Effective Route Guidance in Traffic Networks

Lectures developed by
Andreas S. Schulz and Nicolás Stier

May 13, 2004

Outline

- Lecture 1

  Route Guidance; User Equilibrium; System Optimum; User Equilibria in Networks with Capacities.

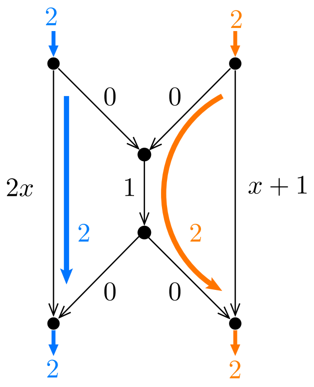
- **Lecture 2**

  Constrained System Optimum; Dantzig-Wolfe Decomposition; Constrained Shortest Paths; Computational Results.

Review of Traffic Model

- Directed graph $G = (V, A)$ with capacities, k demands (o_i, d_i) with rate r_i
- Flows on paths f_P. Can be non-integral.
- Traversal times: *latency functions* $t_a(\cdot)$
  → *continuous* and *nondecreasing*
  → belong to a given set $\mathcal{L}$ (e.g. linear)
- The total travel time of a flow is:

$$C(f) := \sum_{a \in A} t_a(f_a) f_a$$



$2 \cdot 4 + 2 \cdot 1 = 10$

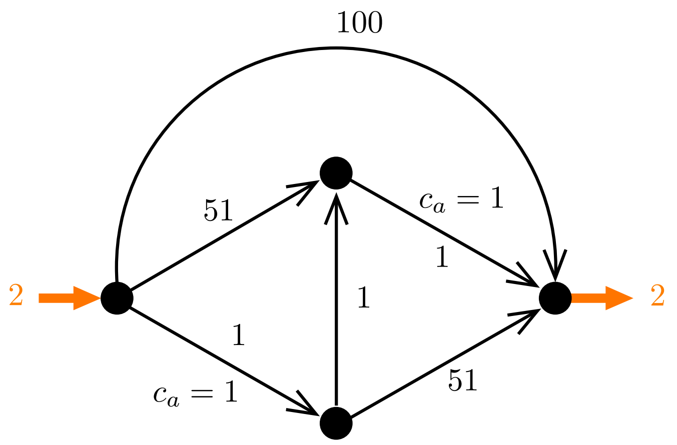
Review of First Lecture

No capacities	With capacities
UE unique	Set of **UE** may be non-convex
UE/**SO** $\geq \alpha(\mathcal{L})$	**UE**/**SO** unbounded
UE/**SO** $\leq \alpha(\mathcal{L})$	**BUE**/**SO** $\leq \alpha(\mathcal{L})$

selfish users	**central planner**	**the goal**
optimize own travel time	optimize system welfare	
fair, not efficient	efficient, not fair	**fair, efficient**

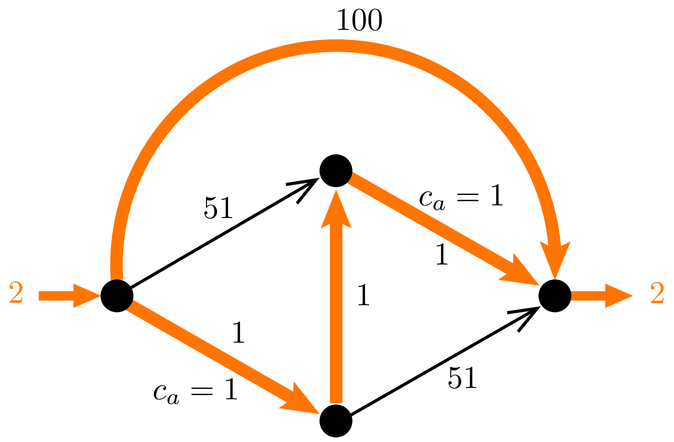
Long Detours in SO

Instance with constant latencies:



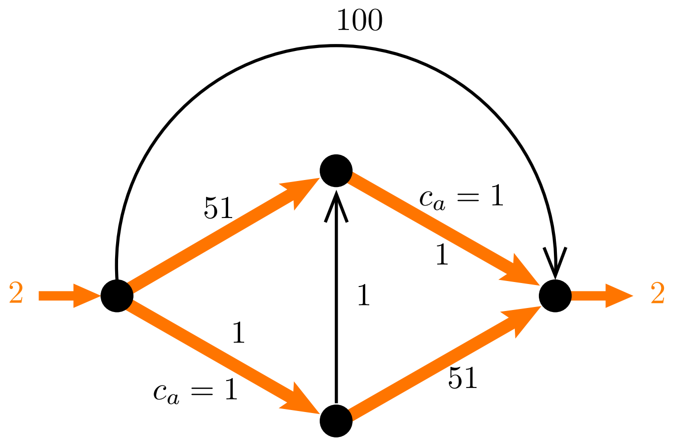
Long Detours in SO

SO routes 1 unit along each path: $C(\textbf{SO}) = 100 + 3$. **Unfair!**



Long Detours in SO

Compare to routing 1 unit along the other paths: $C(f) = 104$ but **fair!**



Constrained System Optimum

Route Guidance

- **SO** cannot be implemented in practice due to unfairness
- **UE** does not take into account the global welfare

Use constrained SO instead!

- **CSO** = min total travel time
  s.t. demand satisfied
  users are assigned to "fair" routes
  capacity constraints

Technological Requirements

exact knowledge of the current position

2-way communication to a main server

Constrained SO: Normal Lengths

Normal lengths: a-priori belief of network

- Geographic distances
- Free-flow travel times (times in empty network)
- Travel times under **UE**

Notation:

- normal length of arc: ℓ_a
- normal length of path: $\ell_P = \sum_{a \in P} \ell_a$

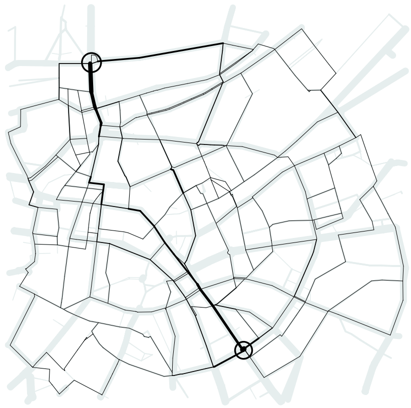
Constrained SO: Definition

- Fix a tolerance $\varepsilon \geq 0$
- A path $P \in \mathcal{P}_i$ is valid if $\ell_P \leq (1+\varepsilon) \times \min_{Q \in \mathcal{P}_i} \ell_Q$
- Definition:

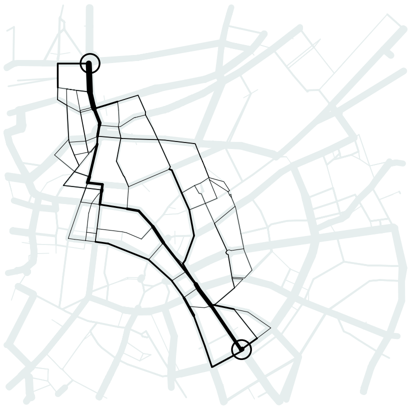
$$\textbf{CSO}_\varepsilon = \text{min total travel time}$$

$$\begin{aligned} \text{s.t.} \sum_{P \in \mathcal{P}_i : P \textbf{ valid}} f_P &= r_i && \text{for all } i \\ \sum_{P \ni a} f_P &\leq c_a && \text{for } a \in A \\ f_P &\geq 0 \end{aligned}$$

CSO Example



SO



CSO

Remarks about CSO

- It is a non-linear, convex, minimization problem over a polytope (constrained min-cost multi-commodity flow problem)
- We solve it using the Frank-Wolfe algorithm: we solve a sequence of linear programs
- No need to consider all path variables simultaneously: we use column generation

Computing CSO

- Each algorithm uses the next as a subroutine:

1. Frank-Wolfe algorithm: linearize using current gradient
2. Simplex algorithm to solve resulting LP
3. Column generation to handle exponentially many paths
4. Constrained Shortest Path Problem (CSPP) algorithm for pricing
5. Dijkstra's algorithm as a routine for CSPP

Frank-Wolfe Algorithm

0. Initialization: start with flow x^0. Set $k = 0$ and $LB = -\infty$.

1. Update upper bound: set $UB = C(x^k)$ and $\bar{x} = x^k$.

2. Compute next iterate:
   $z^* = \min\{\, C(\bar{x}) + \nabla C(\bar{x})^T (x - \bar{x}) : x \text{ feasible} \,\}$.
   Let x^* be the optimal flow.

