High Performance Traffic Assignment Based on Variational Inequality…

XIAO Yujie, SHI Zhenmei
Mentor: Dr. LIU Cheng, Dr. WONG Kwai

Agenda

- Introduction
  - Traffic Assignment Problem
  - Variational Inequality
- STA
- DTA

Joint Institute for Computational Sciences

Traffic Assignment

Traffic assignment is a kernel component in transportation planning and real-time applications in optimal routing, signal control, and traffic prediction in traffic networks.

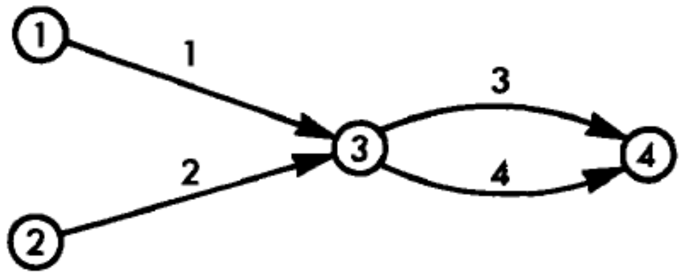
Introduction

Traffic Assignment Problem

Node

Link

Origin-Destination Pair



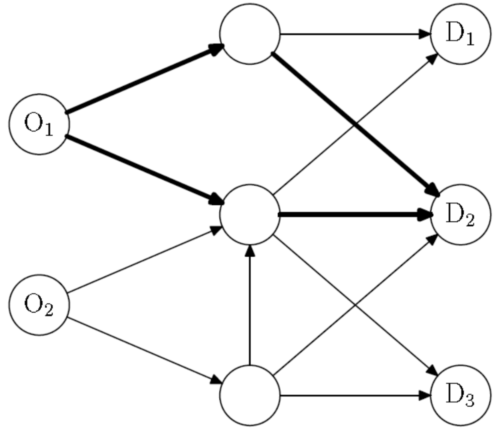


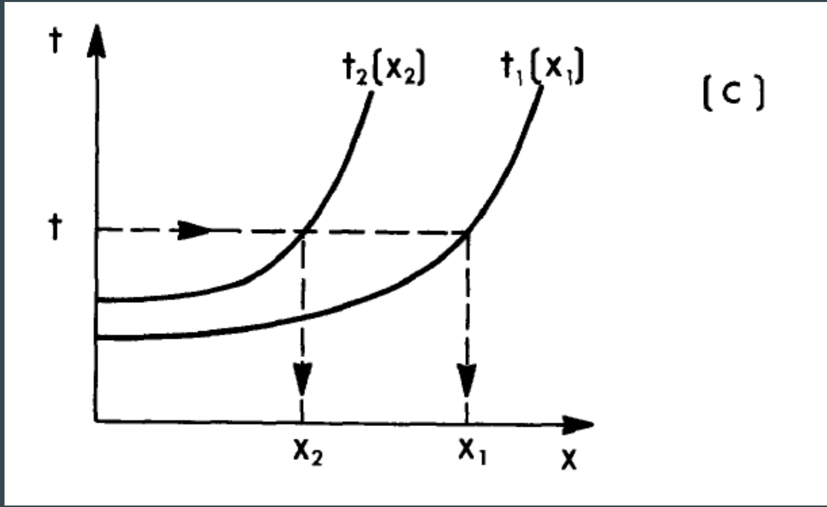
Figure 1.5: An illustration of the traffic equilibrium problem.

Time Cost

Traffic Assignment Problem

Optimization

- System equilibrium
- User equilibrium



Time Cost Function

$$time = freeflowtime * (1 + B * (flow/capacity)^{Power})$$

Traffic Assignment Problem

Given:

1. A graph representation of the urban transportation network

2. The associated link performance functions

3. An origin-destination matrix

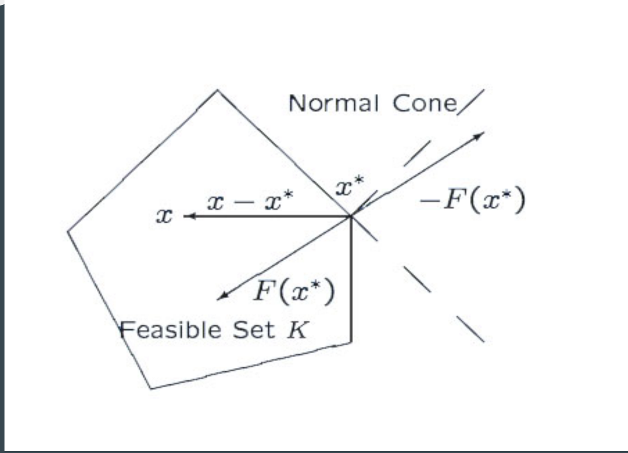
Find the flow (and travel time) on each of the network links, such that the network satisfies user-equilibrium （UE） principle.

