

An Efficient Insertion Operator in Dynamic Ridesharing Services

**Yi Xu, Yongxin Tong, Yexuan Shi,
Qian Tao, Ke Xu, Wei Li**

**Key Laboratory of Software Development Environment,
School of Computer Science and Engineering,
Beihang University, China**



北京航空航天大学
BEIHANG UNIVERSITY

Outline

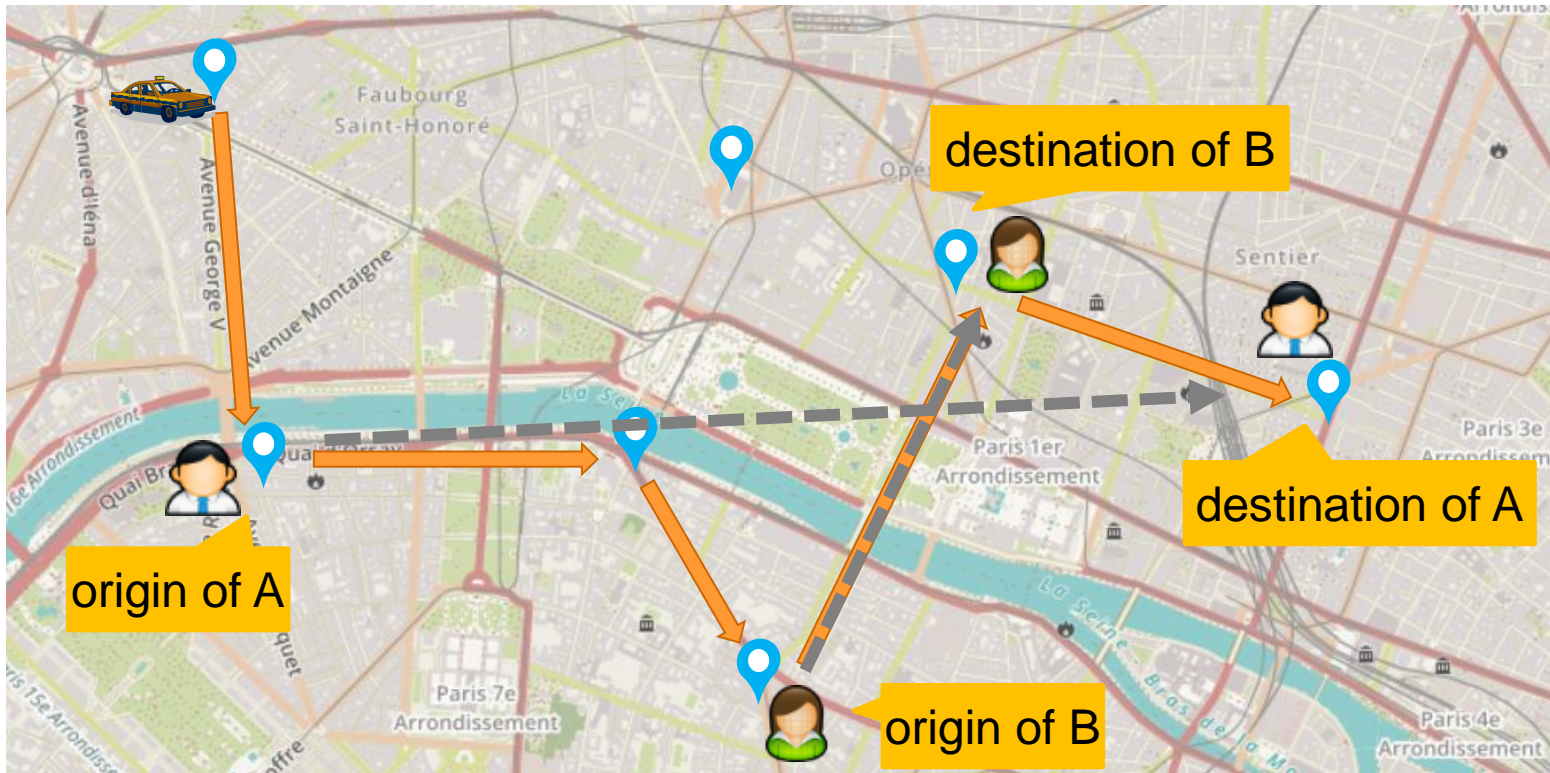
- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

Dynamic Ridesharing Services

- Dynamic ridesharing: services that arrange one-time shared rides on short notice

Dynamic Ridesharing Services

- Dynamic ridesharing: services that arrange one-time shared rides on short notice.
 - Car-pooling,



- Dynamic ridesharing: services that arrange one-time shared rides on short notice.
 - Car-pooling, **Food Delivery**,

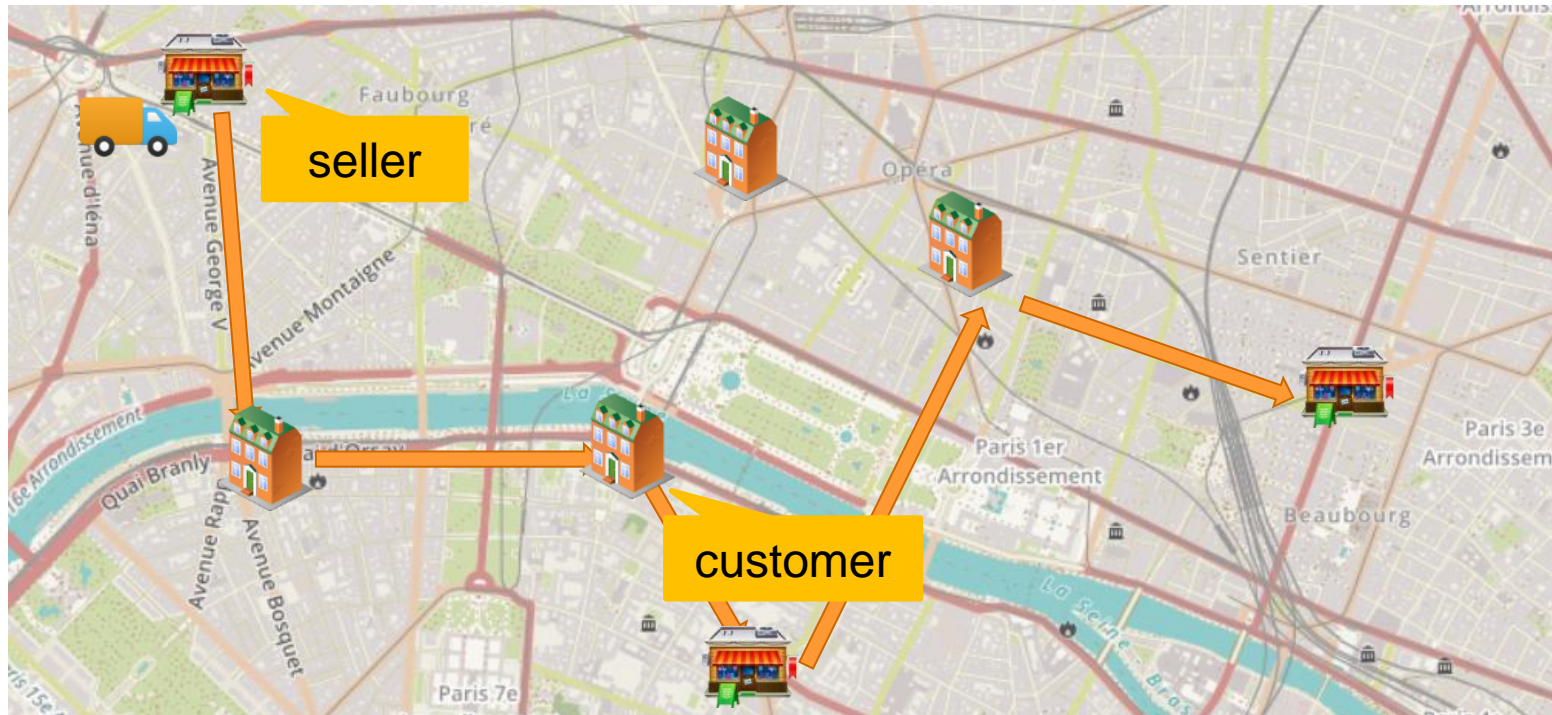


seamless



Dynamic Ridesharing Services

- Dynamic ridesharing: services that arrange one-time shared rides on short notice.
 - Car-pooling, Food Delivery, **Last-mile Logistics**



