_{(w,v)\in E} flow(w,v)$ 的中间顶点v称为**活顶点**，量 $\sum_{(w,v)\in E} flow(w,v) - \sum_{(v,w)\in E} flow(v,w)$ 称为顶点 v 的存流
- 按此定义，源 s 和汇 t 不可能成为活顶点
- 对网络 G 上的一个预流，如果存在活顶点，则说明该预流不是可行流
- 预流推进算法就是要选择活顶点，并通过把一定的流量推进到它的邻点，尽可能地将当前活顶点处正的存流减少为 0，直至网络中不再有活顶点，从而使预流成为可行流
- 如果当前活顶点有多个邻点，那么首先推进到哪个邻点呢?
- 由于算法最后的目的是尽可能将流推进到汇点 t，因此算法应寻求把流量推进到它的邻点中距顶点 t 最近的顶点
- 预流推进算法中用到一个高度函数 h 来确定推流边
- 对于给定网络 G=(V,E) 的一个流，其高度函数 h 是定义在 G 的顶点集 V 上的一个非负函数，该函数满足：
  - 对于 G 的残流网络中的每一条边 (u,v) 有，h(u) ≤ h(v)+1
  - h(t)=0
- G 的残流网络中满足 h(u) = h(v)+1 的边 (u,v) 称为 G 的**可推流边**

一般的预流推进算法

步骤0：构造初始预流 flow：

对源顶点 s的每条出边 (s,v)，令 flow(s,v)=cap(s,v)

对其余边 (u,v) 令 flow(u,v)=0；构造一有效的高度函数 h

步骤1：如果残量网络中不存在活顶点，则计算结束，已经得到最大流；否则转步骤 2

步骤2：在网络中选取活顶点 v

如果存在顶点 v 的出边为可推流边，则选取一条这样的可推流边，并沿此边推流

否则，令 h(v) = min{h(w)+1 | (v,w) 是当前残流网络中的边}，并转步骤 1

- 一般的预流推进算法的每次迭代是一次推进运算或者一次高度重新标号运算
- 如果推进的流量等于推流边上的残留容量，则称为饱和推进，否则称为非饱和推进
- 算法终止时，网络中不含有活顶点，此时只有顶点 s 和 t 的存流非零，此时的预流实际上已经是一个可行流
- 算法预处理阶段已经令 h(s)=n，而高度函数在计算过程中不会减少，因此算法在计算过程中可以保证网络中不存在增广路
- 根据增广路定理，算法终止时的可行流是一个最大流

一般的预流推进算法

- 一般的预流推进算法并未给出如何选择活顶点和可推流边
- 不同的选择策略导致不同的预流推进算法
- 在基于顶点的预流推进算法中，选定一个活顶点后，算法沿该活顶点的所有推流边进行推流运算，直至无可推流边或该顶点的存变成 0 时为止

❖ 算法的计算复杂性

- 基于顶点的预流推进算法用一个广义队列 gQ 存储当前活顶点集合
- 广义队列可以是通常的 FIFO 队列，LIFO 栈，随机化队列，随机化栈，或按各种优先级定义的优先队列
- 算法的效率与广义优先队列的选择密切相关
- 如果选用通常的 FIFO 队列，则在最坏情况下，预流推进算法求最大流所需的计算时间为 $O(mn^2)$，其中 m 和 n 分别为图 G 的边数和顶点数
- 如果以顶点高度值为优先级，选用优先队列实现预流推进算法，则在最坏情况下，求最大流所需的计算时间为 $O(\sqrt{m}n^2)$
- 这个算法也称为最高顶点标号预流推进算法
- 近来已提出许多其它预流推进算法的实现策略，在最坏情况下算法所需的计算时间已接近 $O(mn)$