Variational Inequality

- What?
  - Definition

$$(y-x)^T F(x) \geq 0, \ \forall y \in K$$

  - Graphically



Variational Inequality

❖ Category

$$
\begin{array}{ccccc}
\text{VI } (K, q, M) & & & & \text{VI } (K, q, M) \\
\Uparrow & & & & \Downarrow \\
\text{VI } (K, F) & \Rightarrow & \text{linearly constrained VI} & \Rightarrow & \text{AVI } (K, q, M) \\
\Downarrow & & \Updownarrow & & \Updownarrow \\
\text{CP } (K, F) & \Rightarrow & \text{MiCP } (F) & \Rightarrow & \text{MLCP} \\
 & & \Downarrow & & \Downarrow \\
 & & \text{NCP } (F) & \Rightarrow & \text{LCP } (q, M).
\end{array}
$$

Variational Inequality

- Why?
  - Intuitive: Either scenorio A or scenorio B
  - closely related to equilibrium
- Application
  - Nash Equilibrium Problem
  - Economic Equilibrium Problem
  - Pricing America Options

Traffic Assignment Problem

- Category
  - Static Traffic Assignment
  - Dynamic Traffic Assignment (continuous or discrete)

STA

VI on Static Traffic Assignment Problem (STA)

Frank

Wolfe

Algorithm

Step 0: *Initialization.* Perform all-or-nothing assignment based on $t_a = t_a(0)$, $\forall\ a$. This yields $\{x_a^1\}$. Set counter $n := 1$.

Step 1: *Update.* Set $t_a^n = t_a(x_a^n)$, $\forall\ a$.

Step 2: *Direction finding.* Perform all-or-nothing assignment based on $\{t_a^n\}$. This yields a set of (auxiliary) flows $\{y_a^n\}$.

Step 3: *Line search.* Find α_n that solves

$$\min_{0 \leqslant \alpha \leqslant 1} \sum_a \int_0^{x_a^n + \alpha(y_a^n - x_a^n)} t_a(\omega)\, d\omega$$

Step 4: *Move.* Set $x_a^{n+1} = x_a^n + \alpha_n(y_a^n - x_a^n)$, $\forall\ a$.

Step 5: *Convergence test.* If a convergence criterion is met, stop (the current solution, $\{x_a^{n+1}\}$, is the set of equilibrium link flows); otherwise, set $n := n + 1$ and go to step 1.

VI on Static Traffic Assignment Problem (STA)

Nonlinear Complementarity Problem (NCP)

1.1.5 Definition. Given a mapping $F : \mathbb{R}^n_+ \to \mathbb{R}^n$, the NCP (F) is to find a vector $x \in \mathbb{R}^n$ satisfying

$$0 \leq x \perp F(x) \geq 0. \tag{1.1.5}$$

Joint Institute for Computational Sciences

Computational Sciences

VI on Static Traffic Assignment Problem (STA)

$$\sum_{k \in R_w} f_k^w = q_w,$$

$$C_k^w = \sum_{a \in A} \delta_{ak}^w t_a(x),$$

$$x_a = \sum_{w \in W} \sum_{k \in R_w} \delta_{ak}^w f_k^w,$$

$$u_w \geq 0.$$

$$0 \leq C_p(h) - u_w \perp h_p \geq 0, \quad \forall w \in \mathcal{W} \text{ and } p \in \mathcal{P}_w;$$

$$\sum_{p \in \mathcal{P}_w} h_p = d_w(u), \quad \forall w \in \mathcal{W},$$

$$u_w \geq 0, \quad w \in \mathcal{W}.$$

$$\mathbf{F}(h,u) \equiv \begin{pmatrix} C(h) - \Omega^T u \\ \Omega h - d(u) \end{pmatrix},$$

Traffic Problem

complementarity problem

Nonlinear

VI on Static Traffic Assignment Problem (STA)

- ❖ Limitation
  - ➢ Unrealistic to find all path for a big graph

$$
\begin{array}{c|c}
\text{link} \backslash \text{path} & \overbrace{\begin{array}{cc} 1 & 2 \end{array}}^{\text{O–D } 1\text{–}4} \;\; \overbrace{\begin{array}{cc} 1 & 2 \end{array}}^{\text{O–D } 2\text{–}4} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}
\end{array}
$$

