基于 Eva Tardos 提出的一种强多项式时间最小成费用流算法的实现

张景浩 PB20010399

December 2022

1 问题描述

给定一个单向连通的有向图 G=(V,E), 不考虑特定的节点为源点或者汇点, 求在网络 N=(G,c,l,u) 上使费用达到最小的流，即最小费用循环流问题：

$$
\begin{aligned}
&min \sum_{(i,j)\in E} c_{ij}x_{ij} \\
&s.t. \quad \sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0,\ i \in V \\
&\qquad\qquad l_{ij} \le x_{ij} \le u_{ij},\ (i,j) \in E
\end{aligned}
$$

其中，c_{ij}、x_{ij}、l_{ij} 和 u_{ij} 分别表示边 (i,j) 的费用、流量、容量下限和容量上限。

2 算法原理

（本算法调研自 Eva Tardos 提出的一种强多项式时间的最小费用流算法 [1]）

下面对该算法中的一些名词记号进行简单介绍：

一个流称为可行的，若 $l \le x \le u$。所有可行流的集合是一个多胞体，称为流多胞体，表示为 P(l,u)。

最小费用循环流就是所有可行流中，费用最小的流，记为 c-minimal。

我们称一条边是紧的，若 $l_{ij} = u_{ij}$, 所有紧边构成的集合记为 T(l,u)。

要定义流多面体的面，必须要将定义中的一些不等式 $x_{ij} \ge l_{ij}, x_{ij} \le u_{ij}$ 由等式代替。显然，流多胞体的面就是流多胞体本身。最小费用流的集合形成了一个面，称为 P 的 c-minimizer。这个算法将会找到这个面。

定义每个节点的势为 π_i，所有的节点的势构成势向量 $\pi \in R^V$。

两个费用函数 c 和 c' 称为 (l,u) 等价，若存在一个势向量 π 使得对每一条非紧边 (i,j)，都有

$$c'_{ij} = c_{ij} + \pi_i - \pi_j$$

Lemma 2.1. 设 c 和 c' 是 (l,u) 等价的费用函数。一个可行流 x 是 c-minimal 当且仅当 x 是

c'-minimal。

Lemma 2.2. 设 x 是一个可行流，c' 是一个费用函数，π 是势向量，并且 $\alpha > 0$ 是一个实数，使得

$$
\begin{aligned}
&if\ c'_{ij} + \pi_i - \pi_j \geq \alpha \quad (i,j) \in E,\ then\ x_{ij} = l_{ij} \\
&if\ c'_{ij} + \pi_i - \pi_j \leq -\alpha \quad (i,j) \in E,\ then\ x_{ij} = u_{ij}
\end{aligned}
$$

那么对任意 c'-minimal 流 x'，$|c'_{ij} + \pi_i - \pi_j| \geq |V|\alpha$，都有 $x_{ij} = x'_{ij}$。

本算法将找到 P(l,u) 的 c-minimizer。

接下来，我们给出这个算法的具体描述:

迭代步骤

输入：有向图 G=(V,E) 以及容量函数和成本函数 u,l,c，使得存在可行流 x

输出：或者 c 与恒零函数是 (l,u) 等价的；或者新的容量 l',u' 满足：(a)P(l,u) 的 c-minimizer 与 P(l',u') 重合；(b)$T(l,u) \subset T(l',u')$

(0) 令 c* 是 c 关于子空间 Q:={x|x 是流，且 x(e)=0 若 e 是紧的} 的投影。若 d*=0，则 d 与 0 向量是 (l,u) 等价的，stop；否则，令 c' 是 c* 的正数倍，满足 $c'_{max} = V\sqrt{E}$

(1) 定义成本函数 $\bar{c}_{ij} = [c'_{ij}]$，利用 Out-of-Kilter 方法找到 $\bar{c}$-minimal 流 x 和势向量 pi，满足：(i) 若 $\bar{c}_{ij} + \pi_i - \pi_j > 0$，则 $x_{ij} = l_{ij}$；(ii) 若 $\bar{c}_{ij} + \pi_i - \pi_j < 0$，则 $x_{ij} = u_{ij}$

(2) 定义新容量:令 $l'_{ij} = l_{ij}$ 和 $u'_{ij} = u_{ij}\ (i,j) \in E$,若 $|c'_{ij}+\pi_i-\pi_j| < V$;$l'_{ij} = u'_{ij} = x_{ij}\ \ (i,j) \in E$,若 $|c'_{ij} + \pi_i - \pi_j| \geq V$

完整算法如下，重复迭代步骤直到得到容量 l* 和 u* 使得 c 与恒零函数是 (l*,u*) 等价的。此时，P(l,u) 的 c-minimizer 是 P(l*,u*).

