具有时延约束的无线网络流量控制与调度

电子信息与电气工程学院自动化系　F1703203　岳博(517021910825)　徐加声(517021910958)

一、实验目的

研究具有时延约束的无限网络流量控制与基于冲突图的调度
研究存在非加性耦合时，Lagrange 对偶方法的变换思路
研究 Lagrange 对偶方法的求解思路并推导过程
通过动手实践构造算例，验证分布式算法的有效性和局限性

二、实验场景

一个多跳无线网络由一组无线节点$V=\{1,2,\cdots,|V|\}$组成，形成链路集合为$L=\{1,2,\cdots,|L|\}$。令S代表端到端的流的集合，每条流s有一个效用函数$U_s(x_s)$。集合$L(s)$代表流s路由经过的所有链路组合。集合$S(l)$代表经过链路l的所有流的集合。每一条链路l的最大速率为c_l。在每一时隙，基于冲突图，通过链路层调度，选出并允许互不干扰的链路通信，则链路l的平均速率为$\hat{c_l}$。对应每条链路有相应的排队时延d_l。每条流有端到端时延限制D_s，假设D_s主要由路由经过的链路的平均时延之和构成。通过调节信源速率x_s，以及链路调度，在满足端到端时延限制前提下，最大化系统效用 $\sum_{s\in S}U_s(x_s)$。

三、实验步骤

对于本次大作业，按照三个小问，我们将大作业步骤分为三步：1）基于 Lagrange 对偶方法，设计分布式算法满足上述要求；2）证明所设计算法的收敛性；3）构造算例，通过编程，验证上述算法的有效性。

3.1 基于 Lagrange 对偶方法，设计分布式算法

按照排队论中的 M/M/1 队列模型模拟信息传输过程。这里假设 1）信息到达某链路满足泊松流的特征，且到达速率为λ，同时链路内的服务时间服从负指数分布。可以得到平均时延为$\frac{1}{\mu-\lambda}$，其中μ是负指数分布的参数，这里μ的含义是链路的平均服务速率；2）效用函数$U_s(x_s)$是关于x_s的凹函数。

在构造优化问题时，对于排队时延存在限制条件，该限制条件存在非加性耦合（即各个x_s不能独立地分解开），会使对偶分解法失效。为此，我们对每条链路引入链路裕度变量σ_l，进行变换，从而得到适合对偶分解的目标函数及限制条件。

$$\sum_{l\in L(s)}\frac{1}{\hat{c_l}-\sum_{s\in S(l)}x_s}\le D_s$$

进而，

$$\sigma_l\le\hat{c_l}-\sum_{s\in S(l)}x_s$$

$$\frac{1}{\hat{c}_l - \sum_{s\in S(l)} x_s} \le \frac{1}{\sigma_l} = \varphi_l(\sigma_l)$$

$$\sum_{l\in L(s)} \varphi(\sigma_l) \le D_s$$

通过分别对每条链路的排队时延求和，可得到适合对偶分解的限制条件。接下来，我们构建可对偶分解的目标函数和限制条件。

$$max \sum_{s\in S} U_s(x_s) \tag{1}$$

$$s.t. \sum_{s\in S(l)} x_s \le \hat{c}_l - \sigma_l, \forall l \in L \tag{2}$$

$$\sum_{l\in L(s)} \varphi(\sigma_l) \le D_s, \forall s \in S \tag{3}$$

$$over\ \hat{c} \in \wedge \tag{4}$$

(1)式是目标函数，（2）式是拥塞限制条件，（3）式是端到端时延约束条件，（4）式是冲突域限制条件。

由 Lagrange 对偶方法，可得四项 Lagrange 函数如下（μ_s, p_l是拉格朗日系数）：

$$L(x,p,\sigma,\mu,\hat{c})$$

$$= \sum_{s\in S(l)} U_s(x_s) - \sum_{s\in S(l)} \mu_s[\sum_{l\in L(s)} \varphi_l(\sigma_l) - D_s] - \sum_{l\in L(s)} p_l[\sum_{s\in S(l)} x_s - \hat{c}_l + \sigma_l]$$

$$= \sum_{s\in S(l)} [U_s(x_s) - x_s(\sum_{l\in L(s)} p_l)] - \sum_{l\in L(s)} [p_l\sigma_l + \sum_{s\in S(l)} \varphi_l(\sigma_l)\mu_s] + \sum_{l\in L(s)} p_l\hat{c}_l + \sum_{s\in S(l)} \mu_s D_s$$

进而得到对偶目标函数（粗体字符表示对应维度的向量）：

$$D(\boldsymbol{p},\boldsymbol{\mu}) = max_{\boldsymbol{x},\boldsymbol{\sigma},\boldsymbol{\mu}} L(\boldsymbol{x},\boldsymbol{p},\boldsymbol{\sigma},\boldsymbol{\mu},\hat{\boldsymbol{c}})$$

$$= max_{\boldsymbol{\sigma}}\{-\sum_{l\in L(s)} [p_l\sigma_l + \sum_{s\in S(l)} \varphi_l(\sigma_l)\mu_s]\} + max_{\boldsymbol{x}}\{\sum_{s\in S(l)} [U_s(x_s) - x_s(\sum_{l\in L(s)} p_l)]\} + max_{\hat{\boldsymbol{c}}}\{\sum_{l\in L(s)} p_l\hat{c}_l\} + \sum_{s\in S(l)} \mu_s D_s$$

由假设 2）可知前两项均为凹函数，故可利用一阶条件求解，得到最优解。

$$x_s{}^* = argmax_{x_s} \sum_{s\in S(l)} [U_s(x_s) - x_s(\sum_{l\in L(s)} p_l)] = {U_s}'^{-1}(q_s),\ q_s = \sum_{l\in L(s)} p_l, \forall s \in S(L) \tag{5}$$

$$\sigma_l{}^* = argmax_{\sigma_l} -\sum_{l\in L(s)} [p_l\sigma_l + \sum_{s\in S(l)} \varphi_l(\sigma_l)\mu_s] = {\varphi_l}'^{-1}\left(\frac{-p_l}{\eta_l}\right), \eta_l = \sum_{s\in S(l)} \mu_s, \forall l \in L(s) \tag{6}$$