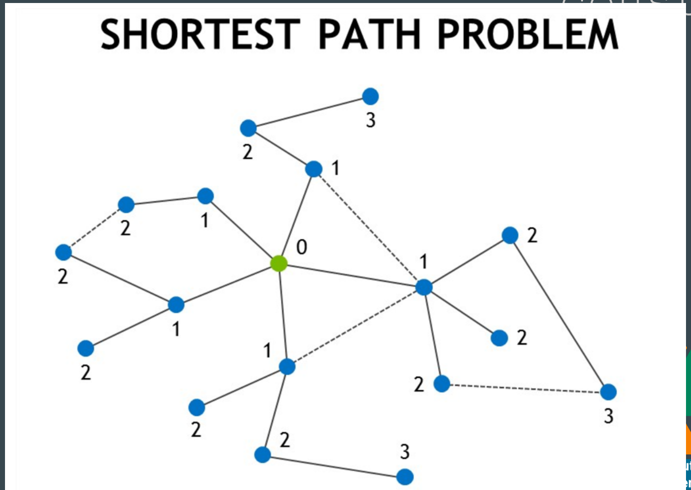
VI on Static Traffic Assignment Problem (STA)

- Solution
  - Find 7 nonsimilar path for each OD-pair to reduce Matrix size
  - Use Shortest Path Algorethm
  - Get approximate Optimization

Algorithm

Step 1: Use One to All shortest path algorithm to find 7 paths for each OD pair. Here the solver uses nvGRAPH package in CUDA library which runs on GPU.

```
nvgraphStatus_t nvgraphSssp (nvgraphHandle_t,const
nvgraphGraphDescr_t , const size_t, const
int *, const size_t);
```



Algorithm

Step 2: Convert all data in to NCP formulation in Siconos, which is a non-smooth numerical simulation package

$$\mathbf{A}_{sparse} = \begin{bmatrix} 0 & A_{12} & A_{13} & 0 & 0 \\ 0 & A_{22} & 0 & 0 & 0 \\ 0 & 0 & A_{33} & 0 & 0 \\ 0 & 0 & A_{43} & A_{44} & 0 \\ 0 & A_{52} & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Algorithm

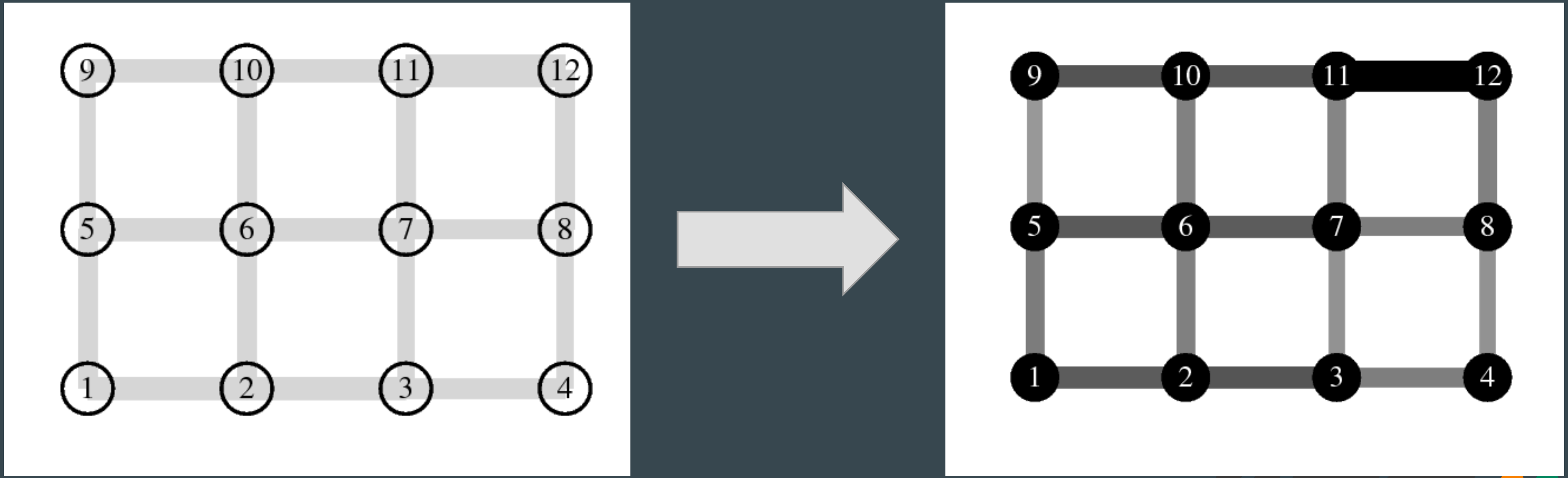
Step 3: Use NCP FBLSA Algorithm to solve the problem with given error bound. Here the solver uses Siconos and MUMPS library, which is a parallel sparse direct solver using MPI.

```
info = ncp_driver(problem, z, F, &options);
```