Lemma 2.3. P(l',u') 的 c-minimizer 与 P(l,u) 的 c-minimizer 是一致的。

Lemma 2.4. $T(l,u) \subset T(l',u')$，且 $T(l,u) \neq T(l',u')$.

其中 Out-of-Kilter 方法由 D. R. FULKERSON 提出 [2], 具体实现参考自《网络优化》[3]。下面对 Out-of-Kilter 方法的实现给出描述：

Out-of-Kilter 方法

输入：有向图 G=(V,E) 以及容量函数和成本函数 u,l,c

输出：最小费用流 x 和对应势向量 π

(0) 给出初始流 x=0，初始势向量 π=0

(1) 计算弧上的 kilter 数，若网络中不存在瑕疵弧，stop，返回 x 和 π；否则在残量网络中 N(x) 中，选择一条瑕疵弧 (p,q)

(2) 在 $N(x)\backslash\{(q,p)\}$ 中以 max{0，c^{π}_{ij}} 为 (i,j) 弧的弧长，计算从节点 q 到所有节点 i 的最短路路长 d_i, 并记从节点 q 到节点 p 的最短路为 P，令 π_i=π_i-d_i

(3) 若 (p,q) 仍为瑕疵弧，则沿 $P\bigcup\{(p,q)\}$ 确定的增广圈增广流量（增广流值为增广圈容量），返回 (1); 否则，直接返回 (1)

算法时间复杂度： 由 [1] 中结论，设 T 是 Out-of-Kilter 方法所用时间，则本算法时间复杂度为 $O(E^{3/2}V^2T)$

3 数据集说明

3.1 数据集 1.

边起点集：{0, 0, 1, 1, 1, 2, 2, 3, 3, 4}

边终点集：{1, 2, 2, 3, 4, 3, 4, 4, 0, 0}

边容量上限：{(0, 1): 15, (0, 2): 8, (1, 2): 20, (1, 3): 4, (1, 4): 10, (2, 3): 15, (2, 4): 4, (3, 4): 20, (3, 0): 5, (4, 0): 15}

边容量下限：{(0, 1): 1, (0, 2): 2, (1, 2): 2, (1, 3): 1, (1, 4): 0, (2, 3): 1, (2, 4): 4, (3, 4): 9, (3, 0): 5, (4, 0): 15}

边单位费用：{(0, 1): 4, (0, 2): 4, (1, 2): 2, (1, 3): 2, (1, 4): 6, (2, 3): 1, (2, 4): 3, (3, 4): 2, (3, 0): 0, (4, 0): 0}

3.2 数据集 2.

边起点集：{0, 0, 1, 1, 2, 2, 3, 4, 4, 5}

边终点集：{1, 3, 2, 3, 3, 5, 4, 2, 5, 0}

边容量上限：{(0, 1): 8, (0, 3): 7, (1, 2): 9, (1, 3): 5, (2, 3): 2, (2, 5): 5, (3, 4): 9, (4, 2): 6, (4, 5): 10, (5, 0): 14}

边容量下限：{(0, 1): 3, (0, 3): 0, (1, 2): 1, (1, 3): 2, (2, 3): 0, (2, 5): 1, (3, 4): 2, (4, 2): 0, (4, 5): 4, (5, 0): 10}

边单位费用：{(0, 1): 2, (0, 3): 8, (1, 2): 2, (1, 3): 5, (2, 3): 1, (2, 5): 6, (3, 4): 3, (4, 2): 4, (4, 5): 7, (5, 0): 0}

3.3 数据集 3.