最小费用流问题

基本概念

网络流的费用

- 在实际应用中，与网络流有关的问题，不仅涉及流量，而且还有费用的因素
- 网络的每一条边 (v,w) 除了给定容量 cap(v,w) 外，还定义了一个单位流量费用 cost(v,w)。对于网络中一个给定的流 flow，其费用定义为：

$$cost(flow) = \sum_{(v,w)\in E} cost(v,w) \times flow(v,w)$$

最小费用流问题

- 给定网络G，要求G的一个最大用流flow，使流的总费用最小

最小费用可行流问题

- 给定多源多汇网络 G，要求 G 的一个可行流 flow，使可行流的总费用最小
- 可行流问题等价于最大流问题，最小费用可行流问题也等价于最小费用流问题

消圈算法

算法基本思想

- 最小费用流问题有关的算法中，仍然沿用残流网络的概念，此时，残流网络中边的费用定义为：

```
int costRto(int v) { return from(v) ? -pcost : pcost; }
```

- 当残流网络中的边是向前边时，其费用不变
- 当残流网络中的边是向后边时，其费用为原费用的负值
- 由于残流网络中存在负费用边，因此残流网络中就不可避免地会产生负费用圈
- 在与最小费用流问题有关的算法中，负费用圈是一个重要概念

最小费用流问题的最优性条件

- 网络 G 的最大流 flow 是 G 的一个最小费用流的充分必要条件是 flow 所相应的残流网络中没有负费用圈

消圈算法

❖ 最小费用流的消圈算法

步骤0：用最大流算法构造最大流 f low

步骤1：如果残量网络中不存在负费用圈，则计算结束，已经找到最小费用流；否则转步骤 2

步骤2：沿找到的负费用圈增流，并转步骤1

消圈算法

算法的计算复杂性

- 给定网络中有 n 个顶点和 m 条边，且每条边的容量不超过 M，每条边的费用不超过 C
- 最大流的费用不超过 mCM，而每次消去负费用圈至少使得费用下降1个单位，因此最多执行 mCM 次找负费用圈和增流运算
- 用 Bellman-Ford 算法找 1 次负费用圈需要 $O(mn)$ 计算时间
- 最小费用流的消圈算法在最坏情况下需要计算时间 $O(m^2nCM)$

最小费用路算法

算法基本思想

- 消圈算法首先找到网络中的一个最大流，然后通过消去负费用圈使费用降低
- 最小费用路算法不用先找最大流，而是用类似于求最大流的增广路算法的思想，不断在残流网络中寻找从源 s 到汇 t 的最小费用路，然后沿最小费用路增流，直至找到最小费用流
- 残流网络中从源 s 到汇 t 的最小费用路是残流网络中从 s 到 t 的以费用为权的最短路
- 残流网络中边的费用定义为：

$$wt(v,w)=\begin{cases}\cos t(v,w) & (v,w)\in P^{+}\\ -\cos t(w,v) & (v,w)\in P^{-}\end{cases}$$

  - 当残流网络中边 (v,w) 是向前边时，其费用为cost(v,w)
  - 当 (v,w) 是向后边时，其费用为 -cost(w,v)

最小费用路算法

最小费用流的最小费用路算法

- **步骤0**：初始可行0流
- **步骤1**：如果不存在最小费用路，则计算结束，已经找到最小费用流；
- 否则，用最短路算法在残流网络中找从s到t的最小费用可增广路，转步骤2
- **步骤2**：沿找到的最小费用可增广路增流，并转步骤1

最小费用路算法

算法的计算复杂性

- 算法的主要计算量在于连续寻找最小费用路并增流
- 给定网络中有 n 个顶点和 m 条边，且每条边的容量不超过 M，每条边的费用不超过 C
- 每次增流至少使得流值增加 1 个单位，因此最多执行 M 次找最小费用路算法
- 如果找 1 次最小费用路需要 s(m,n,C) 计算时间，则求最小费用流的最小费用路算法需要 $O(Ms(m,n,C))$ 计算时间