Sample input

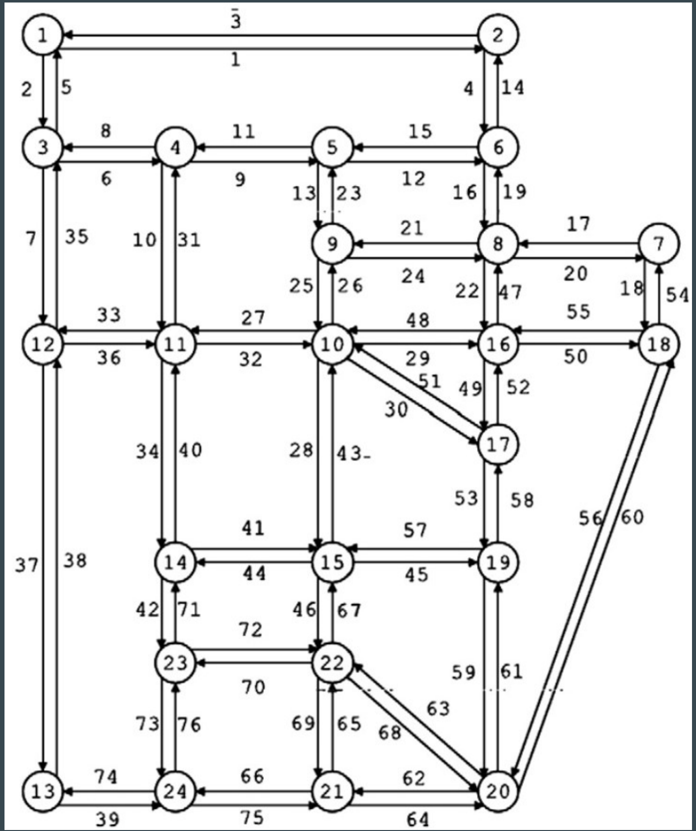
```
<LINKS>
~ Init node | Term node | Capacity | Length | Free Flow Time | B | Power | Speed limit | Toll | Type
    1      2  25900.200640  6.000000  6.000000  0.150000  4.000000  0.000000  0.000000  1
    1      3  23403.473190  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    2      1  25900.200640  6.000000  6.000000  0.150000  4.000000  0.000000  0.000000  1
    2      6  4958.180928  5.000000  5.000000  0.150000  4.000000  0.000000  0.000000  1
    3      1  23403.473190  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    3      4  17110.523720  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    3     12  23403.473190  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    4      3  17110.523720  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    4      5  17782.794100  2.000000  2.000000  0.150000  4.000000  0.000000  0.000000  1
    4     11  4908.826730  6.000000  6.000000  0.150000  4.000000  0.000000  0.000000  1
    5      4  17782.794100  2.000000  2.000000  0.150000  4.000000  0.000000  0.000000  1
    5      6  4947.995469  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    5      9  10000.000000  5.000000  5.000000  0.150000  4.000000  0.000000  0.000000  1
    6      2  4958.180928  5.000000  5.000000  0.150000  4.000000  0.000000  0.000000  1
    6      5  4947.995469  4.000000  4.000000  0.150000  4.000000  0.000000  0.000000  1
    6      8  4898.587646  2.000000  2.000000  0.150000  4.000000  0.000000  0.000000  1
    7      8  7841.811310  3.000000  3.000000  0.150000  4.000000  0.000000  0.000000  1
    7     18  23403.473190  2.000000  2.000000  0.150000  4.000000  0.000000  0.000000  1
    8      6  4898.587646  2.000000  2.000000  0.150000  4.000000  0.000000  0.000000  1
    8      7  7841.811310  3.000000  3.000000  0.150000  4.000000  0.000000  0.000000  1
    8      9  5050.193156  10.000000  10.000000  0.150000  4.000000  0.000000  0.000000  1
    8     16  5045.822583  5.000000  5.000000  0.150000  4.000000  0.000000  0.000000  1
Origin   1
    1 :      0.0;     2 :    100.0;     3 :    100.0;     4 :    500.0;     5 :    200.0;
    6 :    300.0;     7 :    500.0;     8 :    800.0;     9 :    500.0;    10 :   1300.0;
   11 :    500.0;    12 :    200.0;    13 :    500.0;    14 :    300.0;    15 :    500.0;
   16 :    500.0;    17 :    400.0;    18 :    100.0;    19 :    300.0;    20 :    300.0;
   21 :    100.0;    22 :    400.0;    23 :    300.0;    24 :    100.0;
```