How to plan a route for the worker is central for DR

Dynamic Ridesharing Services

- The problem of route planning in dynamic ridesharing is very difficult and most existing solutions are heuristic

T-share

[ICDE'13]

Adaptive
insertion

[EJOR'11]

Kinetic

[VLDB'14]

Prune
GreedyDP

[VLDB'18]

Algorithm 2: Insertion feasibility check
Data: Query Q , taxi status V , insertion position i for Q , insertion position j for Q , current time t_{cur}
Result: Return *new_schedule* if the insertion succeeds; otherwise return *False*.
/* $t_{cur} \rightarrow$ represents travel time between two locations here */
1 if $t_{cur} + (|V| \rightarrow Q) > Q_{up}$ then /* cannot arrive Q on time */
2 return *False*
3 if the time delay incurred by the insertion of Q causes the slack time of any point after position i in *schedule* V s smaller than 0 then
4 return *False*
5 *new_schedule* \leftarrow insert Q into V s at position i /* the slack time of each pickup(delivery) point after position i is updated accordingly at the sensitive v */
6 if the i -th point of *new_schedule* is i -th point of V s
7 $i_1 \leftarrow$ the geographical location of the j -th point of V s
8 if $i_1 + (i_j \rightarrow Q) > Q_{up}$ then /* cannot arrive Q on time */
9 return *False*
10 if the time delay incurred by the insertion of Q causes the slack time of any point after position j in *new_schedule* smaller than 0 then
11 return *False*
12 *new_schedule* \leftarrow insert Q into *new_schedule* at position j /* the slack time of each pickup(delivery) point in *new_schedule* is updated accordingly at the sensitive v */
13 return *new_schedule*



Algorithm 1: The advanced insertion algorithm
Set $S_1 = \{0\}$; (S_1 = set of feasible service sequences with respect to customers $1, \dots, 1$)
for each $i \in \{1, \dots, N\}$ do
Set $S_i = \emptyset$;
for each $s \in S_{i-1}$ do
for each $h \in \{1, \dots, |s| + 1\}$ and $k \in \{h + 1, \dots, |s| + 2\}$ do
Set $r = I(s, i, h, k)$; (I = the insertion function)
if service sequence r is feasible then
 $S_i = S_i \cup \{r\}$;
end if
end for
end for
if $S_i = \emptyset$ then
STOP: The problem has no solution;
end if
end for
end for
for each $s \in S_N$
Calculate cost $C(s)$;
end for



Algorithm 1 insertNodes algorithm.
Parameter: root node l , request points $P' = (x_1, x_2, \dots)$, current depth $depth$
if *feasible*($l, x_1, depth + d(l, x_1)$) then
Initialize *fail* = 0
for each c such that edge (l, c) exists do
copyNodes($n, c, d(l, n) + d(n, c) - d(l, c)$)
if copy failed, set *fail* = 1
end for (Insert remaining request points to n)
if *fail* = 0 and $|P'| > 1$ then (Detour now begins negative because we haven't inserted x_2 yet)
insertNodes($n, \{x_2, \dots\}, -d(x_1, x_2)$)
end if (Now insert request points into children)
for each c such that edge (l, c) exists do
insertNodes($c, P', detour + d(l, c)$)
if insert failed, delete (l, c)
end for
if *fail* = 0 then
Add(n, l, n)
else if no nodes c with edge (l, c) exists then
Insert failed, notify caller that this sub-tree is infeasible
else
Insert succeeded
end if
else
Insert failed, notify caller that this sub-tree is infeasible
end if



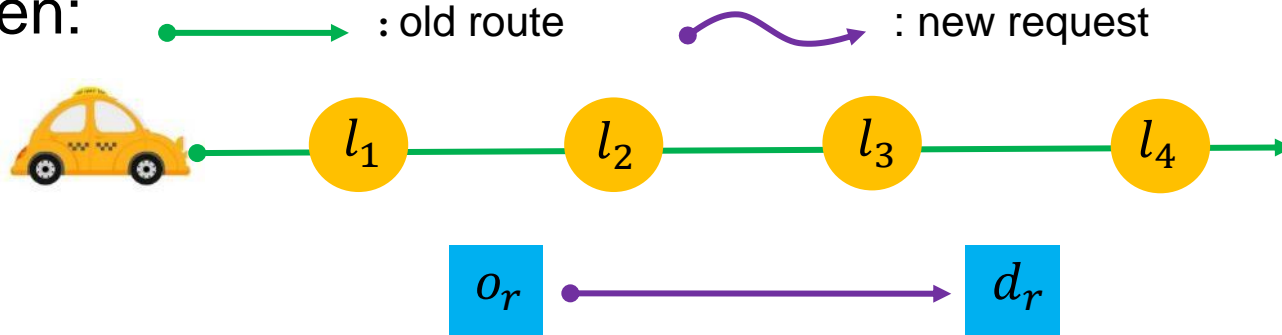
Algorithm 3: Linear DP Insertion
input : a worker w with route S_w and a request r
output: a new route S^* for the worker w
1 $S^* \leftarrow S_w, \Delta^* \leftarrow \infty, Dio[0] \leftarrow \infty, Plc[0] \leftarrow NIL$;
2 Initialize *arr*, *dis*, *plc* by Eq. (6), Eq. (9);
3 foreach $j \leftarrow 0$ to n do
4 Update Δ^*, i^*, j^* with special cases as shown in Fig. 2a and Fig. 2b;
5 if $j > 0$ and Corollary 1 is satisfiable then
6 $\Delta_j^* \leftarrow det(l_j, d_r, l_{j+1}) + Dio[j]$;
7 if $\Delta_j^* < \Delta^*$ then
8 $\Delta^* \leftarrow \Delta_j^*, i^* \leftarrow Plc[j], j^* \leftarrow j$;
9 if *arr*[j] + *dis*(o_r, c_r) $> c_r$ then break;
10 Update *Dio*[$j + 1$] and *Plc*[$j + 1$] according to Eq. (11) and Eq. (12);
11 if $\Delta^* < \infty$ then
12 $S^* \leftarrow$ insert o_r at i^* -th and d_r at j^* -th in S_w ;
13 return S^* ;



Insertion Operator: insert a new request into the current route

Insertion Operator

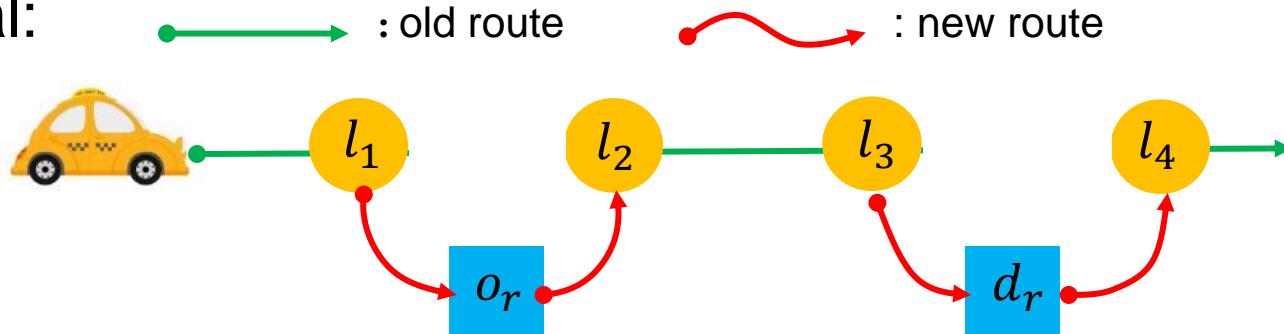
Given:



Insertion:

finding the appropriate location to
Insert the new request

Goal:



Outline

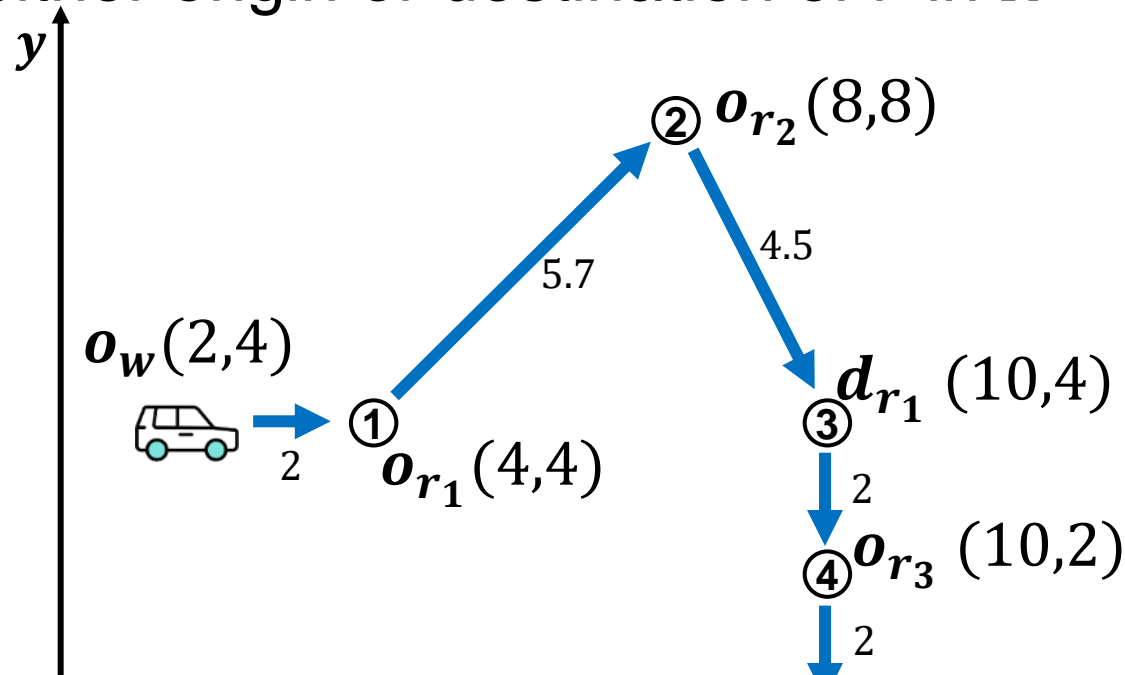
- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

Worker

- Worker: $w = \langle o_w, c_w \rangle$
 - o_w : current location
 - c_w : capacity
 - Capacity constraint: the number of passengers he takes at the same time is less than his capacity.
- Request: $r = \langle o_r, d_r, t_r, e_r, c_r \rangle$
 - o_r, d_r : origin and destination
 - t_r, e_r : release time and deadline
 - c_r : occupation
 - Deadline constraint: the request is delivered before the deadline.

Route

- Route: $S_R = \langle l_0, l_1, \dots, l_n \rangle$
 - R : a set of requests
 - l_0 : the current location of worker
 - l_i : either origin or destination of r in R



Feasible: (1) Capacity constraint; (2) Deadline constraint

Problem Formulation

- Insertion Operator

- Given:

- Worker w , feasible original route S_R , new request r'

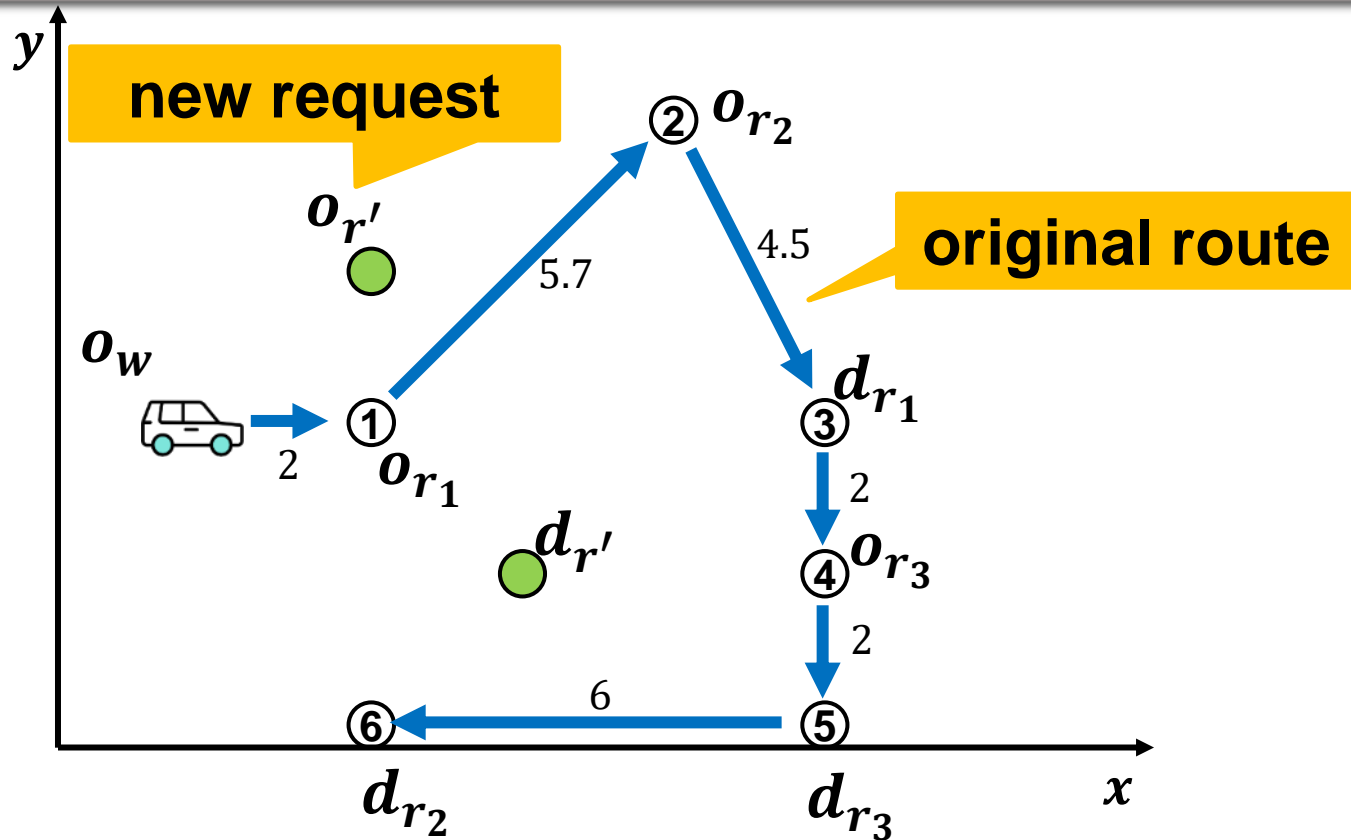
- Goal:

- Inserts r' into S_R to obtain a new feasible route with:

- **Minimizing maximum flow time** of all requests:
i.e., minimizing maximum waiting time of all requests (waiting time = delivery time - release time)
- **Minimizing total travel time** of the worker, i.e., the delivery time of the last request for the worker