两式中，${U_s}'^{-1}(q_s)$表示${U_s}'(q_s)$的反函数，${\varphi_l}'^{-1}(\frac{-p_l}{\eta_l})$表示${\varphi_l}'(\frac{-p_l}{\eta_l})$的反函数。

对第三项，在相互冲突的冲突域中，选出使$p_l c_l$值最大的链路l，在当前时刻传输，并令$\hat{c}_l = c_l$。

$$\hat{c}_l{}^* = argmax_{\hat{c}_l\in\wedge}\left(\sum_{l\in L(s)} p_l\hat{c}_l\right) \tag{7}$$

对第四项，由于端到端时延限制D_s为常量且在分布式算法中μ_s已经更新，故不参与求最大值的过程。

对于拉格朗日系数（对偶变量），采用梯度算法进行更新。

$$p_l(t+1) = \left[p_l(t) - \beta\left(\frac{\partial}{\partial p_l} D(\boldsymbol{p},\boldsymbol{\mu})\right)\right]^+ = [p_l(t) + \beta(\sum_{s\in S(l)} x_s - \hat{c}_l + \sigma_l)]^+ \tag{8}$$

$$\mu_s(t+1) = \left[\mu_s(t) - \gamma\left(\frac{\partial}{\partial \mu_s} D(\boldsymbol{p},\boldsymbol{\mu})\right)\right]^+ = [\mu_s(t) + \gamma[\sum_{l\in L(s)} \varphi_l(\sigma_l) - D_s]]^+ \tag{9}$$

这里，$[x]^+ = max\{0,x\}$，这里的非负性是为了保证了拉格朗日系数（对偶变量）的非负性。

综上所述，可得到**分布式算法**如下：

①初始化拉格朗日系数（对偶变量）和迭代步长，即$p_l(0)$和$\mu_s(0)$，β和γ；

②由 7）式选出使$p_l c_l$值最大的链路l，求得$\hat{c}_l^*$，令$\hat{c}_l = c_l$；

③由已知的$p_l(t)$，代入 5）式求得x_s^*；

④由已知的$\mu_s(t)$，代入 6）式求得σ_l^*；

⑤由 8）式迭代更新$p_l(t+1)$；

⑥由 9）式迭代更新$\mu_s(t+1)$；

⑦重复②~⑤直至满足迭代结束条件。

3.2 证明所设计算法的收敛性

这边从更宏观的视角来证明对偶梯度分布式算法的收敛性。

对原 Lagrange 函数$L(\boldsymbol{X},\boldsymbol{P}) = U(\boldsymbol{X}) - \boldsymbol{P}^T(g(\boldsymbol{X}))$（X 是主变量，P 是拉格朗日系数，$g$(X)是限制条件），构造对偶问题，可得：

$$D(\boldsymbol{P}^*) = min_{\boldsymbol{P}\geq\boldsymbol{0}} D(\boldsymbol{P}) = U(\boldsymbol{X}^*)$$

$$D(\boldsymbol{P}) = max_{\boldsymbol{X}}\{U(\boldsymbol{X}) - \boldsymbol{P}^{\boldsymbol{T}}(g(\boldsymbol{X}))\}$$

由分布式算法可得迭代过程为：$\boldsymbol{P}(t+1) = [\boldsymbol{P}(t) + \alpha g(\boldsymbol{X})]^+, \alpha \in (0,1)$。

以下证明对偶梯度算法的收敛性。

$$||\boldsymbol{P}(t+1) - \boldsymbol{P}^*||_2^{\,2} = ||[\boldsymbol{P}(t) + \alpha g(\boldsymbol{X})]^+ - \boldsymbol{P}^*||_2^{\,2}$$

$$\leq ||\boldsymbol{P}(t) + \alpha g(\boldsymbol{X}) - \boldsymbol{P}^*||_2^{\,2}$$

$$= ||\boldsymbol{P}(t) - \boldsymbol{P}^*||_2^{\,2} - 2\alpha g^T(\boldsymbol{X})(-\boldsymbol{P}(t) + \boldsymbol{P}^*) + \alpha^2||g(\boldsymbol{X})||_2^{\,2}$$

$$= ||\boldsymbol{P}(t) - \boldsymbol{P}^*||_2^{\,2} - 2\alpha[(U(\boldsymbol{X}^*) - g^T(\boldsymbol{X})\boldsymbol{P}(t)) - (U(\boldsymbol{X}^*) - g^T(\boldsymbol{X})\boldsymbol{P}^*)] + \alpha^2||g(\boldsymbol{X})||_2^{\,2}$$

由于效用函数$U(\boldsymbol{X})$是凹函数，所以对偶函数$D(\boldsymbol{P})$是凸函数，所以$D(\boldsymbol{P})$的梯度函数$-g(\boldsymbol{X})$单调递增，$g(\boldsymbol{X})$单调递减。在梯度更新时，从原问题来看，由于$U(\boldsymbol{X})$是凹函数，所以为了取到最大值，$\boldsymbol{X}$需要变大。由于$\boldsymbol{X}$变大，$g(\boldsymbol{X})$变小，所以$-g^T(\boldsymbol{X}) \leq -g^T(\boldsymbol{X}^*)$；又由于$-U(\boldsymbol{X}^*) \leq -U(\boldsymbol{X})$，代入可得：

$$\leq ||\boldsymbol{P}(t) - \boldsymbol{P}^*||_2^{\,2} - 2\alpha[(U(\boldsymbol{X}) - g^T(\boldsymbol{X})\boldsymbol{P}(t)) - (U(\boldsymbol{X}^*) - g^T(\boldsymbol{X}^*)\boldsymbol{P}^*)] + \alpha^2||g(\boldsymbol{X})||_2^{\,2}$$

$$= \left|\left|\boldsymbol{P}(t) - \boldsymbol{P}^*\right|\right|_2^{\,2} - 2\alpha[D(\boldsymbol{P}(t)) - D(\boldsymbol{P}^*)] + \alpha^2\left|\left|g(\boldsymbol{X})\right|\right|_2^{\,2} \qquad (10)$$

