Stochastic User Equilibrium: Method of Successive Averages

AZAREL CHAMORRO OBRA

M1福田研究室

Stochastic User Equilibrium

Assignment based on perceived cost (eg: travel time)

Travel Time is flow-dependent. Link performance function

Equilibrium: No user can minimize their travel time by unilaterally changing their path.

$$f_k^{rs} = q_{rs} P_k^{rs}, \forall\, k, r, s \quad \left\{ \begin{array}{ll} t_a = t_a(x_a) & \forall\, a \\ \sum_k f_k^{rs} = q_{rs} & \forall\, r, s \end{array} \right.$$

Method of Successive Averages

Equilibrium Optimization problem: move size α

A descent vector for search direction is always found.

Forced algorithm: it will always converge

$$\left\{ \begin{aligned} &\sum_{n=1}^{\infty} \alpha_n = \infty \\ &\sum_{n=1}^{\infty} \alpha_n^2 < \infty \end{aligned} \right.$$

$$\alpha_n = \frac{1}{n};\ x^{n+1} = x^n + \frac{1}{n} d^n = x^n + \frac{1}{n}(y^n - x^n)$$

Therefore, solution at iteration n is the **average of the variables y in preceding iterations**.

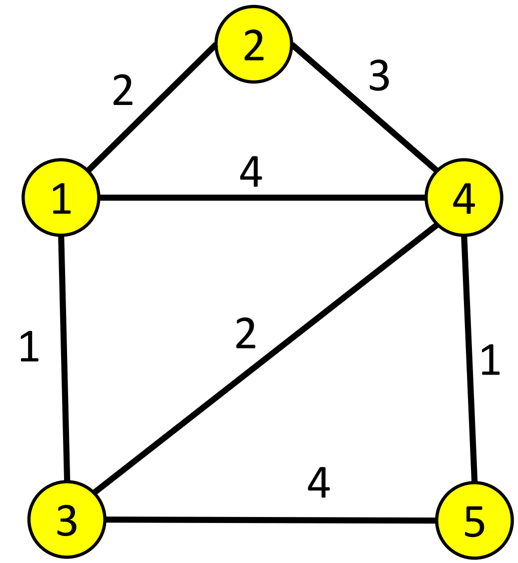
Link performance function

Network loading models:

- Travel time is constant

Stochastic User Equilibrium:

- Travel time depends on flow
- User Equilibrium conditions are particular case



Link performance function

Network loading models:

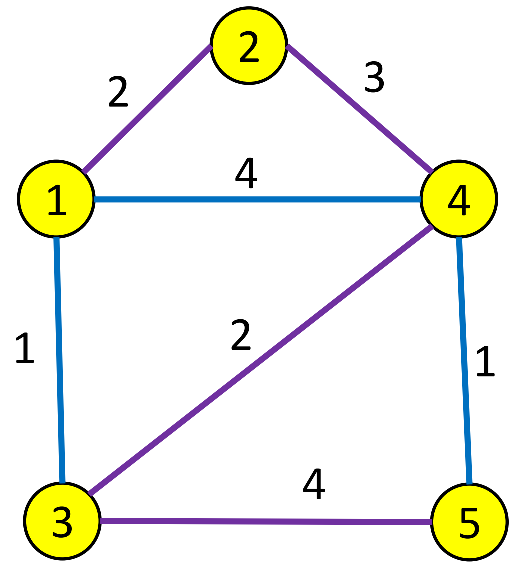
- Travel time is constant

Stochastic User Equilibrium:

- Travel time depends on flow
- User Equilibrium conditions are particular case

$$T_{blue} = To + \frac{x}{3000}^{5}$$

$$T_{purple} = To + \frac{x}{2000}^{5}$$



MSA: Algorithm

0) Preliminaries: Stochastic network loading considering initial travel times (free flow speed) t_a^0. Link flows x_a^1 are calculated.

1) Considering link function, calculate new travel times $t_a^n = t_a(x_a^n), \forall a$

2) Direction finding: New auxiliary link flows y_a^n are calculated according to the new travel times t_a^n.

3)Movement: $x_a^{n+1} = x_a^n + \frac{1}{n}(y_a^n - x_a^n)$

4) Convergence: if convergence is achieved algorithm is stopped. Else next iteration n.

Convergence

Step size is always reduced at iterating: “forcing convergence”.

Flow average over last m iterations is a reliable parameter.

$$\frac{\sqrt{\sum_a (\bar{x}_a^{n+1} - \bar{x}_a^n)^2}}{\sum_a \bar{x}_a^n} \leq \kappa$$