网络单纯形算法

❖ 算法基本思想

- 消圈算法的计算复杂度不仅与算法找到的负费用圈有关，而且与每次找负费用圈所需的时间有关
- 网络单纯形算法是从解线性规划问题的单纯形算法演变而来，但从算法的运行机制来看，可以将网络单纯形算法看作另一类消圈算法，其基本思想是用一个可行支撑树结构来加速找负费用圈的过程
- 对于给定的网络 G 和一个可行流，相应的**可行支撑树**定义为 G 的一棵包含所有弱流边的支撑树
- 网络单纯形算法的第一步是构造可行支撑树
  - 从一个可行流出发，不断找由弱流边组成的圈，然后沿找到的弱流圈增流，消除所有弱流圈
  - 在剩下的所有弱流边中加入零流边或饱和边构成一棵可行支撑树

网络单纯形算法

- 在可行支撑树结构的基础上，网络单纯形算法通过顶点的势函数，巧妙地选择非树边，使它与可行支撑树中的边构成负费用圈；然后，沿找到的负费用圈增流
- 定义了顶点的势函数 Φ 后，残流网络中各边 (v,w) 的势费用定义为：

$$c^*(v,w)=c(v,w)-(\Phi(v)-\Phi(w))$$

其中，c(v,w) 是 (v,w) 在残流网络中的费用

- 如果对可行支撑树中所有边 (v,w) 有 $c^*(v,w)=0$，则相应的势函数 Φ 是一个有效势函数
- 对于一棵可行支撑树，如果将一条非树边加入可行支撑树，产生残流网络中的一个负费用圈，则称该非树边为一条可用边

❖ **可用边定理**：给定一棵可行支撑树及其上的一个有效势函数，非树边 e 是一条可用边的**充分必要**条件是，e 是一条有正势费用的饱和边，或 e 是一条有负势费用的零流边

网络单纯形算法

- 说明：
  - 事实上，设 e=(v,w)，边 e 与树边 t1,t2,…,td 构成一个圈 cycle: t1,t2,…,t_d，t1，其中 v=t1, w=t_d
  - 按照边的势费用的定义有：
    - $c(w,v)=c^*(w,v)+\Phi(t_d)-\Phi(t_1)$
    - $c(t1,t2)=\Phi(t1)-\Phi(t2)$
    - $c(t2,t3)=\Phi(t2)-\Phi(t3)$
    - …
    - $c(t_{d-1}, t_d)=\Phi(t_{d-1})-\Phi(t_d)$
- 各式相加得：$cost(cycle)=c^*(w,v)$
- 由此可见，e 是一条可用边当且仅当 $cost(cycle)<0$
- 当且仅当 $c^*(w,v)<0$
- 当且仅当 e 是一条有正势费用的饱和边或 e 是一条有负势费用的零流边

网络单纯形算法

- **最优性条件**：给定网络 G 的可行流 flow 及相应的可行支撑树 T，如果不存在 T 的可用边，则 flow 是一个最小费用流
- 事实上，如果不存在 T 的可用边，则由可用边的定义知残流网络中没有负费用圈；又由最小费用流问题的最优性条件可知 flow 是一个最小费用流

最小费用流的网络单纯形算法

步骤0：构造flow为初始可行0流；构造相应的可行支撑树 T 和有效的顶点势函数

步骤1：如果不存在 T 的可用边，则计算结束，已经找到最小费用流；否则转步骤 2

步骤2：选取 T 的一条可用边与 T 的树边构成负费用圈，沿找到的负费用圈增流，从 T 中删去一条饱和边或零流边，重构可行支撑树，并转步骤 1

网络单纯形算法

算法的计算复杂性

- 给定网络中有 n 个顶点和 m 条边，且每条边的容量不超过 M，每条边的费用不超过 C
- 最大流的费用不超过 mCM，而每次消去负费用圈至少使得费用下降 1 个单位，因此最多执行 mCM 次找负费用圈和增流运算
- 用网络单纯形算法找 1 次负费用圈需要 $O(m)$ 计算时间
- 因此，求最小费用流的网络单纯形算法在最坏情况下需要计算时间 $O(m^2CM)$

更多请关注：

http://staff.ustc.edu.cn/~zlwang/