将不等式 10）从 t=1 进行叠加，得到：

$$\left|\left|\boldsymbol{P}(t+1) - \boldsymbol{P}^*\right|\right|_2^{\,2} \leq \left|\left|\boldsymbol{P}(1) - \boldsymbol{P}^*\right|\right|_2^{\,2} - 2\alpha\sum_{\tau=1}^{t}(D(\boldsymbol{P}(\tau)) - D(\boldsymbol{P}^*)) + \alpha^2\sum_{\tau=1}^{t}\left|\left|g(\boldsymbol{X}(\tau))\right|\right|_2^{\,2}$$

由于$||\boldsymbol{P}(t+1) - \boldsymbol{P}^*||_2^{\,2} \geq 0$，得到：

$$2\alpha\sum_{\tau=1}^{t}[D(\boldsymbol{P}(\tau)) - D(\boldsymbol{P}^*)] \leq ||\boldsymbol{P}(1) - \boldsymbol{P}^*||_2^{\,2} + \alpha^2||g(\boldsymbol{X})||_2^{\,2}$$

令$\left|\left|g(\boldsymbol{X}(\tau))\right|\right|_2^{\,2} \leq G_{max}$，代入后不等式两边同除以$2\alpha t$，得到：

$$\frac{1}{t}\sum_{\tau=1}^{t}[D(\boldsymbol{P}(\tau)) - D(\boldsymbol{P}^*)] \leq \frac{1}{2\alpha t}||\boldsymbol{P}(1) - \boldsymbol{P}^*||_2^{\,2} + \frac{\alpha(G_{max})^2}{2}$$

由凸函数 Jensen's 不等式且$D(\boldsymbol{P}(t))$是凸函数，得到：

$$D(\bar{\boldsymbol{P}}(t)) - D(\boldsymbol{P}^*) = D(\frac{1}{t}\sum_{\tau=1}^{t}\boldsymbol{P}(t)) - D(\boldsymbol{P}^*) \leq \frac{1}{t}\sum_{\tau=1}^{t}\left[D\left(\boldsymbol{P}(\boldsymbol{t})\right) - D(\boldsymbol{P}^*)\right]$$

取极限得到:

$$0 \le \lim_{t\to\infty} D(\bar{\boldsymbol{P}}(t)) - D(\boldsymbol{P}^*) \le \frac{\alpha(G_{max})^2}{2}$$

前一个不等号是由于$D(\boldsymbol{P})$是凸函数。从而得到$\lim_{t\to\infty} D(\bar{\boldsymbol{P}}(t))$以一定误差限收敛到$D(\boldsymbol{P}^*)$。

3.3 构造算例，通过编程，验证上述算法有效性

首先，定性描述分布式算法的有效性。通过赋给分布式算法中各参数物理意义，根据关系式中的单调性（凹函数的导数单调减，凸函数的导数单调增），可知：当端到端时延μ_s变大时，η_l变大，链路的裕度变量σ_l变大（链路能提供的实际服务速率下降），进而带宽价格p_l变大，进而q_s变大，进而信号源速率x_s变小。可见，当端到端时延变大时，信号源速率变小，符合调控思想。

其次，我们构造了一个新的算例，并绘制出了它的网络拓扑图，并基于如下两点假设画出了信息流冲突图：1）如果两个节点之间的距离小于等于两跳，那么这两个节点可以相互监听；2）链路均为半双工，且同一时间只能接受来自一个节点的信息。另外，我们假定该算例中，效用函数表示为：$U_S(x_s) = \ln x_s$。