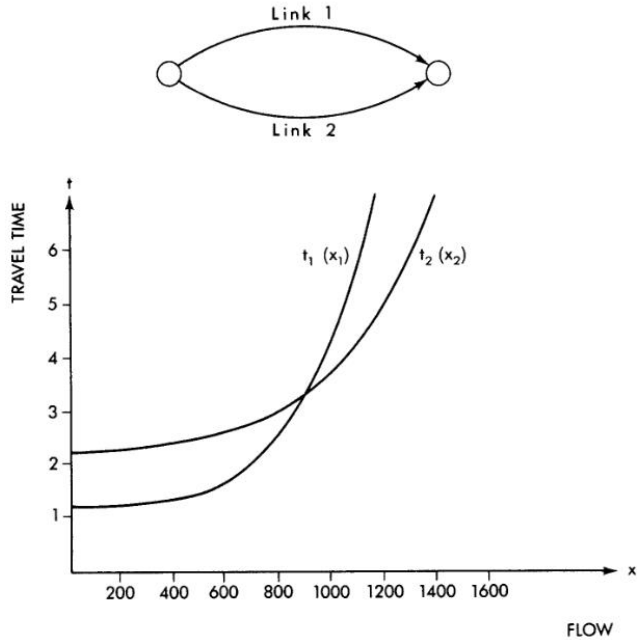


Figure 12.4 Network example with two paths connecting a single O–D pair.

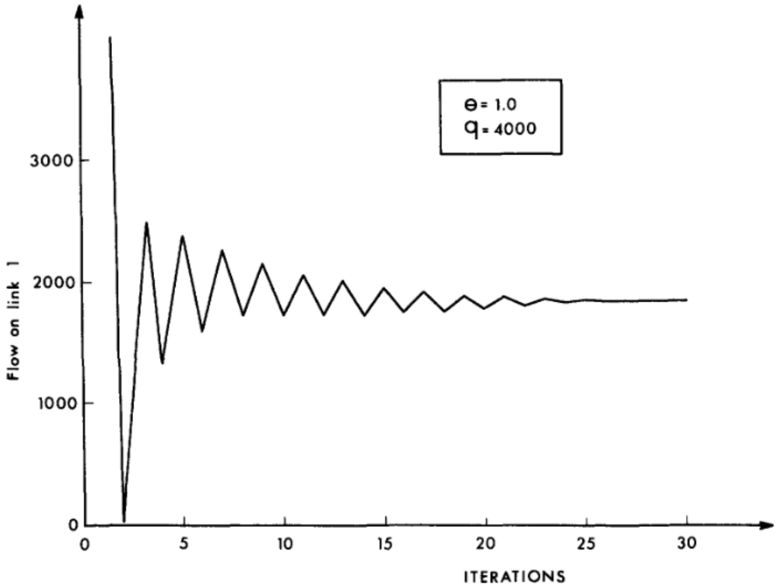


Figure 12.5 Convergence pattern of the MSA for the network in Figure 12.4; the case of relatively large perception variance ($\theta = 1.0$) and relatively high congestion level ($q = 4000$).

MSA: Algorithm

```
iteration = 1
param = 10
x_n = [row[:] for row in lf]
temp = [row[:] for row in x_n]
criterion = 0.05
convergence = 1
m_n = [row[:] for row in m]

fo=[
[0,(lf[0][1]/2000)**5,(lf[0][2]/3000)**5,(lf[0][3]/3000)**5,0],   # A (0)
[(lf[1][0]/2000)**5,0,0,(lf[1][3]/2000)**5,0],   # B (1)
[(lf[2][0]/3000)**5,0,0,(lf[2][3]/2000)**5,(lf[2][4]/2000)**5],   # C (2)
[(lf[3][0]/3000)**5,(lf[3][1]/2000)**5,(lf[3][2]/2000)**5,0,(lf[3][4]/3000)**5],   # D (3)
[0,0,(lf[4][2]/2000)**5,(lf[4][3]/3000)**5,0],   # E (4)
]


#Preliminary: Copy m

while convergence>criterion and iteration < 100:
#Step 1: Set m_n a function of flow

    for i in range(0, len(m_n)):
        for j in range(0,len(m_n[0])):
            m_n[i][j] = (m[i][j] + fo[i][j])

```

lf as Link Flow Matrix

fo can be redefined

MSA: Algorithm

```
#Step 2: Perform stochastic network flow algorithm on m_n for link flow matrix y_n
    y_n = link_flow(m_n)

    fo=[
    [0,(y_n[0][1]/2000)**5,(y_n[0][2]/3000)**5,(y_n[0][3]/3000)**5,0],   # A (0)
    [(y_n[1][0]/2000)**5,0,0,(y_n[1][3]/2000)**5,0],   # B (1)
    [(y_n[2][0]/3000)**5,0,0,(y_n[2][3]/2000)**5,(y_n[2][4]/2000)**5],   # C (2)
    [(y_n[3][0]/3000)**5,(y_n[3][1]/2000)**5,(y_n[3][2]/2000)**5,0,(y_n[3][4]/3000)**5],   # D (3)
    [0,0,(y_n[4][2]/2000)**5,(y_n[4][3]/3000)**5,0],   # E (4)
    ]

#Step 3: Set x_n = x_n + (1/n) * (y_n - x_n)
    for i in range(0, len(x_n)):
        for j in range(0,len(x_n[0])):
            temp[i][j] = x_n[i][j]
    for i in range(0, len(m_n)):
        for j in range(0,len(m_n[0])):
            x_n[i][j] = x_n[i][j] + (1 / iteration) * (y_n[i][j] - x_n[i][j])
    iteration = iteration + 1


print('MSA result for iteration ' + str(iteration))
print(x_n)
```