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
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

\[ time = free\ flow\ time \times (1 + B \times (flow/capacity)^{Power}) \]
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

➢ Graphically

\[(y-x)^\top F(x) \geq 0, \forall y \in K\]
Variational Inequality

- **Category**

<table>
<thead>
<tr>
<th>VI ($K, q, M$)</th>
<th>VI ($K, q, M$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>VI ($K, F'$)</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>$\Leftrightarrow$</td>
</tr>
<tr>
<td>CP ($K, F$)</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$\downarrow$</td>
<td></td>
</tr>
</tbody>
</table>
Variational Inequality

❖ Why?

➢ Intuitive: Either scenario A or scenario 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)

**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 $\alpha_n$ that solves

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

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

$$0 \leq x \perp F(x) \geq 0.$$  \hfill (1.1.5)
VI on Static Traffic Assignment Problem (STA)

\[
\sum_{k \in R_w} f_{k}^{w} = q_{w},
\]

\[
C_{k}^{w} = \sum_{a \in A} \delta_{a,k}^{w} t_{a}(x),
\]

\[
x_{a} = \sum_{w \in W} \sum_{k \in P_{w}} \delta_{a,k}^{w} f_{k}^{w},
\]

\[
0 \leq C_{p}(h) - u_{w} \parallel h_{p} \geq 0, \quad \forall w \in \mathcal{W} \text{ and } p \in \mathcal{P}_{w};
\]

\[
\sum_{p \in \mathcal{P}_{w}} h_{p} = d_{w}(u), \quad \forall w \in \mathcal{W},
\]

\[
u_{w} \geq 0, \quad w \in \mathcal{W}.
\]

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

Traffic Problem complementarity problem

Nonlinear
VI on Static Traffic Assignment Problem (STA)

- **Limitation**
  - Unrealistic to find all path for a big graph
VI on Static Traffic Assignment Problem (STA)

Solution

- Find 7 nonsimilar path for each OD-pair to reduce Matrix size
- Use Shortest Path Algorithm
- 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 into NCP formulation in Siconos, which is a non-smooth numerical simulation package.
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.

```c
info = ncp_driver(problem, z, F, &options);
```
### Sample input

<table>
<thead>
<tr>
<th>Init node</th>
<th>Term node</th>
<th>Capacity</th>
<th>Length</th>
<th>Free Flow Time</th>
<th>B</th>
<th>Power</th>
<th>Speed limit</th>
<th>Toll</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>25900.200640</td>
<td>6.000000</td>
<td>6.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>23403.473190</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>25900.200640</td>
<td>6.000000</td>
<td>6.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4958.180928</td>
<td>5.000000</td>
<td>5.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>23403.473190</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>17110.523720</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>23403.473190</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>17110.523720</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>17782.794100</td>
<td>2.000000</td>
<td>2.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>4908.826730</td>
<td>6.000000</td>
<td>6.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>17782.794100</td>
<td>2.000000</td>
<td>2.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>4947.995469</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10000.000000</td>
<td>5.000000</td>
<td>5.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4958.180928</td>
<td>5.000000</td>
<td>5.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>4947.995469</td>
<td>4.000000</td>
<td>4.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>4898.587646</td>
<td>2.000000</td>
<td>2.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7841.811310</td>
<td>3.000000</td>
<td>3.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>23403.473190</td>
<td>2.000000</td>
<td>2.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>4898.587646</td>
<td>2.000000</td>
<td>2.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7841.811310</td>
<td>3.000000</td>
<td>3.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>5050.193156</td>
<td>10.000000</td>
<td>10.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>5045.822583</td>
<td>5.000000</td>
<td>5.000000</td>
<td>0.150000</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
</tr>
</tbody>
</table>