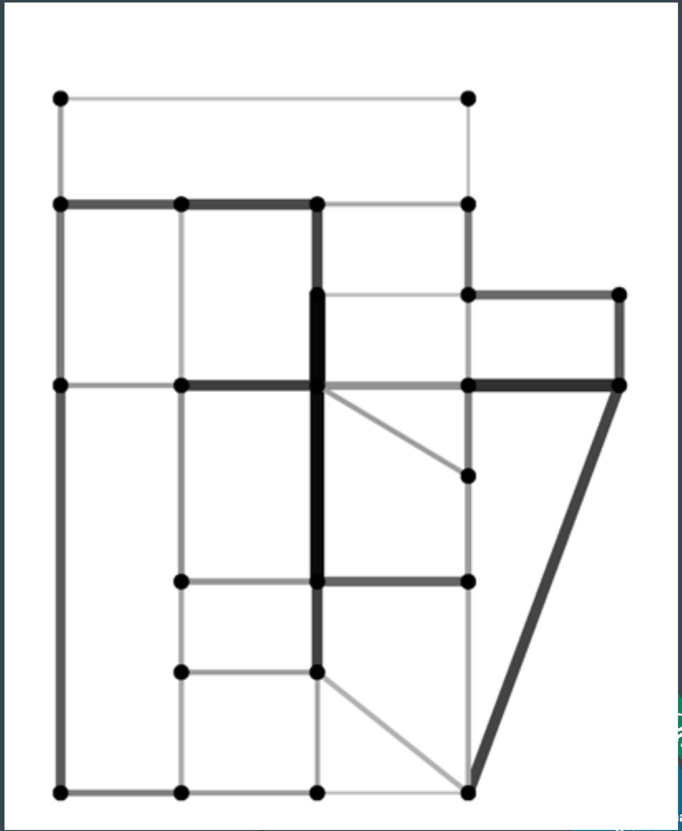
Result with 4 OD Pair



Joint Institute for Computational Sciences

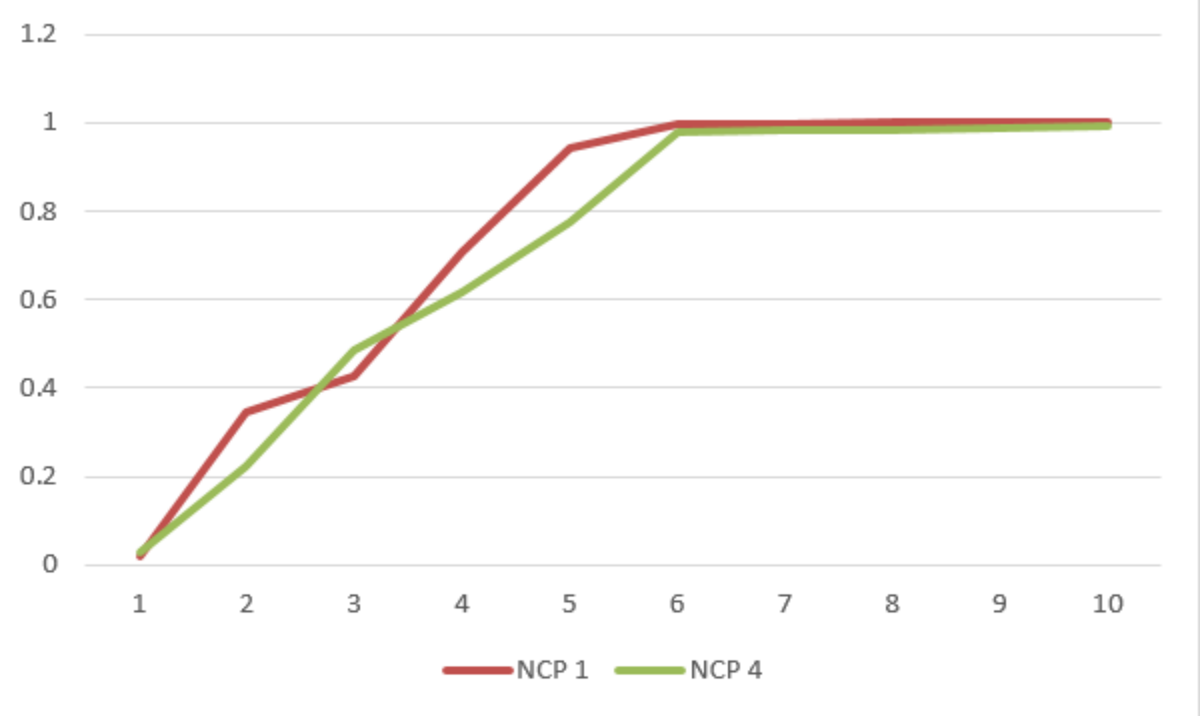
Result with 24 OD Pair





Result

The accuracy varies with path number for each OD pair



Analysis - Compared with Frank Wolfe Algorithm

NCP:

1. Dominant cost: Matrix solver
2. Approximate optimize
3. A little faster when graph is big and with a few OD pair (Matrix size is OD pair number + path number)

FW:

1. Dominant cost: shortest path algorithm
2. Real Optimize
3. Faster when OD pair is more

Conclusion

1. Frank Wolfe Algorithm is still better than NCP Algorithm in general.

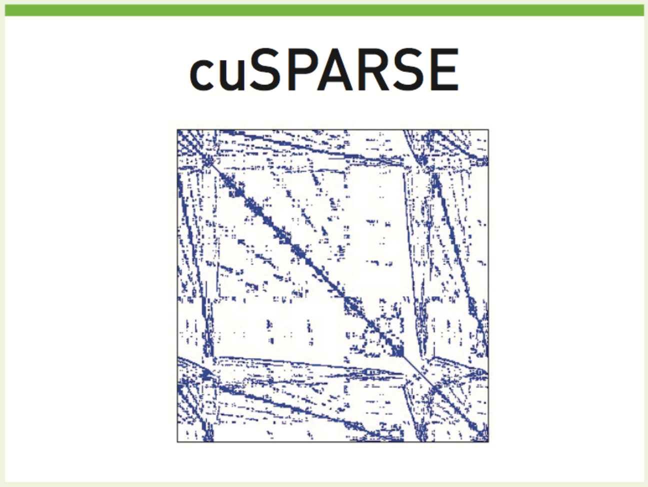
2. In special cases, when graph is big and number of OD - Pair is little NCP Algorithm is faster than Frank Wolfe Algorithm.

3. When select 7 paths for each OD - pair in NCP algorithm, the result accuracy can reach 95%.

Future Work

- Do comprehensive tests

- alcul



DTA

Mathematical Formulation

➢ Variational Inequality formulation:

- Nash equilibrium nature

$$h_p(t) > 0 \Rightarrow C_p(t, h) = \mu_{kl} \quad \forall_\nu(t)$$

$$C_p(t, h) \geqslant \mu_{kl} \quad \forall_\nu(t).$$

$$\left.\begin{array}{c} \text{find } h^* \in \Lambda \text{ such that} \\ \sum_{p \in \mathcal{P}} \int_{t_0}^{t_f} \Psi_p(t, h^*)\left(h_p - h_p^*\right) dt \geqslant 0 \\ \forall h \in \Lambda \end{array}\right\} DVI(\Psi, \Lambda, [t_0, t_f])$$

Joint Institute for Computational Sciences

Mathematical Formulation

- ➢ Dynamic Network Loading:
  - Given h, return path delay operator
  - Approximated by ODE systems

$$\frac{dx^p_{a_i}(t)}{dt} = g^p_{a_{i-1}}(t) - g^p_{a_i}(t) \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$x^p_{a_i}(0) = x^{p,0}_{a_i} \in \Re^1_+ \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$h^{\tau,k}_p(t) = g_{a_1}(t + D_{a_1}[x_{a_1}(t)])\left(1 + D'_{a_1}[x_{a_1}(t)]\dot{x}_{a_1}\right)$$

$$g^p_{a_{i-1}}(t) = g^p_{a_i}(t + D_{a_i}[x_{a_i}(t)])\left(1 + D'_{a_i}[x_{a_i}(t)]\dot{x}_{a_i}(t)\right) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]$$

$$\frac{dx^p_{a_1}(t)}{dt} = h^{\tau,k}_p(t) - g^p_{a_1}(t) \quad \forall p \in \mathcal{P}$$

$$\frac{dx^p_{a_i}(t)}{dt} = g^p_{a_{i-1}}(t) - g^p_{a_i}(t) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]$$

$$\frac{dg^p_{a_i}(t)}{dt} = r^p_{a_i}(t) \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$\frac{dr^p_{a_1}(t)}{dt} = R^p_{a_1}(x, g, r, h^{\tau,k}) \quad \forall p \in \mathcal{P}$$

$$\frac{dr^p_{a_i}(t)}{dt} = R^p_{a_i}(x, g, r) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]$$

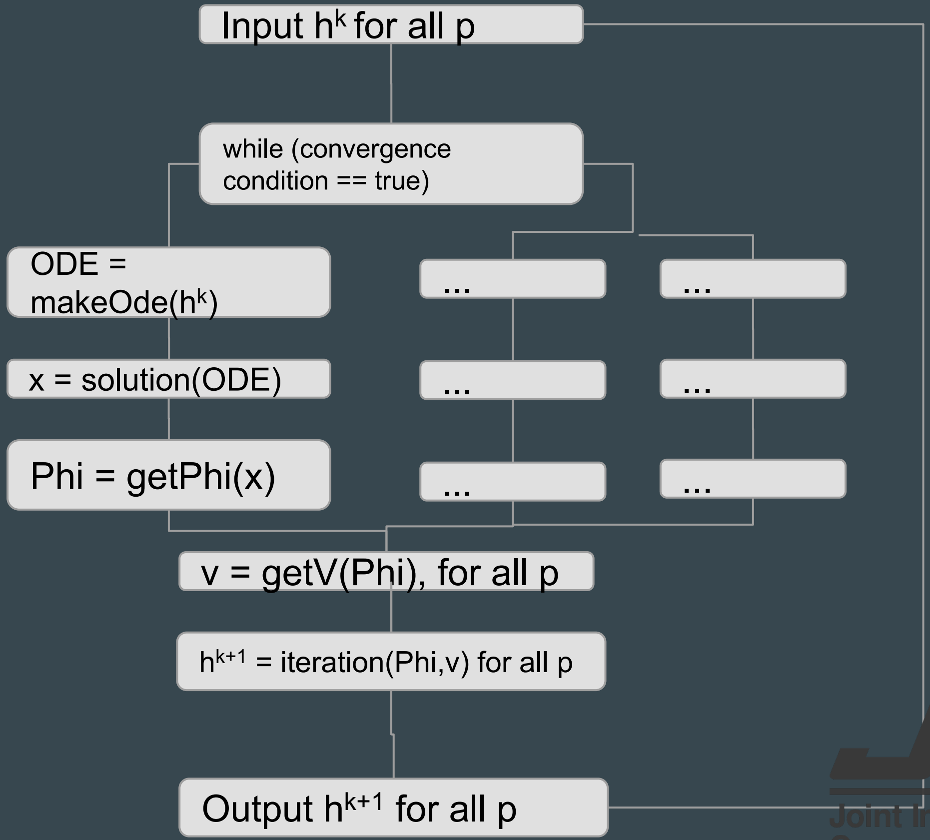
$$x^p_{a_i}((\tau - 1) \cdot \Delta) = x^{p,0}_{a_i} \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$g^p_{a_i}((\tau - 1) \cdot \Delta) = 0 \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$r^p_{a_i}((\tau - 1) \cdot \Delta) = 0 \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