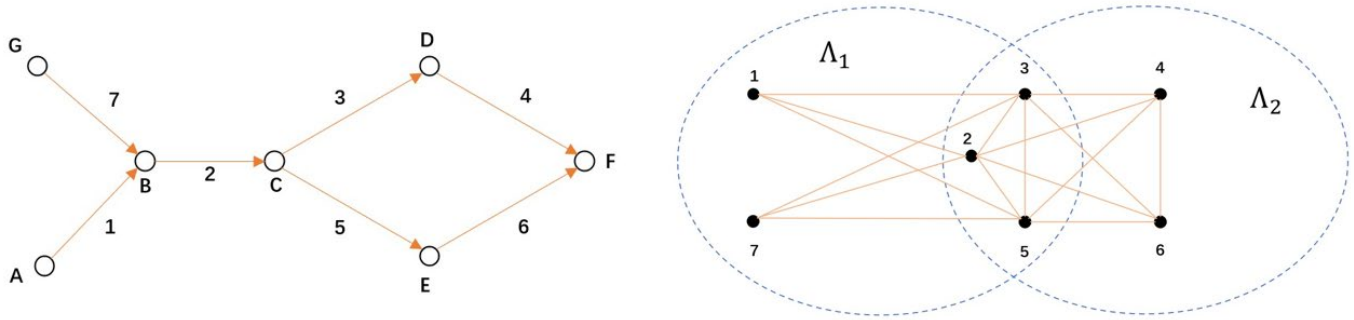

图 1 左图为网络拓扑图，右图为相应的信息流冲突图

最后，我们编程定量验证分布式算法的有效性，代码如下:

```
import numpy as np
import math
import matplotlib.pyplot as plt

# 算例实例化
################################################################################
# 假定效用函数为 Us(x) = ln(x)
# 四条流 s1，s2，s3，s4，传输时延 Ds
s1 = np.array([1, 2, 3, 4])
s2 = np.array([1, 2, 5, 6])
s3 = np.array([7, 2, 3, 4])
s4 = np.array([7, 2, 5, 6])
Ds = 10

# S(l)
s_l = [(1, 2), (1, 2, 3, 4), (1, 3), (1, 3), (2, 4), (2, 4), (3, 4)]
```

```
17.
18. # 冲突域Λ
19. r1 = np.array([1, 2, 3, 5, 7])
20. r2 = np.array([2, 3, 4, 5, 6])
21. tmp = np.array([val for val in r1 if val in r2])  # 两个冲突域重叠部分
22.
23.
24. def comb(r1, r2, tmp):  # 所有可能的链路组合
25.     get1 = list(tmp)
26.     for i in r1:
27.         if i not in tmp:
28.             for j in r2:
29.                 if j not in tmp:
30.                     get1.append((i, j))
31.     return get1
32.
33.
34. get = comb(r1, r2, tmp)
35. ################################################################################
36.
37. # 初始值设定
38. ################################################################################
39. # 对偶算子迭代步长
40. beta = 0.005
41. gamma = 0.005
42.
43. # 初始化对偶算子表，横轴时间，纵轴节点或链路
44. num = 15000  # 对偶算子迭代次数
45. lian = 7  # 链路个数
46. liu = 4  # 流的个数
47. p = np.array([53.0] * lian)  # 拉格朗日系数 p 初始化
48. m = np.array([21.0] * liu)  # 拉格朗日系数 μ 初始化
49.
50. # 初始化各链路裕度变量 σ
51. sigma = np.array([0.1] * lian)
52.
53. # 初始化每一条链路 l 的最大速率 c，c_l 代表实际速率（有的会被置零））
54. c = np.array([4.0] * lian)
55. ################################################################################
56.
57. # 迭代过程用于记录的数组
58. k = len(get)
59. lian_choose = np.array([0.0] * k)
60. xs1 = np.array([0.0] * num)
61. xs2 = np.array([0.0] * num)
```

```
xs3 = np.array([0.0] * num)
xs4 = np.array([0.0] * num)
m_ = [[] for i in range(4)]
p_ = [[] for i in range(7)]

# 迭代过程
for t in range(num):
    # 选出使 p_l*c_l 最大的链路
    a = 0
    for i in get:
        if a <= len(tmp) - 1:
            lian_choose[a] = p[i - 1] * c[i - 1]
        else:
            lian_choose[a] = p[i[0] - 1] * c[i[0] - 1] + p[i[1] - 1] * c[i[1] - 1]
        a += 1
    index = np.argmax(lian_choose)

    # 计算 x_s*
    qs1 = np.sum(np.array([p[val - 1] for val in s1]))
    qs2 = np.sum(np.array([p[val - 1] for val in s2]))
    qs3 = np.sum(np.array([p[val - 1] for val in s3]))
    qs4 = np.sum(np.array([p[val - 1] for val in s4]))

    def us_d_i(qs):  # us 求导再求反函数
        x = 1 / qs
        return x
    xs1[t] = us_d_i(qs1)
    xs2[t] = us_d_i(qs2)
    xs3[t] = us_d_i(qs3)
    xs4[t] = us_d_i(qs4)

    # 计算 σ_l*
    ita = np.array([0.0] * lian)
    for i in range(lian):
        if len(s_l[i]) == 2:
            ita[i] = m[s_l[i][0] - 1] + m[s_l[i][1] - 1]
        else:
            ita[i] = m[s_l[i][0] - 1] + m[s_l[i][1] - 1] + m[s_l[i][2] - 1] + m[s_l[i][3] - 1]

     for i in range(lian):
         sigma[i] = math.sqrt(ita[i] / p[i])

    # 更新 p_l
    c_l = np.array([0.0] * lian)
    if type(get[index]) == tuple:
```

```
            c_l[get[index][0] - 1] = c[get[index][0] - 1]
            c_l[get[index][1] - 1] = c[get[index][1] - 1]
        else:
            c_l[get[index] - 1] = c[get[index] - 1]

        for i in range(lian):
            if len(s_l[i]) == 4:
                p[i] = max(0.01, p[i] + beta * (sigma[i] - c_l[i] + xs1[t] + xs2[t] + xs3[t] + xs4[t]
    ))
            elif s_l[i] == (1, 2):
                p[i] = max(0.01, p[i] + beta * (sigma[i] - c_l[i] + xs1[t] + xs2[t]))
            elif s_l[i] == (1, 3):
                p[i] = max(0.01, p[i] + beta * (sigma[i] - c_l[i] + xs1[t] + xs3[t]))
            elif s_l[i] == (2, 4):
                p[i] = max(0.01, p[i] + beta * (sigma[i] - c_l[i] + xs2[t] + xs4[t]))
            elif s_l[i] == (3, 4):
                p[i] = max(0.01, p[i] + beta * (sigma[i] - c_l[i] + xs3[t] + xs4[t]))
            p_[i].append(p[i])

        # 更新 m_s
        sigma1 = np.array([0.0] * lian)
        # print(sigma)
        for i in range(lian):
            sigma1[i] = 1 / sigma[i]
            # print(sigma1[i])

        m[0] = max(0, m[0] + gamma * (sigma1[0] + sigma1[1] + sigma1[2] + sigma1[3] - Ds))
        m[1] = max(0, m[1] + gamma * (sigma1[0] + sigma1[1] + sigma1[4] + sigma1[5] - Ds))
        m[2] = max(0, m[2] + gamma * (sigma1[6] + sigma1[1] + sigma1[2] + sigma1[3] - Ds))
        m[3] = max(0, m[3] + gamma * (sigma1[6] + sigma1[1] + sigma1[4] + sigma1[5] - Ds))
        m_[0].append(m[0])
        m_[1].append(m[1])
        m_[2].append(m[2])
        m_[3].append(m[3])


# 总效用函数计算
def us_calculation():
    return np.log(xs1) + np.log(xs2) + np.log(xs3) + np.log(xs4)


# 可视化部分
plt.rcParams['font.family'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