**Origin 1**

<table>
<thead>
<tr>
<th>1 :</th>
<th>0.0;</th>
<th>2 :</th>
<th>100.0;</th>
<th>3 :</th>
<th>100.0;</th>
<th>4 :</th>
<th>500.0;</th>
<th>5 :</th>
<th>200.0;</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 :</td>
<td>300.0;</td>
<td>7 :</td>
<td>500.0;</td>
<td>8 :</td>
<td>800.0;</td>
<td>9 :</td>
<td>500.0;</td>
<td>10 :</td>
<td>1300.0;</td>
</tr>
<tr>
<td>11 :</td>
<td>500.0;</td>
<td>12 :</td>
<td>200.0;</td>
<td>13 :</td>
<td>500.0;</td>
<td>14 :</td>
<td>300.0;</td>
<td>15 :</td>
<td>500.0;</td>
</tr>
<tr>
<td>16 :</td>
<td>500.0;</td>
<td>17 :</td>
<td>400.0;</td>
<td>18 :</td>
<td>100.0;</td>
<td>19 :</td>
<td>300.0;</td>
<td>20 :</td>
<td>300.0;</td>
</tr>
<tr>
<td>21 :</td>
<td>100.0;</td>
<td>22 :</td>
<td>400.0;</td>
<td>23 :</td>
<td>300.0;</td>
<td>24 :</td>
<td>100.0;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Result with 4 OD Pair
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

❖ Use cuSPARSE direct calculate Sparse Matrix
Mathematical Formulation

➢ Variational Inequality formulation:

○ Nash equilibrium nature

\[ h_p(t) > 0 \Rightarrow C_p(t, h) = \mu_{kl} \quad \forall \nu(t) \]

\[ C_p(t, h) \geq \mu_{kl} \quad \forall \nu(t). \]
Mathematical Formulation

➢ Dynamic Network Loading:

○ Given h, return path delay operator

○ Approximated by ODE systems

\[
\begin{align*}
\frac{dx^p_{a_1}(t)}{dt} &= g^p_{a_{i-1}}(t) - g^p_{a_i}(t) \quad \forall p \in \mathcal{P}, i \in [1, m(p)] \\
x^p_{a_i}(0) &= x_{a_i}^{p,0} \quad \forall p \in \mathcal{P}, i \in [1, m(p)] \\
h^\tau_{p,i}(t) &= g^p_{a_i}(t + D_{a_i}[x_{a_i}(t)]) \left(1 + D'_{a_i}[x_{a_i}(t)] \dot{x}_{a_i}(t)\right) \\
g^p_{a_{i-1}}(t) &= g^p_{a_i}(t + D_{a_i}[x_{a_i}(t)]) \left(1 + D'_{a_i}[x_{a_i}(t)] \dot{x}_{a_i}(t)\right) \quad \forall p \in \mathcal{P}, i \in [2, m(p)]
\end{align*}
\]
DTA: Algorithm

Overview

- Input $h^k$ for all $p$
- While (convergence condition == true)
  - $\text{ODE} = \text{makeOde}(h^k)$
  - $x = \text{solution}(\text{ODE})$
  - $\Phi = \text{getPhi}(x)$
  - $v = \text{getV}(\Phi)$, for all $p$
  - $h^{k+1} = \text{iteration}(\Phi,v)$ for all $p$
- Output $h^{k+1}$ for all $p$
DTA: Algorithm

➢ ODE = make_ODE(h)

➢ x = solution(ODE)
DTA: Algorithm

➢ Dp = getDp(x)
  ○ x: arc volume
  ○ Dp: traversal time
  ○ Phi: cost function

➢ Phi = getPhi(Dp)
  ○ F: penalty function

\[
D_p = \sum_{i=1}^{m(p)} [\tau_{a_i}^p (t) - \tau_{a_{i-1}}^p (t)] = \tau_{a_{m(p)}}^p (t) - t
\]

\[
\tau_{a_1}^p (t) = t + D_{a_1} [x_{a_1} (t)]
\]

\[
\tau_{a_i}^p (t) = \tau_{a_{i-1}}^p (t) + D_{a_i} [x_{a_i} (\tau_{a_{i-1}}^p (t))] \\
D(x) = \alpha \cdot x + \beta
\]

\[
\Phi_p (t) = D_p (t) + F[D_p (t) + t - T_A]
\]

\[
F(D_p (t) + t - T_A) = 0.5 \cdot (D_p (t) + t - T_A)^2
\]
DTA: Algorithm

➢ v = solution(Phi)

\[ \sum_{p \in P_{ij}} \int_{t_0}^{t_f} [h_p^k(t) - \alpha\Phi(t, h_p^k) + v_{ij}]^+ = Q_{ij} \]

➢ \( h_{k+1} = \text{iteration}(h_k) \)

\[ h_p^{k+1} = [h_p^k(t) - \alpha\Phi(t, h_p^k) + v_{ij}]^+ \]
Result: Sioxfalls network
Result: Departure rate and Optimum cost
Future Work

➢ High speed
➢ Large practical case