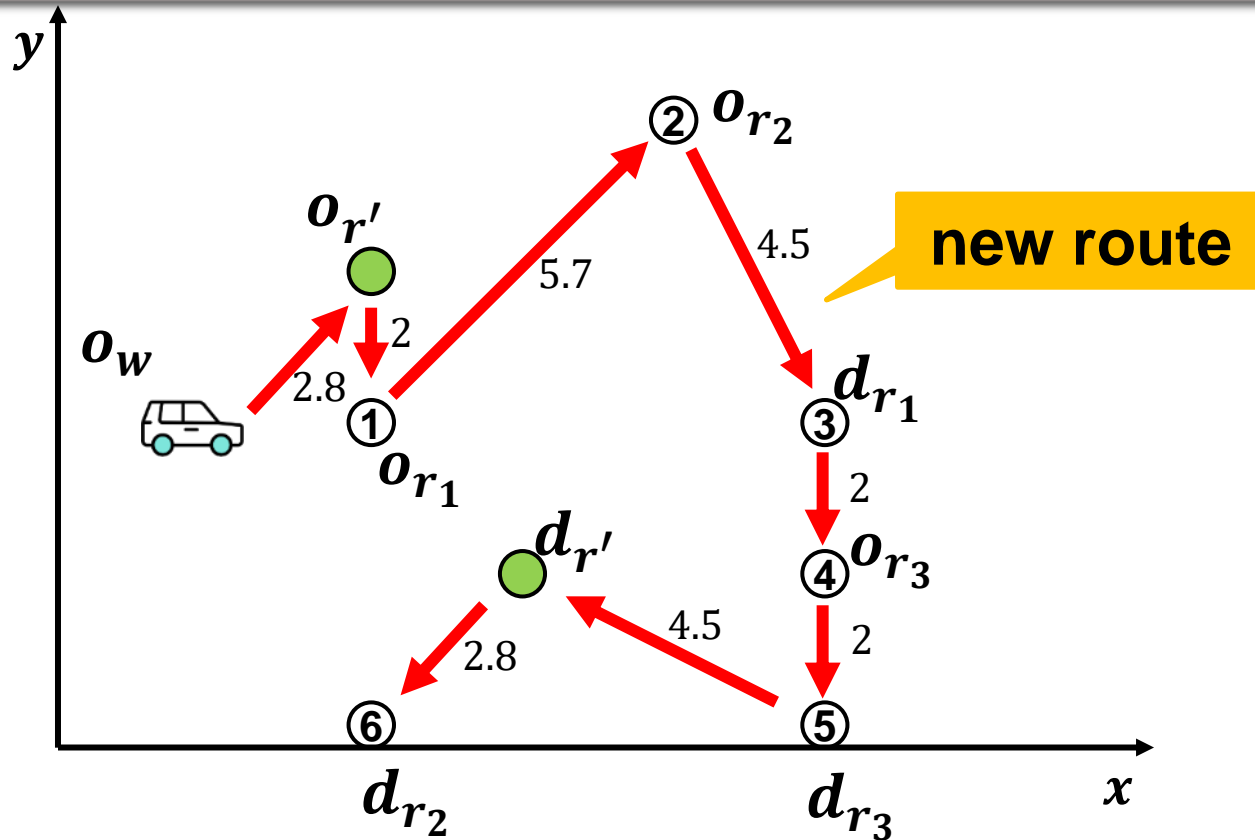
**Focus of
This talk.**

Example



	t_r	e_r	o_r	d_r	c_r
r_1	0	25	(4,4)	(10,4)	1
r_2	0	37	(8,8)	(4,0)	2
r_3	0	33	(10,2)	(10,0)	1
r'	0	26	(4,6)	(6,2)	1

Example



	t_r	e_r	o_r	d_r	c_r
r_1	0	25	(4,4)	(10,4)	1
r_2	0	37	(8,8)	(4,0)	2
r_3	0	33	(10,2)	(10,0)	1
r'	0	26	(4,6)	(6,2)	1

Problem

Time complexity: $O(n^3)$

||

Calculate objective and
check constraints in $O(n)$

×

Enumerate all possible
insertion pairs in $O(n^2)$

Basic Algorithm

Goal

Time consuming



**How to reduce the
time complexity?**

Basic Algorithm

Outline

- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

Partition-based Framework

Time complexity: $O(n^3)$

||

Calculate objective and
check constraints in $O(n)$

×

Enumerate possible
insertion pairs in $O(n^2)$

Basic Algorithm



Time complexity: $O(n^2)$

||

Calculate objective and
check constraints in $O(1)$

×

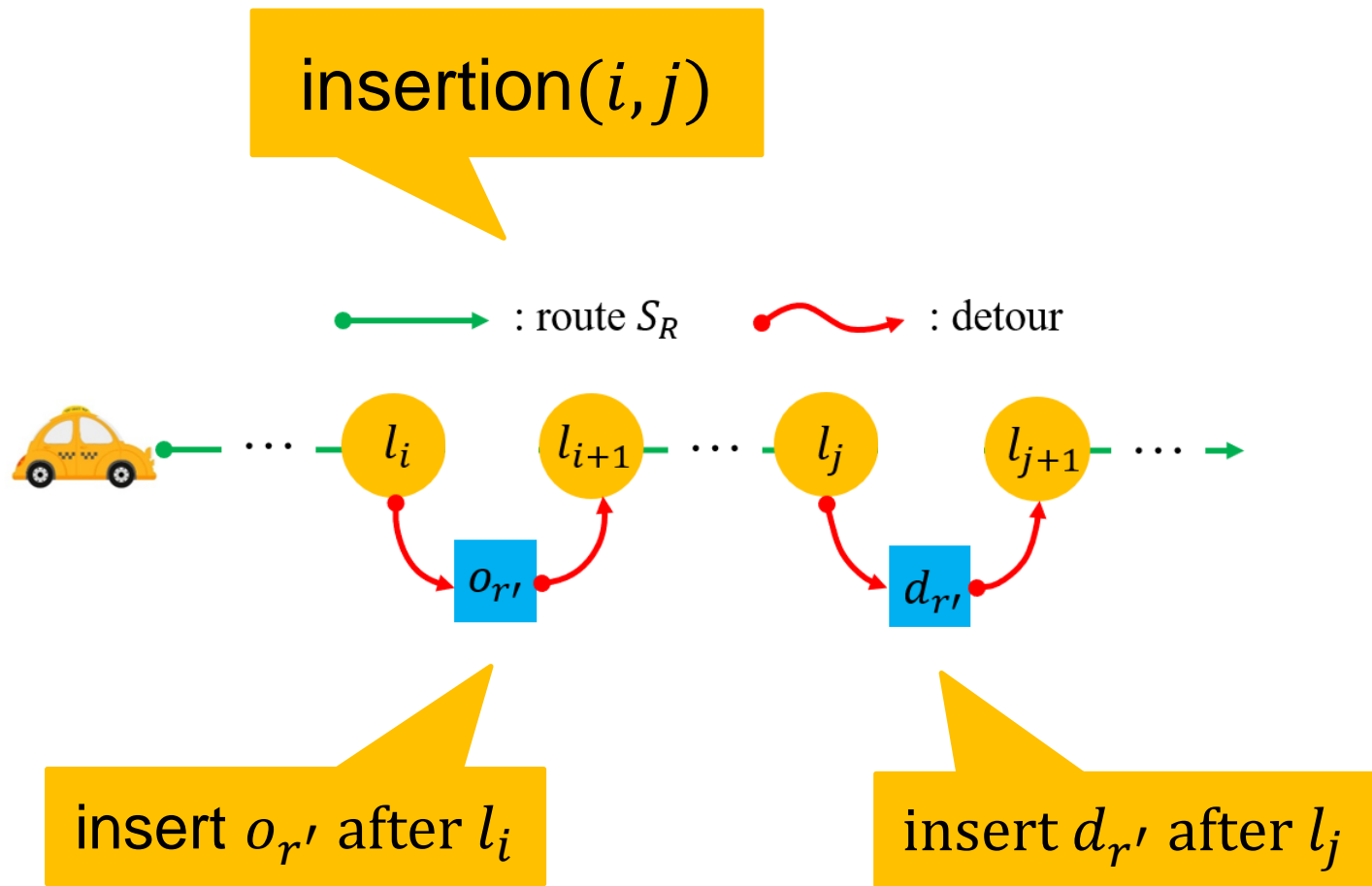
Enumerate all possible
insertion pairs in $O(n^2)$