3. Solve the line-search problem and set $x^{k+1} = \bar{x} + \bar{\alpha}(x^* - \bar{x})$.

4. Update lower bound: set $LB = \max\{LB, z^*\}$.

5. Check stopping criteria: if $|UB - LB| \leq$ tolerance, STOP!
   Otherwise, set $k = k + 1$ and go to step 1.

Linear Problem and Column Generation

- Let $t_a = \frac{\partial C(f^i)}{\partial f_a}$ be the objective coefficient of f_a in the LP
- As there are exponentially many paths in the LP, we form LP' with a subset $\mathcal{P}' \subseteq \mathcal{P}$ of valid paths:

$$
\begin{array}{llll}
\min & \sum_{a \in A} t_a f_a & & \\
\text{s.t.} & \sum_{P \ni a} f_P = f_a & \text{for all } a \in A & \\
 & \sum_{\text{valid } P \in \mathcal{P}'_i} f_P = r_i & \text{for all } i = 1, \ldots, k & (1) \\
 & f_a \leq c_a & \text{for all } a \in A & (2) \\
 & f_P \geq 0 & \text{for all } P \in \mathcal{P}' &
\end{array}
$$

Linear Problem and Column Generation II

- For each demand i, let σ_i be the dual variable corresponding to (1)
- For each arc a, let $\pi_a \geq 0$ be the dual variable corresponding to (2)
- Solution optimal in LP $\Leftrightarrow \displaystyle\sum_{a \in P} (t_a + \pi_a) \geq \sigma_i \quad \forall$ valid $P \in \mathcal{P}_i$
- **The Pricing Problem**:

  For every commodity i, either find a valid path in $\mathcal{P}_i$ with modified cost less than σ_i or assert that no such path exists.

Can be solved as a "Constrained Shortest Path Problem" !

Algorithm for Solving the LP

```
1. Solve the linear program LP'
2. Let $\sigma_i$ and $\pi_a$ be the simplex multipliers of the current optimal solution
3. for all $i$:   find shortest valid path $P_i$ in $\mathcal{P}_i$ w.r.t. arc costs $t_a + \pi_a$
4. if $\sum_{a \in P_i} (t_a + \pi_a) \geq \sigma_i$ for all $i$
5.       Solution is optimal for LP. STOP
6. else
7.       Remove one or more non-basic variables from $\mathcal{P}'$
8.       Add at least one path $P_i$ with $\sum_{a \in P_i} (t_a + \pi_a) < \sigma_i$ to $\mathcal{P}'$
9. goto 1
```

Observations for Solving the LP

- Empirically, very advantageous to add as many new columns to restricted master problem as possible

⇒ Add all paths that price favorably until we run out of space

⇒ Non-basic variables removed when their slots are needed for new candidate paths

- We observed a reduction in computation time by factors of about 50, compared to always adding a single column and removing another one

The Pricing Problem

1. Frank-Wolfe algorithm: linearize using current gradient

2. Simplex algorithm to solve resulting LP

3. Column generation to handle exponentially many paths

4. Constrained Shortest Path Problem (CSPP) algorithm for pricing

5. Dijkstra's algorithm as a routine for CSPP

Shortest Path Problem

Distance Labels

We want the shortest paths from node 1 to all other nodes

- Throughout the run, nodes will have *distance labels*:
  let $d(j)$ denote the label of $j \in A$