边起点集：{0, 0, 1, 1, 2, 2, 3, 4}

边终点集：{1, 2, 3, 4, 1, 3, 4, 0}

边容量上限：{(0, 1): 10.3, (0, 2): 8.1, (1, 3): 2.3, (1, 4): 7, (2, 1): 5.4, (2, 3): 10, (3, 4): 4, (4, 0): 12}

边容量下限：{(0, 1): 1.3, (0, 2): 0.1, (1, 3): 0, (1, 4): 2.1, (2, 1): 0.3, (2, 3): 2.2, (3, 4): 0, (4, 0): 5}

边单位费用: {(0, 1): 4.2, (0, 2): 1.5, (1, 3): 6.3, (1, 4): 1, (2, 1): 2, (2, 3): 3.7, (3, 4): 2.1, (4, 0): 1}

4 关键代码展示

```
def algorithm(D, g, f, d):
    """
    输入有向图D=(V,E)以及容量函数和成本函数g,f,d
    """
    V = D.number_of_nodes()
```

```python
    E = D.number_of_edges()
    x = {}
    while (1):
        n = 0
        d_1 = {}
        for (i, j) in D.edges():
            if f[(i, j)] != g[(i, j)]:
                d_1[(i, j)] = d[(i, j)]
            else:
                d_1[(i, j)] = 0
                n += 1
        if n == E:
            break
        m = max(d_1.values())
        k = V*math.sqrt(E)*1.0/m
        for (i, j) in D.edges():
            d_1[(i, j)] *= k
        d_2 = {}
        for (i, j) in D.edges():
            d_2[(i, j)] = math.floor(d_1[(i, j)])
        a, x, pi = out_of_kilter(D, g, f, d_2)
        for (u, v) in D.edges():
            if math.fabs(d_1[(u, v)]+pi[u]-pi[v]) >= V:
                f[(u, v)] = x[(u, v)]
                g[(u, v)] = x[(u, v)]
    if a:
        print("不存在可行流！")
    return a, x, g, f
```

```python
def out_of_kilter(D, g, f, d):
    """
    利用瑕疵算法计算最小费用流x以及势向量pi
    """
    a = False # 存在可行流
    V = D.number_of_nodes()
    x = {}
    for (u, v) in D.edges():
        x[(u, v)] = 0
    pi = np.zeros(V, dtype=int)
    while True:
        # step1
```

```
        # 构造残量网络N(x)
        N, d_1, g_1 = residual_network(D, x, g, f, d)
        # 计算N(x)每条弧的kilter值
        c = {}
        for (i, j) in N.edges():
            c[(i, j)] = d_1[(i, j)]-pi[i]+pi[j]
        for (i, j) in N.edges():
            k = kilter(D, c, x, g, f, g_1, i, j)
            if k < 1e-12:
                k = 0
            if k > 0:  # 若弧(p,q)是瑕疵弧，跳出检索进入step2
                p = i
                q = j
                break
        if k == 0:
            return a, x, pi  # 若没有瑕疵弧，则返回最小费用流x和势向量pi
        # step2
        if N.has_edge(q, p):
            N.remove_edge(q, p)
        # 构造以max{0, C^{pi}_{ij}}为边长的，由N得到的带权有向图M
        M = nx.MultiDiGraph()
        for (i, j) in N.edges():
            M.add_weighted_edges_from([(i, j, max(0, c[(i, j)]))])
        # 使用Bellman-ford计算从节点q到所有节点i的最短路径
        y = np.zeros(len(pi), dtype=int)
        for i in M.nodes():
            if nx.has_path(M, q, i):
                y[i] = nx.bellman_ford_path_length(M, source=q, target=i)
            else:
                a = True
                return a, x, pi
        P = nx.bellman_ford_path(M, source=q, target=p)
        pi = np.subtract(pi, y)
        # step3
        c[(p, q)] = d_1[(p, q)]-pi[p]+pi[q]
        k_1 = kilter(D, c, x, g, f, g_1, p, q)  # 计算弧(p,q)的kilter数
        if k_1 < 1e-12:
            k_1 = 0
        if k_1 != 0:
            # 确定增广圈
```

```python
            Q = nx.MultiDiGraph()
            Q.add_edge(p, q)
            for i in range(0, len(P)-1):
                Q.add_edge(P[i], P[i+1])
            r = g_1[(p, q)]
            for (i, j) in Q.edges():
                if r > g_1[(i, j)]:
                    r = g_1[(i, j)]
            for (i, j) in Q.edges():
                if D.has_edge(i, j):
                    x[(i, j)] += r
                else:
                    x[(j, i)] -= r
```

5 程序输入输出说明

输入数据集时，边的起始点分别以列表的形式输入，边对应的容量上下限以及成本分别以字典的形式输入。输出结果时，先输出使得 c 与恒零函数 (l*,u*) 等价的 l* 和 u*，再输出网络中最小的费用值，最后以 $(i,j) \to x_{ij}$ 的形式输出每条边上的流值。若数据集不存在可行流，则返回：“不存在可行流！”