+

Pre-processing in $O(n^2)$

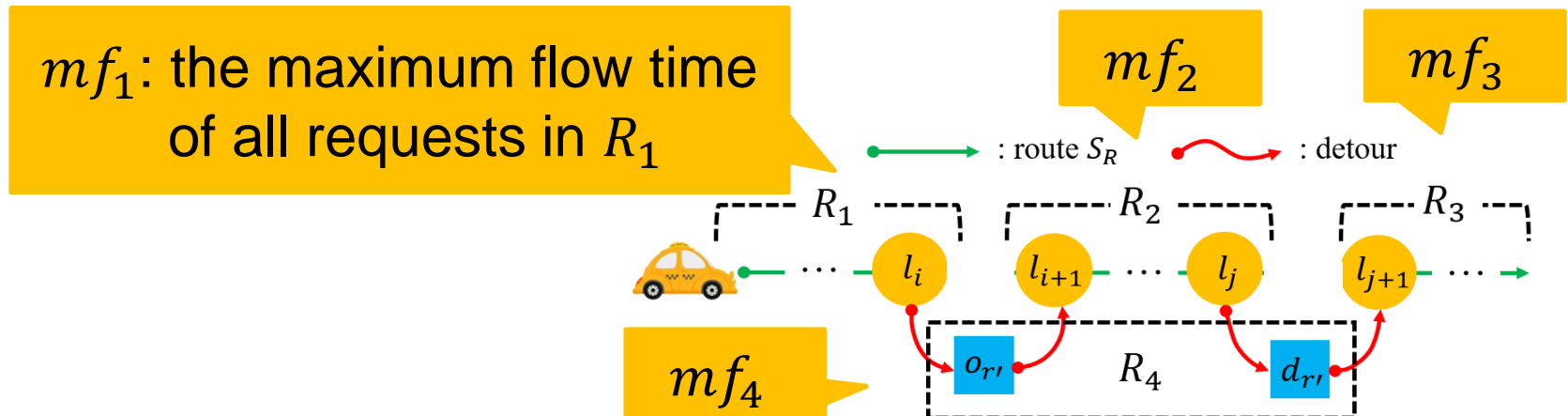
Partition-based Framework

Insertion(i, j)



Partition-based Framework

- Basic idea:
 - Partition the set of requests R^+ into four parts
 - R_1 : requests delivered before i -th location
 - R_2 : requests delivered between i -th and j -th location
 - R_3 : requests delivered after j -th location
 - R_4 : new request r'
 - Objective calculation:
 - $\text{OBJ}(S_{R^+}) = \max\{mf_1, mf_2, mf_3, mf_4\}$



Partition-based Framework

Time complexity: $O(n^2)$

||

Calculate objective and
check constraints in $O(1)$

×

Enumerate possible
insertion pairs in $O(n^2)$

+

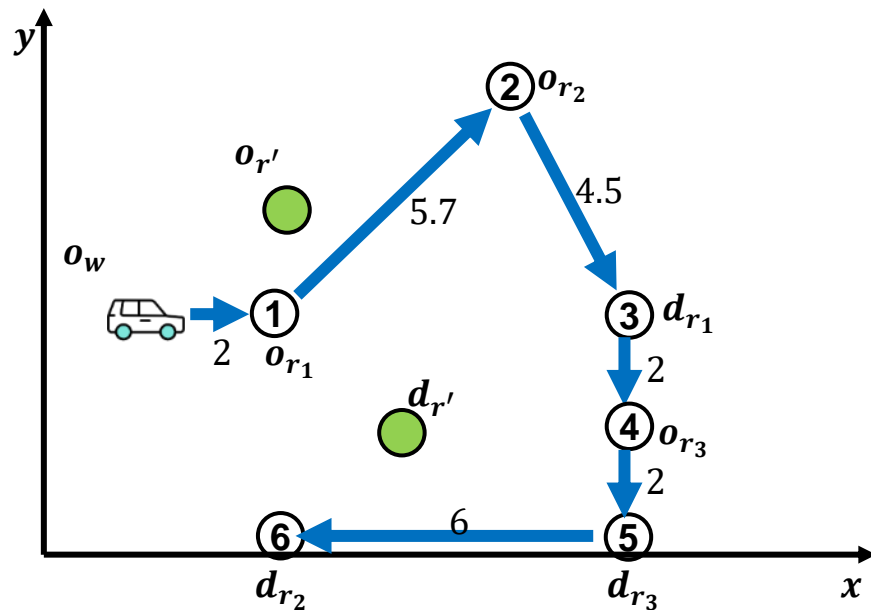
Pre-processing in $O(n^2)$

Pre-processing

- A matrix $mobj$
 - $mobj(i, j)$: the maximum flow time for requests between i -th and j -th location

$$mobj(i, j) = \max\{mobj(i, j-1), \text{flow time of request with destination } l_j\}$$

$O(n^2)$



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

$mobj$

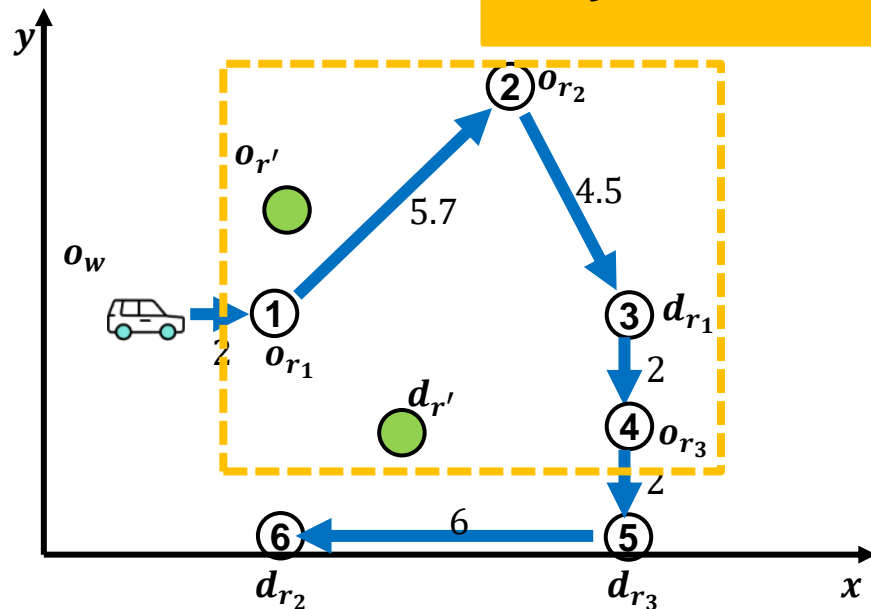
Pre-processing

- A matrix $mobj$
 - $mobj(i, j)$: the maximum flow time for requests between i -th and j -th location

$$mobj(i, j) = \max\{mobj(i, j-1), \text{flow time of request with destination } l_j\}$$