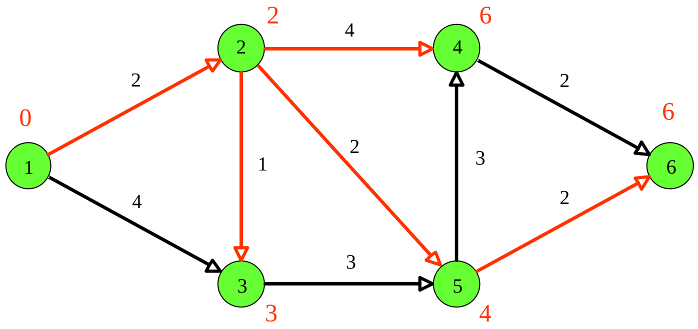
- For $j \in A$, let $d^*(j)$ denote the *shortest distance* from 1 to j

- Labels can be :
  - Temporary: shortest distance found so far
  - Let $T = \{$ temporarily labeled nodes$\} \Rightarrow d(j) \geq d^*(j) \quad \forall j \in T$
  - Permanent: when the label is the shortest distance
  - Let $S = \{$ permanently labeled nodes $\} \Rightarrow d(j) = d^*(j) \quad \forall j \in S$

Optimality Conditions

The distance labels d are shortest path distances iff

$$\boxed{d(j) \leq d(i) + t_{ij} \qquad \forall\,(i,j) \in A}$$

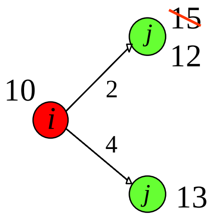


Dijkstra's Algorithm: Update

Given a label $d(i)$ for node i, Update(i) improves the labels of i's neighbors:

Procedure Update(i)

```
for each $(i,j) \in A$ do
    if $d(j) > d(i) + t_{ij}$ then
        $d(j) := d(i) + t_{ij}$;
        $pred(j) := i$;
```



Dijkstra's Algorithm: Main

This routine computes shortest paths from node 1 to all other nodes:

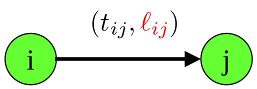
```
$S := \{1\}$; $T := V \setminus \{1\}$;
$d(1) := 0$; $d(j) := \infty$ for $j = 2, 3, \ldots, n$;
update$(1)$;
while $S \neq V$ do
    // find minimum temporary labeled node and update it
    $i :=$ argmin$\{\, d(j) \;:\; j \in T \,\}$;
    $S := S \cup \{i\}$; $T := T \setminus \{i\}$;
    update$(i)$;
```

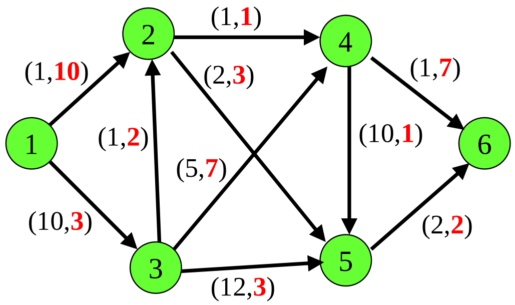
Shortest Path Example

Constrained Shortest Path

Constrained Shortest Path Example

Find the fastest path from node 1 to node 6 with a length of at most 10





Labels

- A label $d(j)$ is now a tuple $d(j) = (d_t(j), d_\ell(j))$
  - $d_t(j)$ is the travel time
  - $d_\ell(j)$ the length of a path from node 1 to j
- A node may have several labels at the same time
- A label $d(j)$ dominates $d'(j)$ iff $d_t(j) \leq d'_t(j)$ and $d_\ell(j) \leq d'_\ell(j)$.
- In the algorithm, every node j has a set $T(j)$ of temporary labels and a set $S(j)$ of permanent labels
- Let T and S be the sets of all temporary and permanent labels, resp.

Update

Given a label $d(i)$ for node i, Update improves the labels of i's neighbors:

```
Procedure Update(node $i$, label $d(i)$)
if $d_t(i) \geq$ min. time of a feasible path from node $1$ to $n$ so far
    return;
for each $(i,j) \in A$ do
    $d^{\text{new}}(j) := d(i) + (t_{ij}, \ell_{ij})$;        // new label for $j$
    if $d^{\text{new}}_\ell(j) \leq L$ and $d^{\text{new}}$ is not dominated by other labels in $j$
        add $d^{\text{new}}$ to $T(j)$;
        delete dominated labels from $T(j)$;
```