```

```
plt.plot(us_calculation())
plt.xlabel('迭代次数')
plt.ylabel("效用函数 Us")
plt.title("效用函数值变化曲线")
plt.savefig("效用函数值变化曲线.png")
plt.show()

xs = [xs1, xs2, xs3, xs4]
xs_name = ['Xs1', 'Xs2', 'Xs3', 'Xs4']
for i in range(len(xs)):
    plt.plot(xs[i])
    plt.xlabel('迭代次数')
    plt.ylabel("信号源速率" + xs_name[i])
    plt.title("信号源速率" + xs_name[i] + "变化曲线")
    plt.savefig("信号源速率" + xs_name[i] + "变化曲线.png")
    plt.show()

p_name = ['p1', 'p2', 'p3', 'p4', 'p5', 'p6', 'p7']
for i in range(7):
    plt.plot(p_[i])
    plt.xlabel('迭代次数')
    plt.ylabel("拉格朗日系数" + p_name[i])
    plt.title("拉格朗日系数" + p_name[i] + "变化曲线")
    plt.savefig("拉格朗日系数" + p_name[i] + "变化曲线.png")
    plt.show()

m_name = ['m1', 'm2', 'm3', 'm4']
for i in range(4):
    plt.plot(m_[i])
    plt.xlabel('迭代次数')
    plt.ylabel("拉格朗日系数" + m_name[i])
    plt.title("拉格朗日系数" + m_name[i] + "变化曲线")
    plt.savefig("拉格朗日系数" + m_name[i] + "变化曲线.png")
    plt.show()
```

四、实验结果分析

在这一部分，我们通过运行程序获得了所构造算例的迭代曲线图以及最终收敛的效用函数值$U_s(x_s)$，信号源速率x_s和拉格朗日参数p_l&μ_s。同时我们通过改变参数的设置，对不同参数下的实验结果进行了讨论。

4.1 实验结果与分析

通过运行上一部分给出的程序代码，我们可以获得所设计算法在该算例的迭代过程和结果。为了获得一个较好的效果，我们尝试了若干种参数初始值设定，最终在代码中设置了如下参数的初始值：

拉格朗日系数$p_l(0)$	5.3	各链路最大速率$c_l(0)$	4.0
拉格朗日系数$\mu_s(0)$	2.1	端到端时延限制$D_s(0)$	10
各链路裕度变量$\sigma_l(0)$	0.1	迭代步长（$\beta = \gamma$）	0.005