$O(n^2)$

$$mobj(1, 4) = \max\{mobj(1, 3), 0\} = \max\{12.2, 0\} = 12.2$$



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

$mobj$

Partition-based Framework

Time complexity: $O(n^2)$

||

Calculate objective and
check constraints in $O(1)$

×

Enumerate possible
insertion pairs in $O(n^2)$

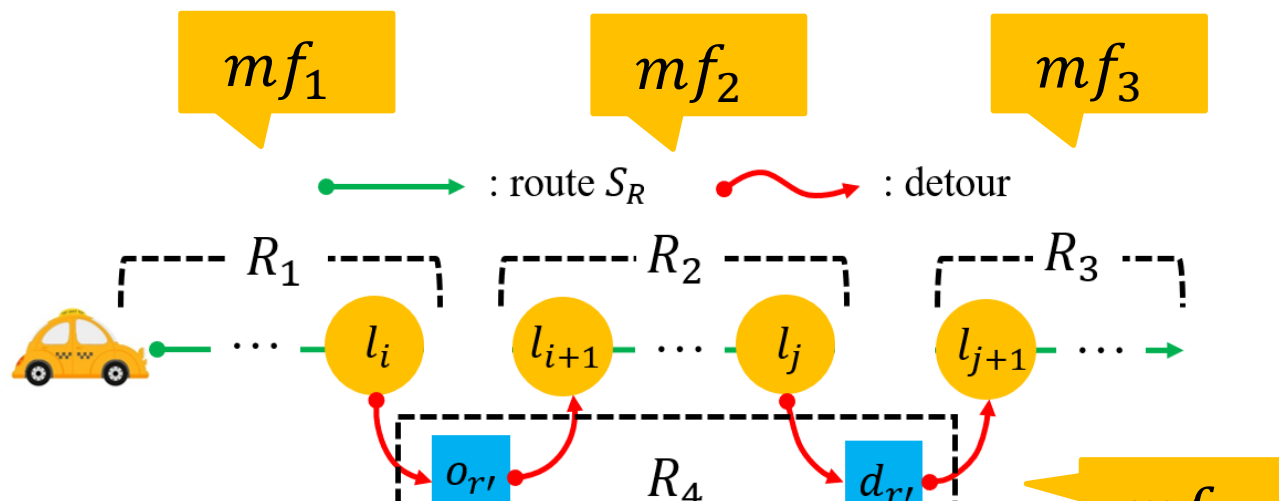
+

Pre-processing in $O(n^2)$

Objective Calculation

- The formulas of mf_1, mf_2, mf_3, mf_4 :

- $mf_1 = mobj(0, i)$ detour of inserting the origin after l_i
- $mf_2 = \det(i, o_{r'}) + mobj(i + 1, j)$ matrix in pre-processing
- $mf_3 = \det(i, o_{r'}) + \det(j, d_{r'}) + mobj(j, n)$
- $mf_4 = arr(j) + \det(i, o_{r'}) + dis(l_j, d_{r'}) + (\alpha - 1)t_{r'}$

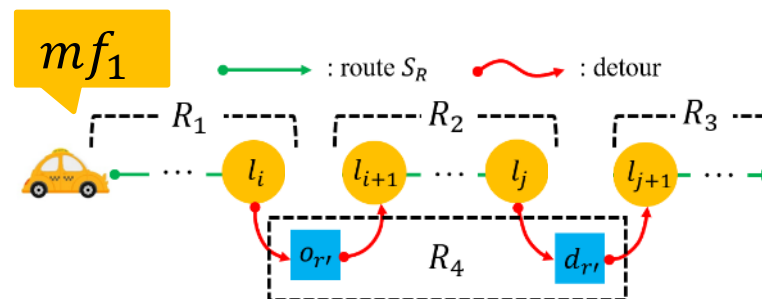
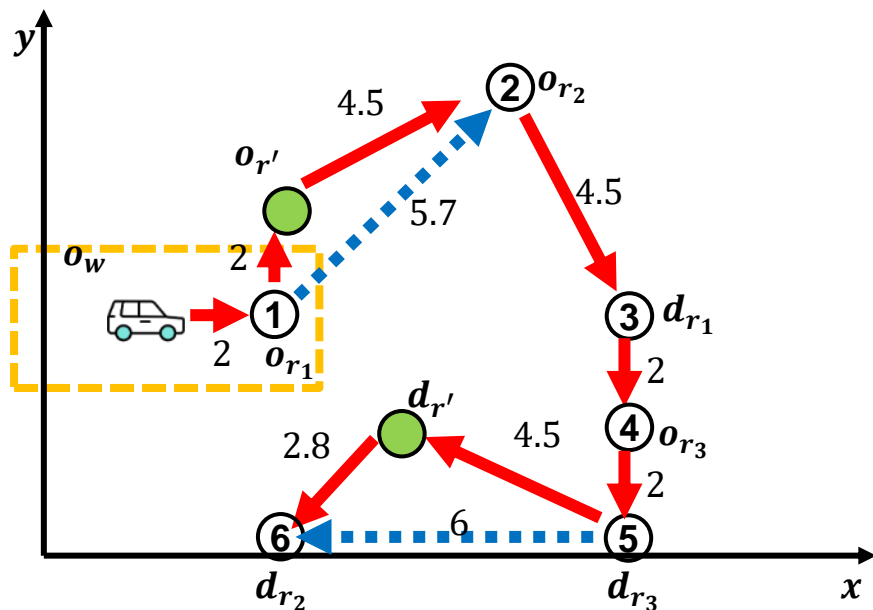


We can calculate the objective in $O(1)$ time with formulas

Objective Calculation

- Insertion (1,5)
 - $mf_1 = mobj(0,1) = 0$

$$R_1 = \emptyset$$



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

$mobj$

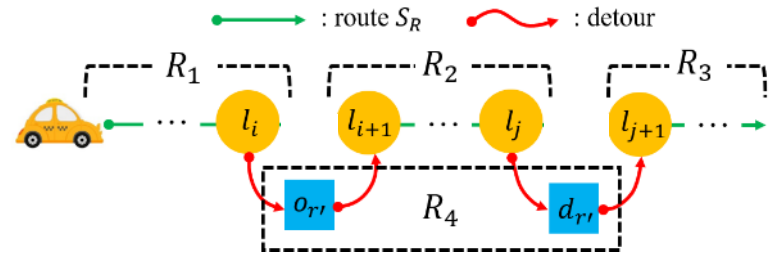
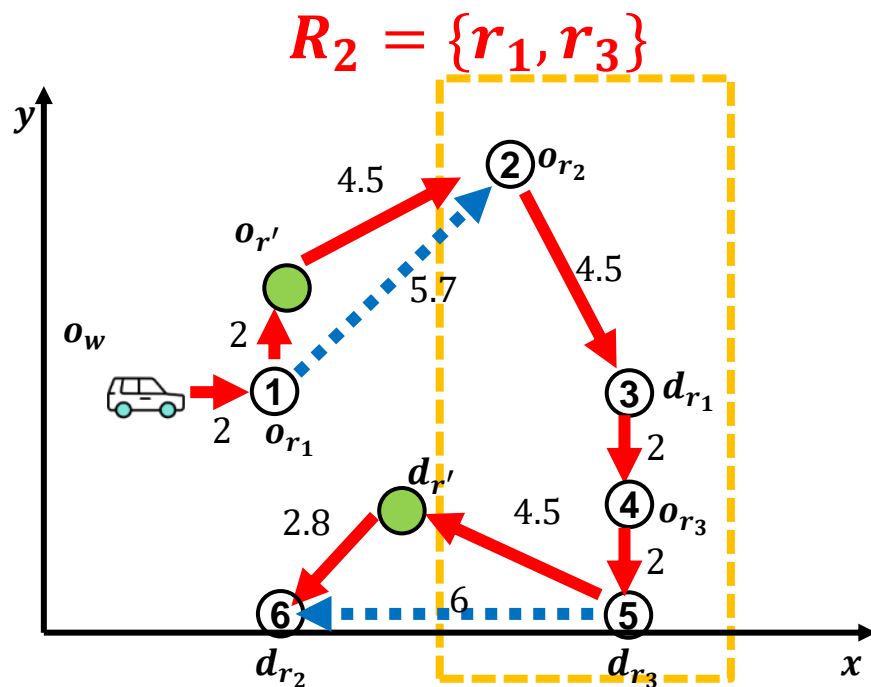
Objective Calculation

- Insertion (1,5)