Labeling Algorithm

This routine computes a fastest path from node 1 to node n such that $\ell(\text{path}) \leq L$:

```
$S(1) := \{(0,0)\}$;
update($1$, $(0,0)$);
while $S(n)$ is empty do
    // find minimum temporary labeled node
    $d :=$ argmin $\{ d_t \ : \ d \in T \}$;
    $i :=$ corresponding node;
    move the label $d$ from $T(i)$ to $S(i)$;
    update($i$, $d$);
```

This can degenerate into a huge enumeration

Constrained Shortest Path Example

Alternative Algorithm for CMCFP

Idea: **Forget about Capacity Constraints**

1. Frank-Wolfe algorithm: linearize using current gradient
2. Simplex algorithm to solve resulting LP
3. Column generation to handle exponentially many paths
4. Constrained Shortest Path Problem (CSPP) algorithm for pricing
5. Dijkstra's algorithm as a routine for CSPP

Relaxing Capacity Constraints

$$\textbf{CSO}_\varepsilon = \min \text{ total travel time}$$

$$\begin{array}{llll} \text{s.t.} & \displaystyle\sum_{P \in \mathcal{P}_i : P \text{ valid}} f_P & = r_i & \text{for all } i \\ & \cancel{\displaystyle\sum_{P \ni a} f_P} & \cancel{\leq c_a} & \cancel{\text{for } a \in A} \\ & f_P & \geq 0 & \end{array}$$

- Capacity constraint violated ⇒ $C(f) = \infty$ because latency is infinity
- Minimization takes care of making the solution feasible
- No capacity constraints → the problem is separable!

CSO can be found with a sequence of CSPP

Computational Experience

Computational Experiments

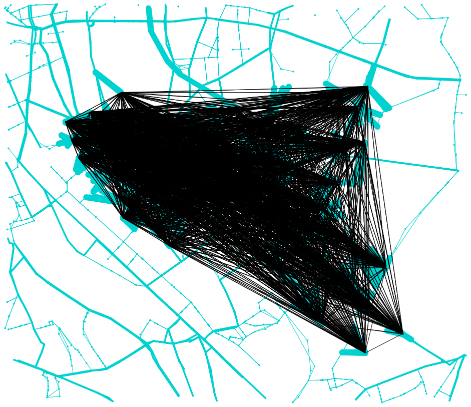
We used real-world instances obtained from *DaimlerChrysler* (Berlin) and from the *Transportation Network Test Problems* website:

http://www.bgu.ac.il/~bargera/tntp/

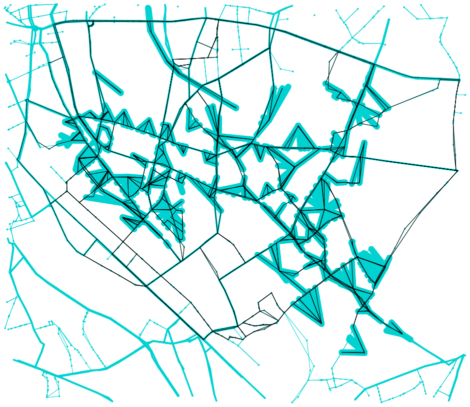
Instance Name	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert K\rvert$	$\lvert A\rvert \cdot \lvert K\rvert$
Sioux Falls	24	76	528	40K
Friedrichshain	224	523	506	265K
Winnipeg	1,067	2,975	4,344	13M
Neukölln	1,890	4,040	3,166	13M
Mitte, Prenzlauerberg & Friedrichshain	975	2,184	9,801	21M
Chicago Sketch	933	2,950	83,113	245M
Berlin	12,100	19,570	49,689	972M

Part of an Instance

Demand



Solution



Unfairness

- Normal unfairness of path P for OD-pair $i = \frac{\ell_P}{\min_{Q\in\mathcal{P}_i} \ell_Q}$

  $\rightarrow$ $1 \leq$ normal unfairness $\leq 1+\varepsilon$

- Loaded unfairness of path P for OD-pair $i = \frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(f)}$