注：测试所用的数据集 1、2、3 均在.py 文件中写出，以注释的方式选择调用。故程序中不采用以用户输入的形式调用数据集。

6 程序测试结果

6.1 数据集 1.

```
最小费用循环流的解集合P(f*,g*):
f*= {(0, 1): 12, (0, 2): 8, (1, 2): 8, (1, 3): 4, (1, 4): 0, (2, 3): 12, (2, 4): 4, (3, 4): 11, (3, 0): 5, (4, 0): 15}
g*= {(0, 1): 12, (0, 2): 8, (1, 2): 8, (1, 3): 4, (1, 4): 0, (2, 3): 12, (2, 4): 4, (3, 4): 11, (3, 0): 5, (4, 0): 15}
最小费用为:  150
最小费用循环流为:
(0, 1)  ->  12
(0, 2)  ->  8
(1, 2)  ->  8
(1, 3)  ->  4
(1, 4)  ->  0
(2, 3)  ->  12
(2, 4)  ->  4
(3, 4)  ->  11
(3, 0)  ->  5
(4, 0)  ->  15
算法运行时间:  0.013003349304199219 单位: s
```

6.2 数据集 2.

```
最小费用循环流的解集合P(f*,g*):
f*= {(0, 1): 8, (0, 3): 2, (1, 2): 6, (1, 3): 2, (2, 3): 1, (2, 5): 5, (3, 4): 5, (4, 2): 0, (4, 5): 5, (5, 0): 10}
g*= {(0, 1): 8, (0, 3): 2, (1, 2): 6, (1, 3): 2, (2, 3): 1, (2, 5): 5, (3, 4): 5, (4, 2): 0, (4, 5): 5, (5, 0): 10}
最小费用为:  135
最小费用循环流为:
(0, 1)  ->  8
(0, 3)  ->  2
(1, 2)  ->  6
(1, 3)  ->  2
(3, 4)  ->  5
(2, 3)  ->  1
(2, 5)  ->  5
(5, 0)  ->  10
(4, 2)  ->  0
(4, 5)  ->  5
算法运行时间:  0.01200246810913086 单位: s
```

6.3 数据集 3.

```
最小费用循环流的解集合P(f*,g*):
f*= {(0, 1): 1.3, (0, 2): 3.6999999999999993, (1, 3): 0, (1, 4): 2.7999999999999994, (2, 1): 1.4999999999999993, (2, 3): 2.2, (3, 4): 2.2, (4, 0): 5.0}
g*= {(0, 1): 1.3, (0, 2): 3.6999999999999993, (1, 3): 0, (1, 4): 2.7999999999999994, (2, 1): 1.4999999999999993, (2, 3): 2.2, (3, 4): 2.2, (4, 0): 5.0}
最小费用为:  34.57
最小费用循环流为:
(0, 1)  ->  1.3
(0, 2)  ->  3.6999999999999993
(1, 3)  ->  0
(1, 4)  ->  2.7999999999999994
(2, 1)  ->  1.4999999999999993
(2, 3)  ->  2.2
(3, 4)  ->  2.2
(4, 0)  ->  4.999999999999999
算法运行时间:  0.015003442764282227 单位: s
```

7 分析总结

从数据集的测试结果来看，该算法可以解决最小费用循环流问题，并且能够处理数据中出现非整数的情况，考虑到浮点数运算带来的误差，计算结果在可接受范围内。在对算法的应用上，由于使用 Out-of-Kilter 方法的过程中需要构造网络 G 在流 x 下的余网络，故对有向图要求是单项连通的。算法本身并不算太复杂，核心问题在于采用 Out-of-Kilter 方法求出可行流 x 和势向量 π，其中包含了利用对偶问题求解原问题的思想，在未来研究解决其他问题的算法时可以借鉴。

参考文献

[1] Tardos, É. A strongly polynomial minimum cost circulation algorithm. Combinatorica 5, 247–255 (1985).

[2] Fulkerson, D. R. An Out-of-Kilter Method for Minimal Cost Flow Problems, RAND Corporation, P-1825, 1960.

[3] (美) 博赛卡斯 (Bertsekas,D.P.) , 著. 网络优化: 连续和离散模型: 信息技术和电气工程学科国际知名教材中译本系列 [M]. 北京: 清华大学出版社,2013.