最终我们获得的迭代曲线为：

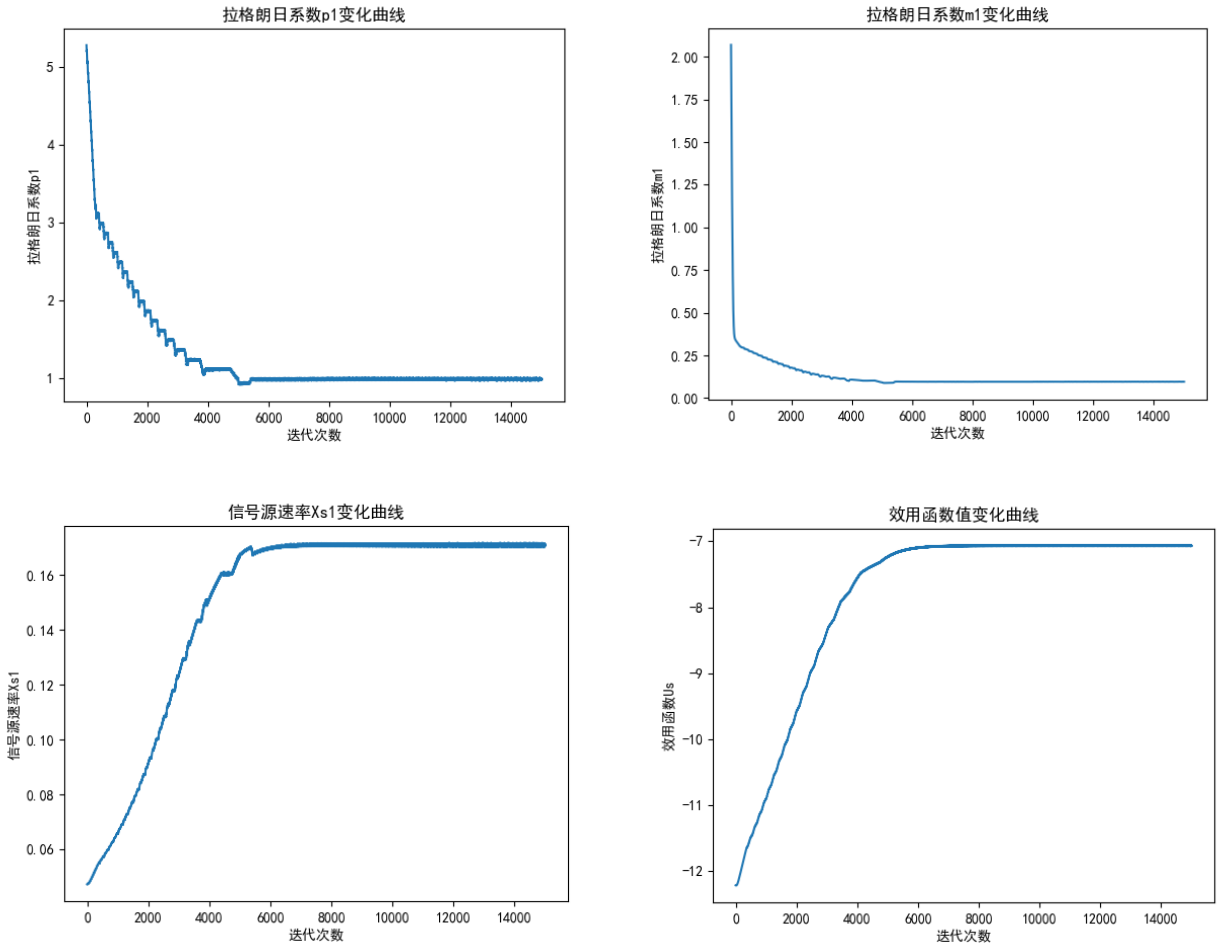

图 2 各参数及变量收敛曲线

对于该算例，一共有 7 条链路和 4 条流，所以会有 7 个拉格朗日系数p_l的图，4 个拉格朗日系数μ_s的图以及 4 个信号源速率X_s的图。我们选取了每种变量的第一个图进行展示，其余图片的形状和所展示的图类似，已在附件中给出。

从上面的迭代图中可以看出，最终拉格朗日系数，信号源速率和效用函数值最终均收敛，符合第二问给出的理论上的收敛性。理论上，我们的目标是使效用函数取得最大值，而在效用函数的曲线中，可以看出效用函数值随迭代次数增加在不断地上升，最终收敛到一个值，这和理论结果一致。上述的图片和分析验证了我们所构造的算法的有效性。

4.2 初始值变化的结果讨论

在 4.1 节中，我们发现该算法对初始条件很敏感，即不同的初始值设置对迭代的结果影响很大。所以我们通过改变初始值的设定来探究初始值对算法的影响。

我们考虑从拉格朗日系数，链路裕度，链路最大速率，端到端时延限制和迭代步长这几个初始值设置的改变导致的结果中来讨论。所有的初始值都是基于 4.1 中得到最终结果的初始值来改变的，同时为了控制变量，在一种初始值改变时另外的初始值保持不变。另外，我们只选择了效用函数曲线进行展示，其余图片在附件中给出。通过实验，我们得到的结果如下：

（1）拉格朗日系数p_l& μ_s初始值改变

我们选择将两组数据① $p_l(0)=0.53, \mu_s(0)=0.21$ ② $p_l(0)=53, \mu_s(0)=21$，得到结果分别在下面左图和右图中展示：