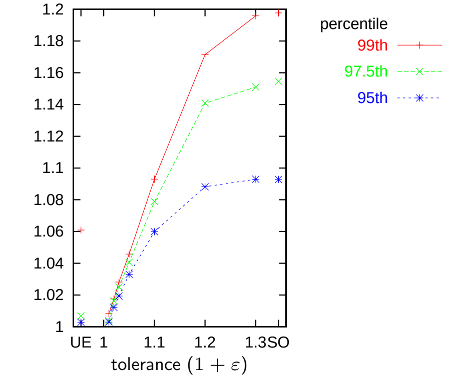
  $\rightarrow$ $1 \leq$ loaded unfairness

- UE unfairness of path P for OD-pair $i = \frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(\textbf{BUE})}$

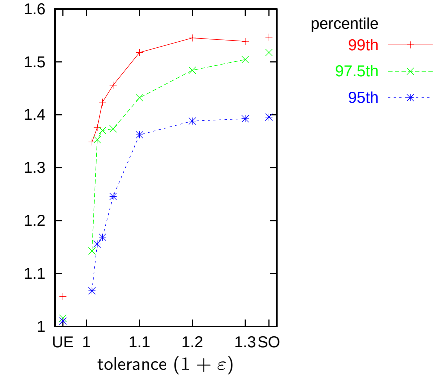
  $\rightarrow$ $0 \leq$ UE unfairness

Unfairness Percentiles

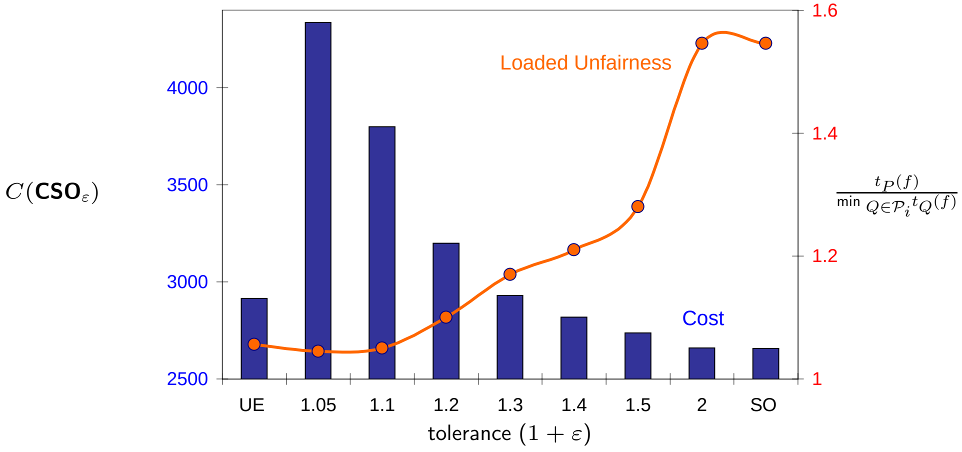
normal unfairness:
controlled directly



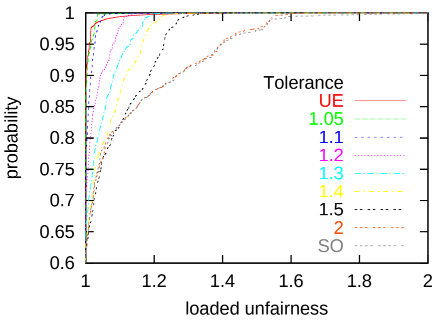
loaded unfairness:
influenced



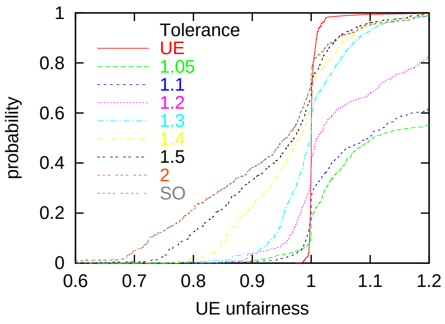
Free-flow Normal Lengths: High Cost



Unfairness Distributions: High Travel Times