- $mf_2 = det(1, o_{r'}) + mobj(2,5) = 0.8 + 16.2 = 17$

$det(k, p)$: the detour of inserting p after k -th location

mf_2



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

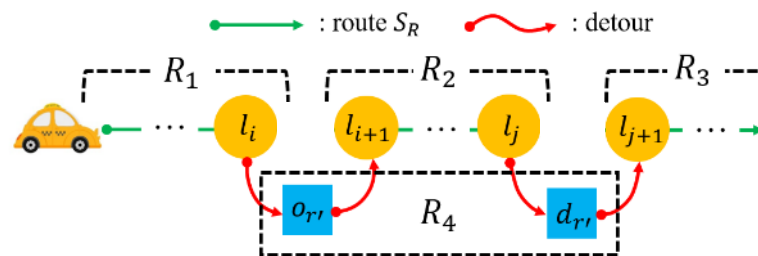
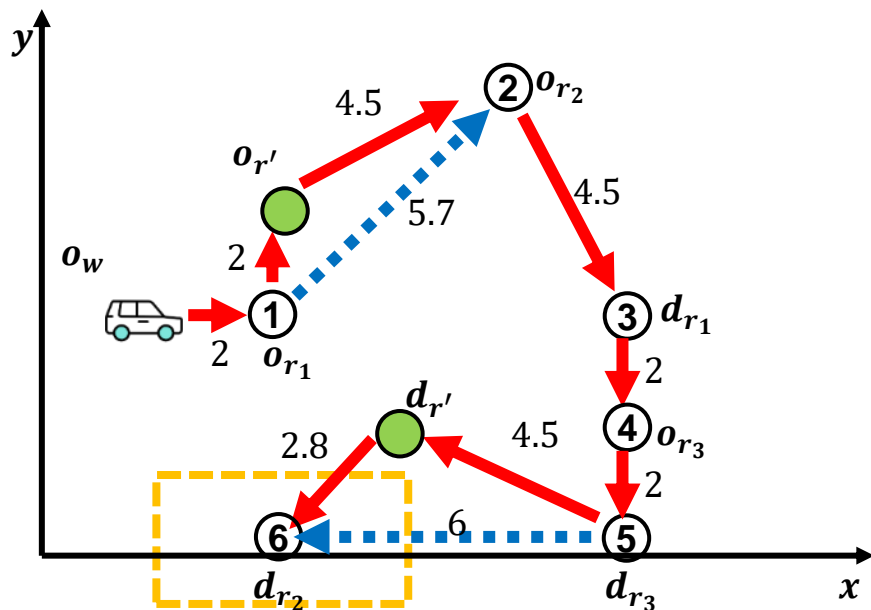
$mobj$

Objective Calculation

- Insertion (1,5)

- $$mf_3 = det(1, o_{r'}) + det(5, d_{r'}) + mobj(6,6) = 0.8 + 1.3 + 22.2 = 24.3$$

$$R_3 = \{r_2\}$$



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

mobj

mf_3

Objective Calculation

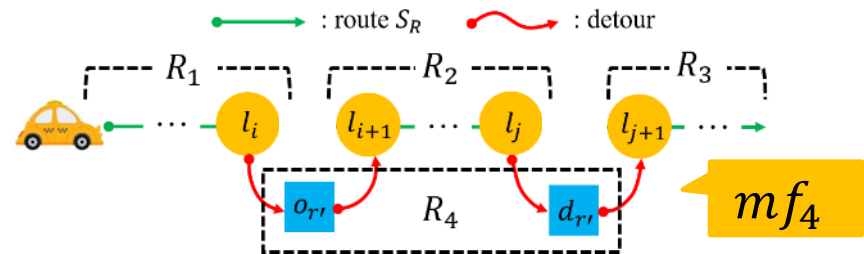
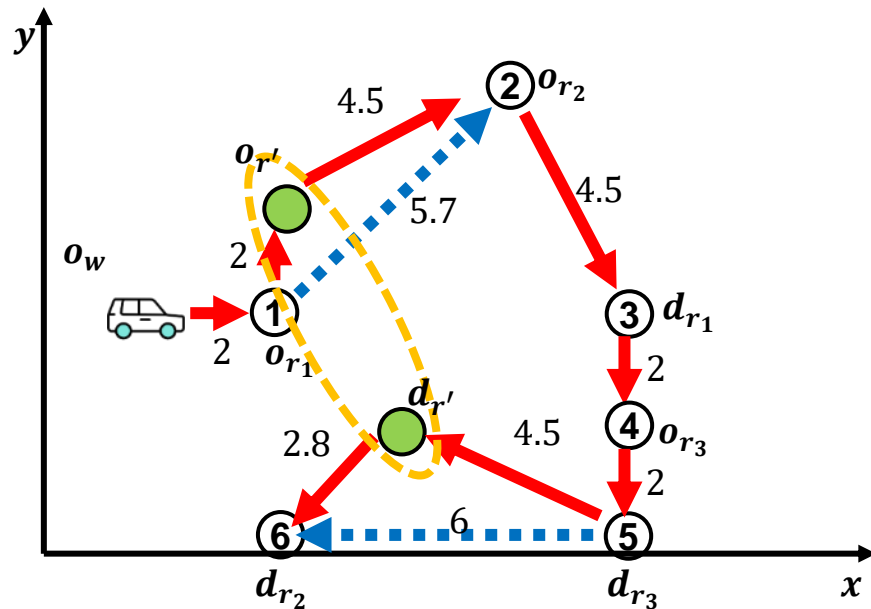
- Insertion (1,5)

- mf_4 is the flow time of r' : $21.5 - 0 = 21.5$

arrival time of $d_{r'}$

release time of r'

$$R_4 = \{r'\}$$

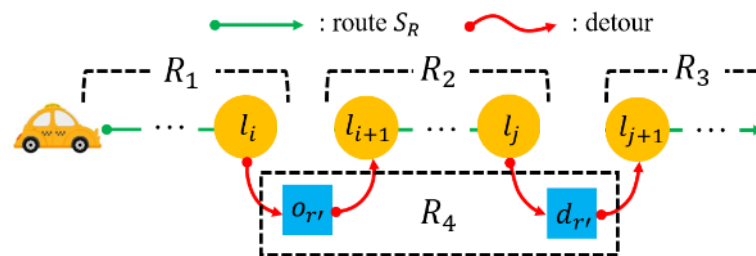
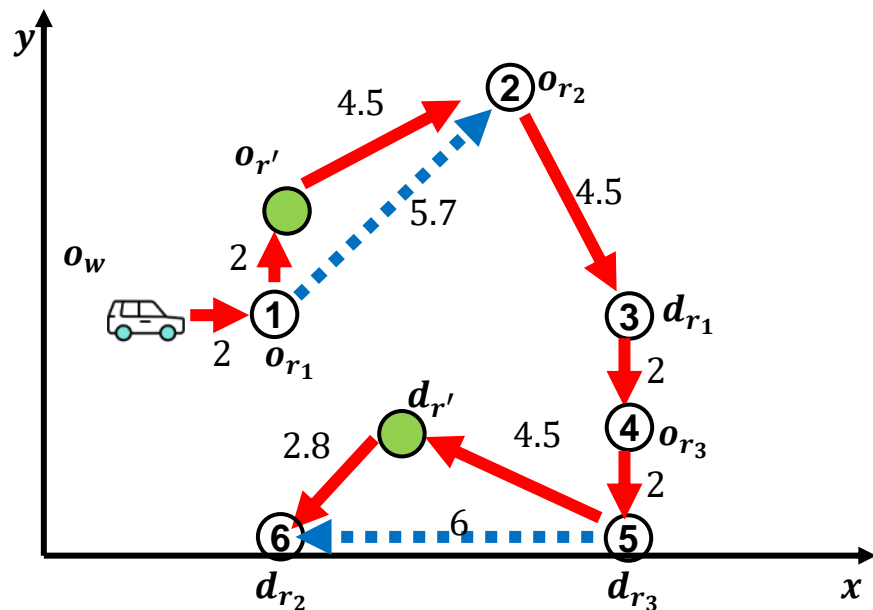