DTA: Algorithm

- ➢ Overview



DTA: Algorithm

- ODE = make_ODE(h)

$$\frac{dx_{a_1}^p(t)}{dt} = h_p^{\tau,k}(t) - g_{a_1}^p(t) \quad \forall p \in \mathcal{P}$$

$$\frac{dx_{a_i}^p(t)}{dt} = g_{a_{i-1}}^p(t) - g_{a_i}^p(t) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]$$

$$\frac{dg_{a_i}^p(t)}{dt} = r_{a_i}^p(t) \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$\frac{dr_{a_1}^p(t)}{dt} = R_{a_1}^p(x, g, r, h^{\tau,k}) \quad \forall p \in \mathcal{P}$$

$$\frac{dr_{a_i}^p(t)}{dt} = R_{a_i}^p(x, g, r) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]$$

$$x_{a_i}^p((\tau - 1) \cdot \Delta) = x_{a_i}^{p,0} \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$g_{a_i}^p((\tau - 1) \cdot \Delta) = 0 \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

$$r_{a_i}^p((\tau - 1) \cdot \Delta) = 0 \quad \forall p \in \mathcal{P}, i \in [1, m(p)]$$

- x = solution(ODE)

DTA: Algorithm

➢ Dp = getDp(x)

- x: arc volume
- Dp: traversal time
- Phi: cost function

$$D_p = \sum_{i=1}^{m(p)} [\tau_{a_i}^p(t) - \tau_{a_{i-1}}^p(t)] = \tau_{a_{m(p)}}^p(t) - t$$
$$\tau_{a_1}^p(t) = t + D_{a_1}[x_{a_1}(t)]$$
$$\tau_{a_i}^p(t) = \tau_{a_{i-1}}^p(t) + D_{a_i}[x_{a_i}(\tau_{a_{i-1}}^p(t))]$$
$$D(x) = \alpha * x + \beta$$

$$\Phi_p(t) = D_p(t) + F[D_p(t) + t - T_A]$$
$$F(D_p(t) + t - T_A) = 0.5 * (D_p(t) + t - T_A)^2$$

➢ Phi = getPhi(Dp)

- F: penalty function

DTA: Algorithm

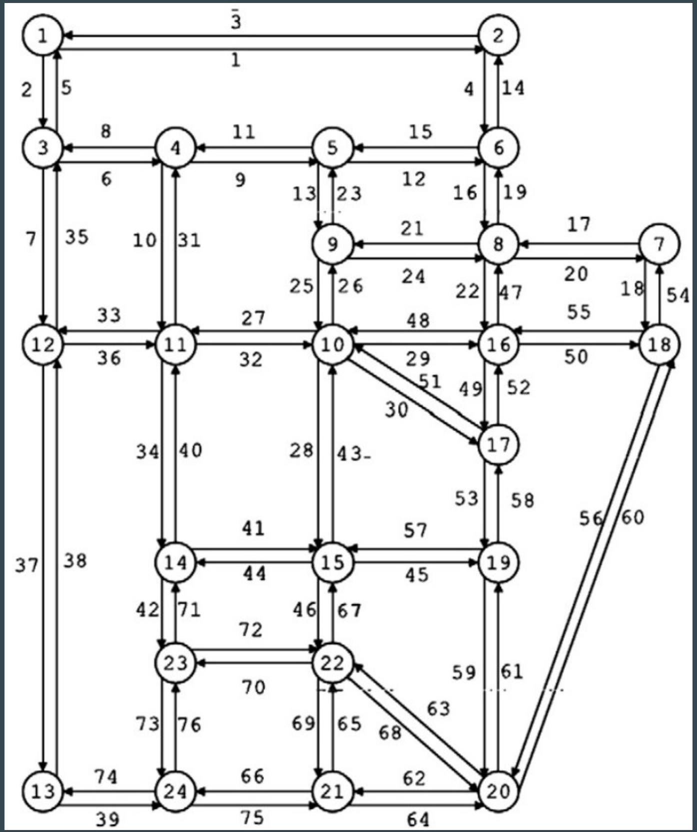
- v = solution(Phi)