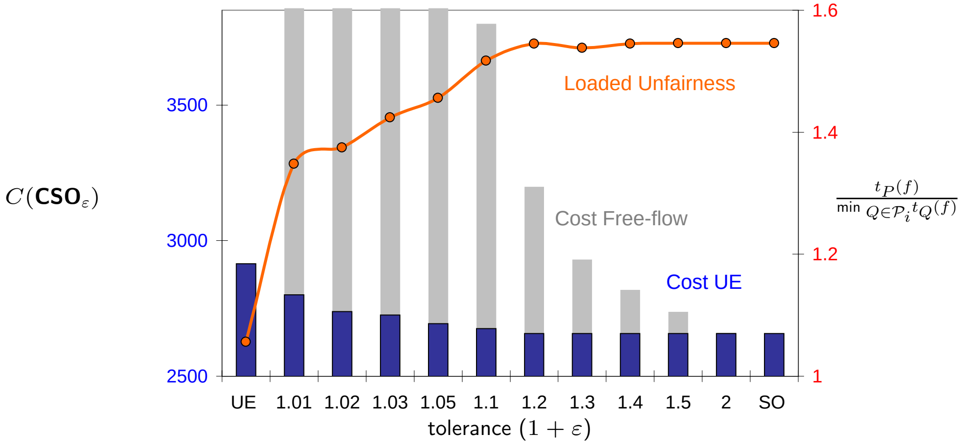


$$\frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(f)}$$

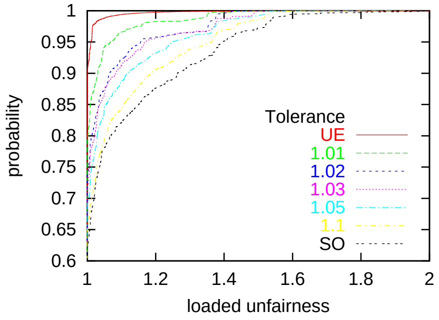


$$\frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(\mathbf{BUE})}$$

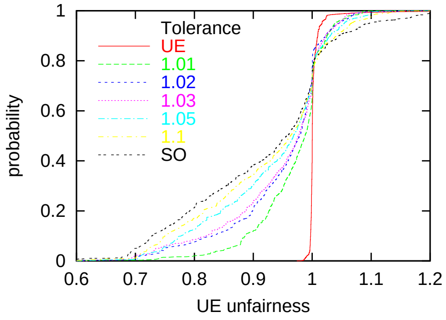
UE Travel Times: Good Normal Lengths



Unfairness Distributions: Fair Enough



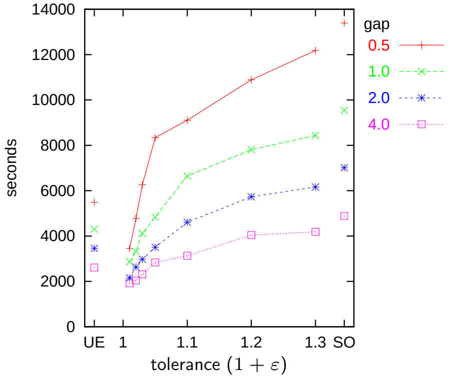
$$\frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(f)}$$



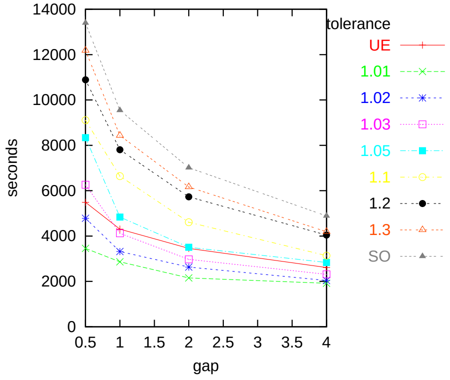
$$\frac{t_P(f)}{\min_{Q\in\mathcal{P}_i} t_Q(\textbf{BUE})}$$