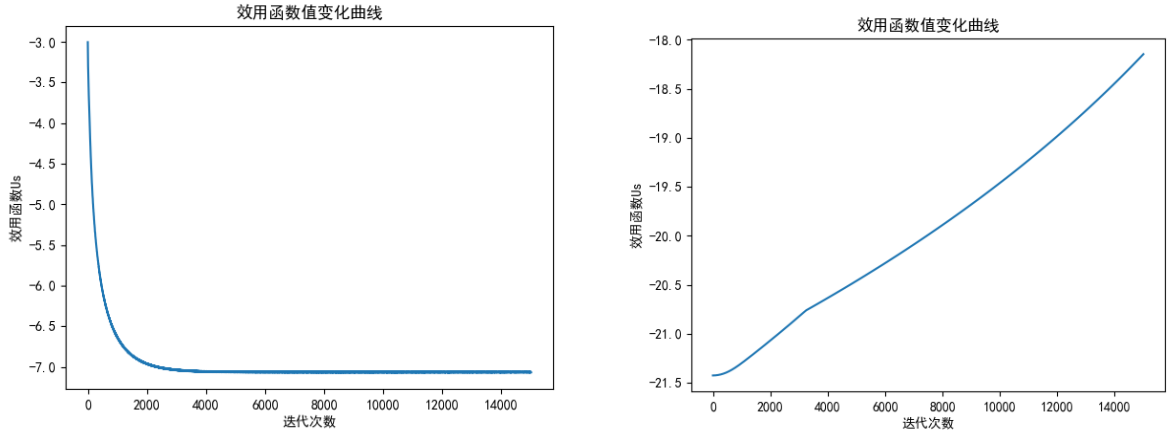

图 3 拉格朗日系数p_l& μ_s初始值变化时，效用函数曲线

通过上面的曲线图，可以看出拉格朗日系数初始值设置过低会导致算法失效，过高会让算法收敛变慢，或者不收敛。

（2）链路裕度σ_l初始值改变

我们选择将两组数据① $\sigma_l(0)=0.01$ ② $\sigma_l(0)=1$，得到结果分别在下面左图和右图中展示：

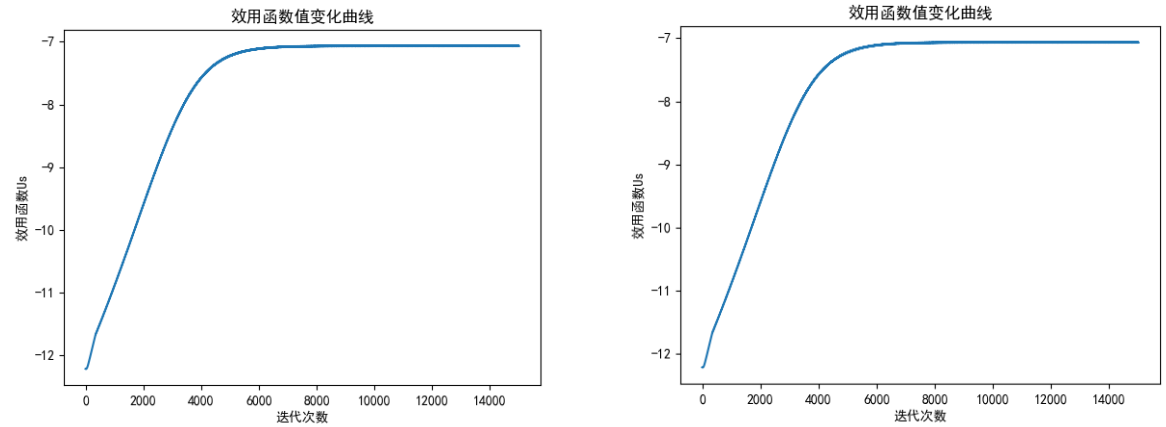

图 4 链路裕度σ_l初始值变化时，效用函数曲线

通过上面的曲线图，可以看出链路裕度初始值设置对结果影响不大。

（3）链路最大速率c_l初始值改变

我们选择将两组数据① $c_l(0)=2$ ② $c_l(0)=10$，得到结果分别在下面左图和右图中展示：

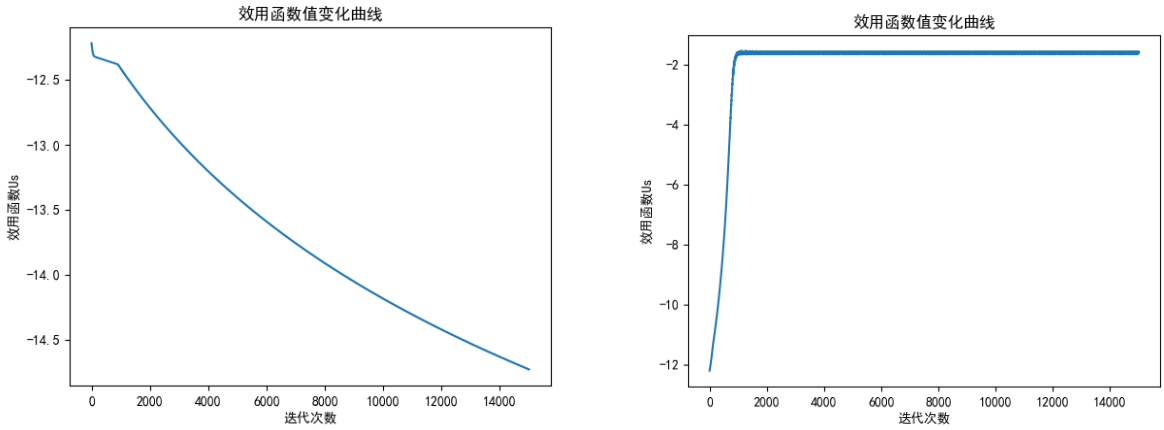

图 5 链路最大速率c_l初始值变化时，效用函数曲线

通过上面的曲线图，可以看出链路最大速率初始值设置过低会使得算法失效。

（4）端到端时延限制D_s初始值改变

我们选择将两组数据① $D_s(0)=5$ ② $D_s(0)=15$，得到结果分别在下面左图和右图中展示：

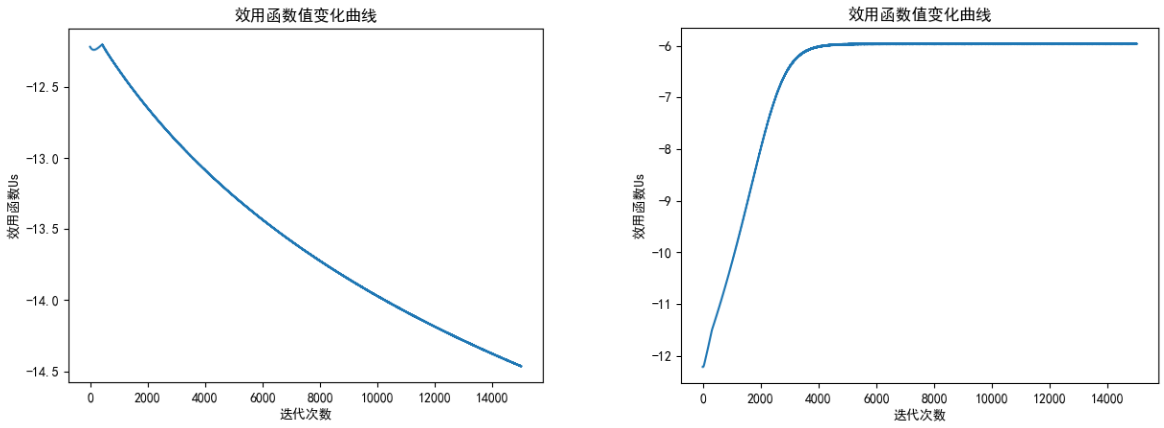

图 6 端到端时延限制D_s初始值变化时，效用函数曲线

通过上面的曲线图，可以看出端到端时延限制初始值设置过低会使得算法失效。

（5）迭代步长β&γ初始值改变

我们选择将两组数据① $\beta=\gamma=0.0005$ ② $\beta=\gamma=0.05$，得到结果分别在下面左图和右图中展示：

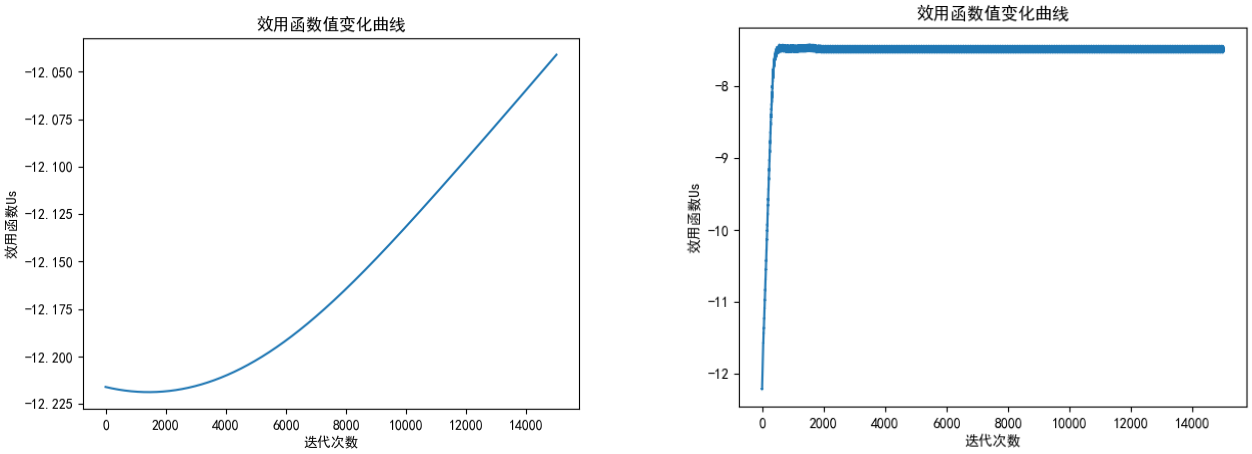

图 7 迭代步长β&γ变化时，效用函数曲线