$$\sum_{p \in P_{ij}} \int_{t0}^{tf} [h_p^k(t) - \alpha\Phi(t, h_p^k) + v_{ij}]_+ = Q_{ij}$$

- h_k+1 = iteration(h_k)

$$h_p^{k+1} = [h_p^k(t) - \alpha\Phi(t, h_p^k) + v_{ij}]_+$$

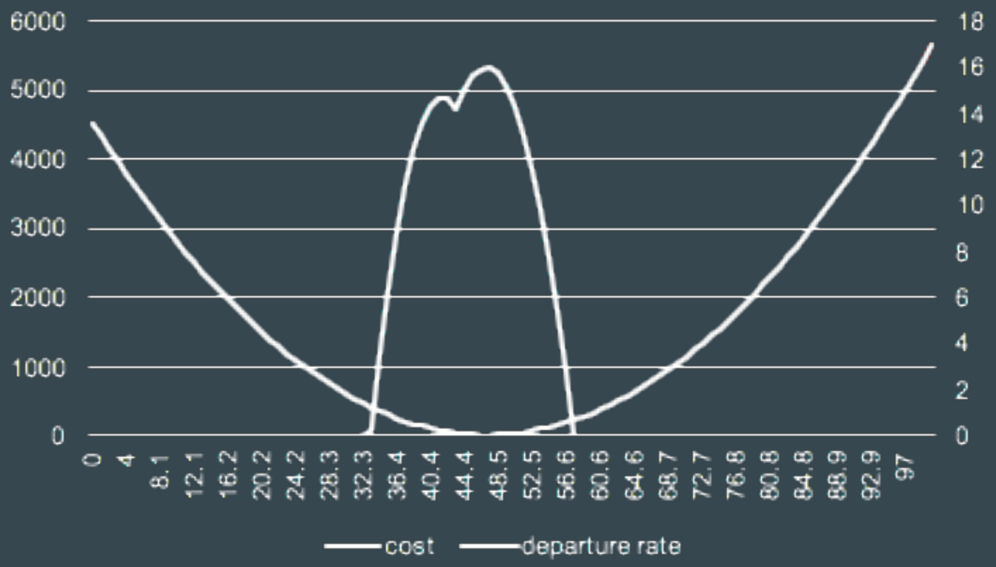
Result: Sioxfalls network



Joint Institute for Computational Sciences

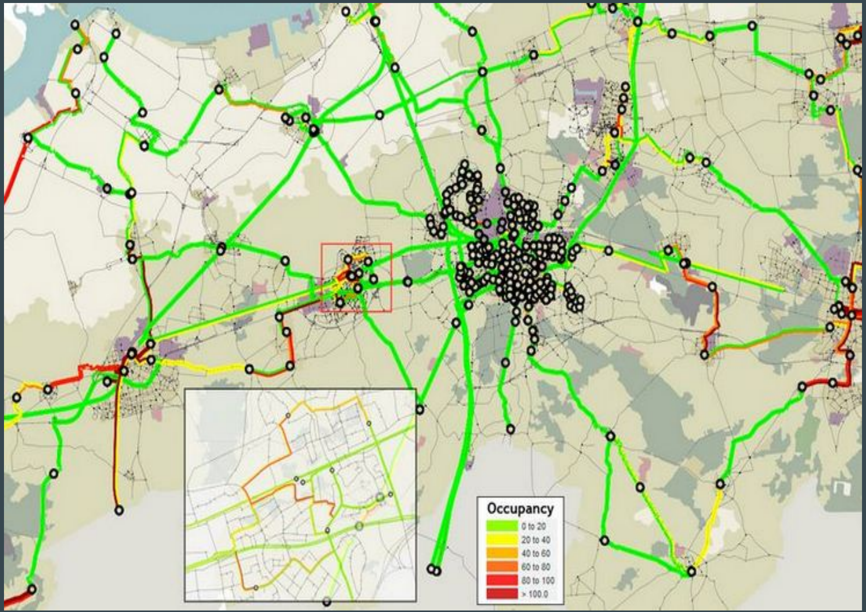
Computational Sciences

Result: Departure rate and Optimum cost



Future Work

- ➢ High speed
- ➢ Large practical case



END

Joint Institute for
Computational Sciences