Convergence

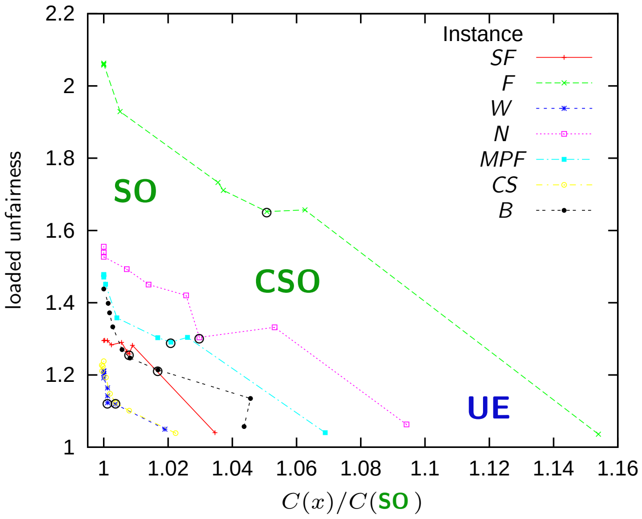
More difficult with bigger tolerance



Runtime grows exponentially



CSO allows us to control the tradeoff between efficiency and unfairness

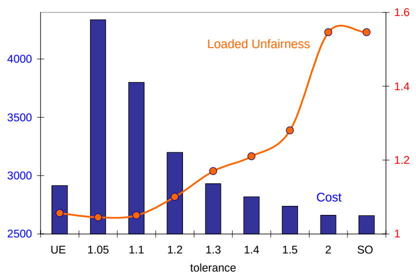


Solutions marked with '○' denote $\textbf{CSO}_{1.02}$

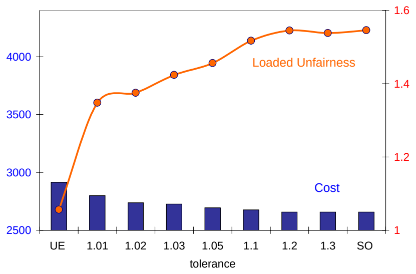
Review

- Results:

Free-flow normal length



UE normal length



Conclusion

- Optimization Approach to Route Guidance
  - Conventional route guidance methods focus on the individual
  - **SO** not implementable
  - **UE** not efficient
  - **CSO** is a better alternative: efficient and fair
- Demand-dependent normal lengths are a better choice
- Considered Networks with Capacities
  - Multiple equilibria
  - Worst **UE** is unbounded
  - Guarantee for best **UE** is as good as without capacities

Summary

- In principle, the system performance can be optimized while obeying individual needs and systems response.
- In fact, many different tools, from non-linear optimization, from linear programming, and from discrete optimization, nicely complement each other to lead to a fairly efficient algorithm for huge (static) instances.
- Yet, more (dynamic) ideas needed before technology ready for field test.

THE END

2002 Urban Mobility Study shows we could be better

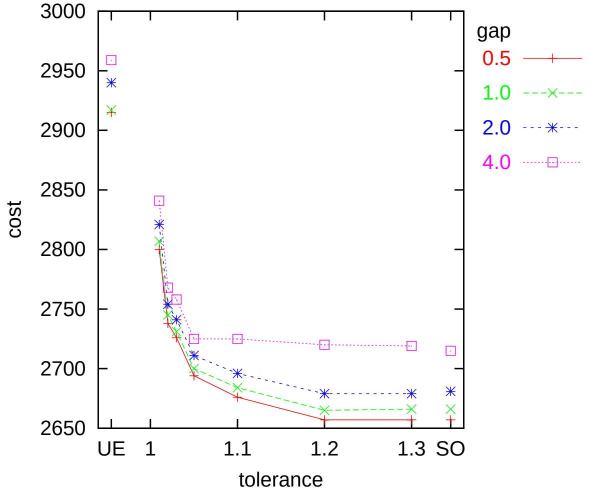
(http://mobility.tamu.edu/ums)

	1982	2000
time penalty for peak period travelers	16 hours	62 hours
period of time with congestion	4.5 hours	7 hours
volume of roadways with congestion	34%	58%

UE Travel Times: Good Normal Lengths

tolerance	cost	99th percentile loaded unfairness
UE	2915	1.056
1.01	2800	1.348
1.02	2738	1.375
1.03	2726	1.424
1.05	2694	1.456
1.10	2676	1.517
1.20	2657	1.545
1.30	2657	1.538
SO	2657	1.546

Solution Quality: UE normal lengths are good


cost
UE
SO
tolerance
gap