通过上面的曲线图，可以看出迭代步长初始值设置过低会使得收敛变慢，过高会使得收敛后有振荡。

综上所述，初始值的设置对该算法有较大的影响。拉格朗日系数初始值设置过低，链路最大速率初始值设置过低，端到端时延限制初始值设置过低会使得算法失效；拉格朗日系数初始值设置过高会使得算法不收敛；迭代步长初始值设置过低会使得收敛变慢，而设置过高会使得收敛后有振荡。

上述讨论证明了该算法对初始值设置比较敏感，但是通过合理设置初始值，可以得到 4.1 中较好的结果，也就是说我们所构造的算法的有效性仍然得到满足。

4.3 实时调控速率验证

如 3.3 第一节定性分析所说，当端到端时延μ_s增大时，信源速率x_s会减小。现将这两者负相关曲线展示如下：

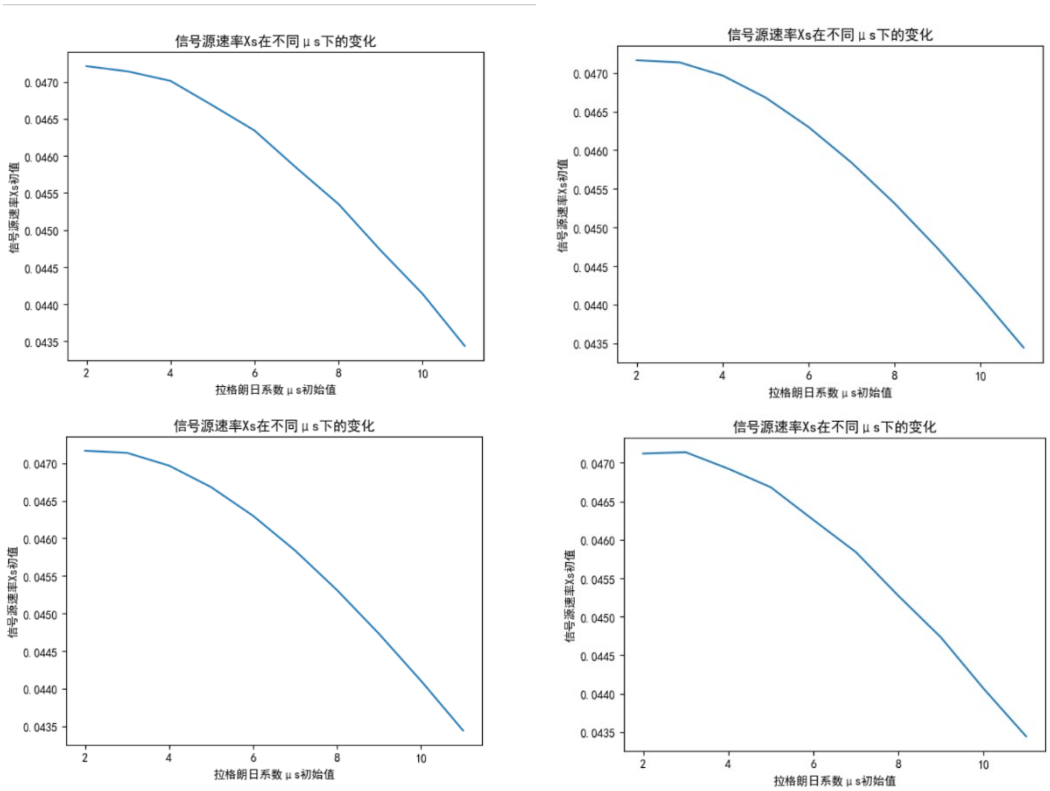

图 8 μ_s和$x_s, s \in \{1,2,3,4\}$负相关曲线

五、讨论与思考

不难发现，该分布式算法虽然可以取得较理想的效果，但也存在一定的局限性：
①需要初始化的参数较多，有很大随机性；
②算法对初始化参数的变化比较敏感；
③对于存在较多节点和链路的复杂拓扑结构，收敛可能较慢。

该分布式算法可以改进的地方：
①设置经验参数，便于初始化；
②做出更多假设，进一步简化或者合并子模型。

反思：
①在更复杂的信息流冲突图下，可以采用贪心算法来近似求得使$p_l c_l$最大的链路l；
②正确理解参数的物理意义，对调整参数，加速收敛具有重要意义。

致谢

由衷感谢认真负责的杨博老师和陈彩莲老师对本次大作业前期理论知识的细致讲解，对大作业的耐心介绍和答疑，也感谢张景龙助教对配置仿真环境及大作业过程中种种问题的耐心细致的解答！《网络系统与控制》是一门让身为本科的我们提前接触研究生内容的课程，让我们进一步了解了控制理论在网络通信中的运用；同时，本次大作业也培养了我们合作完成项目的能力。祝愿《网络系统与控制》可以越来越好！