$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

mobj

Objective Calculation

- Insertion (1,5)
 - The answer is $\max(0, 17, 24.3, 21.5) = 24.3$



$i \backslash j$	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

mobj

Partition-based Framework

- Constraint checking:

Capacity constraint:

$$\begin{aligned} pck(i) &\leq c_w - c_{r'} \\ pck(j) &\leq c_w - c_{r'} \end{aligned}$$

Deadline constraint:

$$\begin{aligned} det(i, o_{r'}) &\leq slk(i) \\ det(i, o_{r'}) + det(j, d_{r'}) &\leq slk(j) \\ arr(j) + det(i, o_{r'}) + dis(l_j, d_{r'}) &\leq e_{r'} \end{aligned}$$

Constraint checking takes $O(1)$ time with formulas

Partition-based Framework

Time complexity: $O(n^2)$

||

Calculate objective and
check constraints in $O(1)$

×

Enumerate possible
insertion pairs in $O(n^2)$

+

Pre-processing in $O(n^2)$

Outline

- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

Segment-based DP Algorithm

Time complexity: $O(n^2)$

||

Calculate objective and
check constraints in $O(1)$

×

Enumerate possible
insertion pairs in $O(n^2)$

+

Pre-processing in $O(n^2)$

Partition-based Framework



Time complexity: $O(n)$

||

Find optimal location to
insert destination in $O(1)$

×

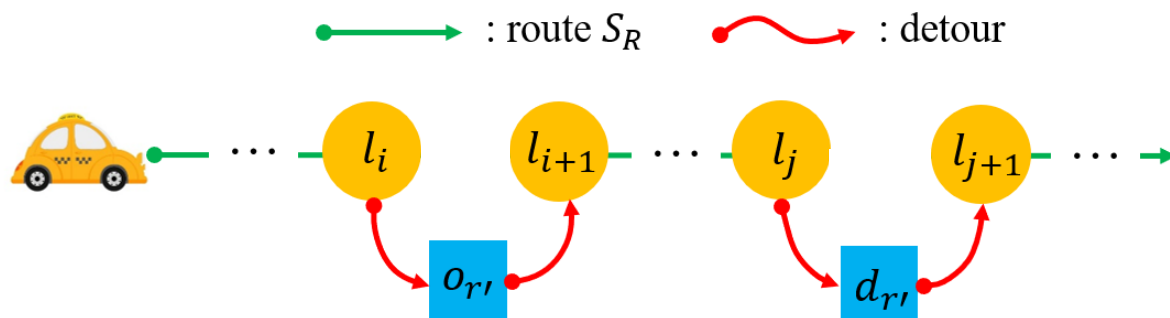
Enumerate locations
to insert origin in $O(n)$

+

Pre-processing in $O(n)$

Segment-based DP Algorithm

Segment-based DP Algorithm

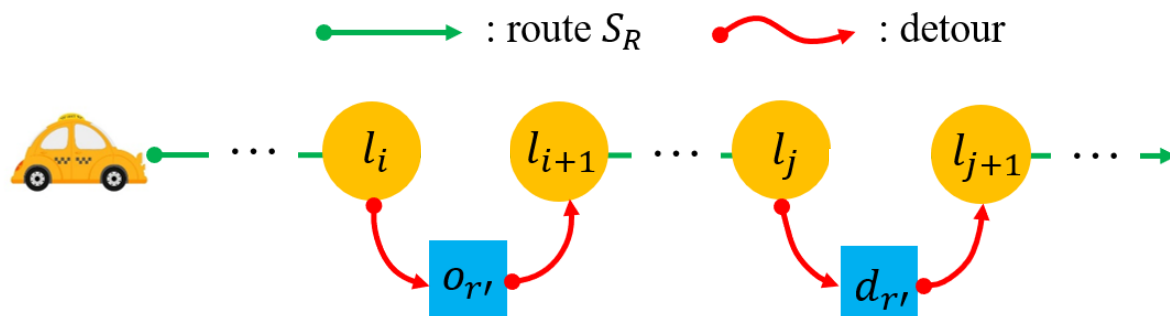


Enumerate i and j

$$O(n^2)$$

Partition-based Framework

Segment-based DP Algorithm



Enumerate i and j

Enumerate i ,
find optimal j^*

$O(n^2)$

Reducing

$O(n)$

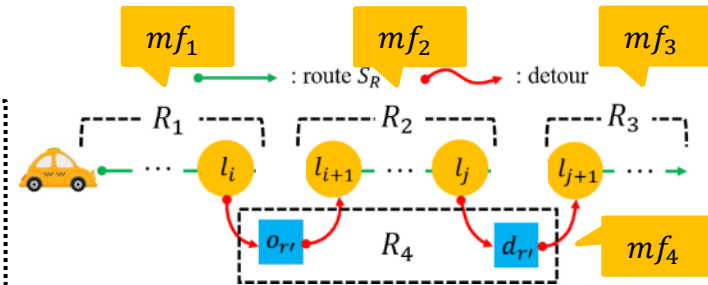
Partition-based Framework

Segment-based DP Algorithm

Segment-based DP Algorithm

Objective Calculation

$$\begin{aligned}
 mf_1 &= mobj(0, i) \\
 mf_2 &= det(i, o_{r'}) + mobj(i+1, j) \\
 mf_3 &= det(i, o_{r'}) + det(j, d_{r'}) + mobj(j, n) \\
 mf_4 &= det(i, o_{r'}) + arr(j) + dis(l_j, d_{r'}) + (\alpha - 1)t_{r'}
 \end{aligned}$$



$$OBJ(S_{R^+}) = \max\{mf_1, mf_2, mf_3, mf_4\}$$

partition i and j

relevant to i

relevant to j

$$OBJ(S_{R^+}) = A(i) + B(j)$$

$$A(i) = det(i, o_{r'})$$

$$B(j) = \max \left\{ \begin{aligned} &det(j, d_{r'}) + mobj(j, n), \\ &arr(j) + dis(l_j, d_{r'}) + (\alpha - 1)t_{r'} \end{aligned} \right\}$$

Segment-based DP Algorithm

- Can $\min_{i,j} OBJ(S_{R^+}) = \min_i A(i) + \min_j B(j)$?

✗ It may violate constraints

Segment-based DP Algorithm

- How to get feasible $\min_{i,j} A(i) + B(j)$?

Segment-based DP Algorithm

- How to get feasible $\min_{i,j} A(i) + B(j)$?
- Method 1:
 - Enumerate i , calculate $A(i)$
 - Enumerate j , calculate $B(j)$ and check feasibility
 - Maintain the minimum and feasible $A(i) + B(j)$ $O(n^2)$

Segment-based DP Algorithm

- How to get feasible $\min_{i,j} A(i) + B(j)$?
- Method 1:
 - Enumerate i , calculate $A(i)$
 - Enumerate j , calculate $B(j)$ and check feasibility
 - Maintain the minimum and feasible $A(i) + B(j)$ $O(n^2)$
- Method 2:
 - Enumerate i , calculate $A(i)$
 - Find feasible and minimum $B(j^*)$ quickly $O(?)$

Segment-based DP Algorithm

- How to get feasible $\min_{i,j} A(i) + B(j)$?
 - Method 1:
 - Enumerate i , calculate $A(i)$
 - Enumerate j , calculate $B(j)$ and check feasibility
 - Maintain the minimum and feasible $A(i) + B(j)$ $O(n^2)$
 - Method 2:
 - Enumerate i , calculate $A(i)$
 - Find feasible minimum $B(j^*)$ directly $O(?)$
- $$B(j^*) = \min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Segment tree
 $O(\log n)$

Fenwick tree (dynamic)
 $O(1)$

$O(n \log n)$

 $O(n)$

Example

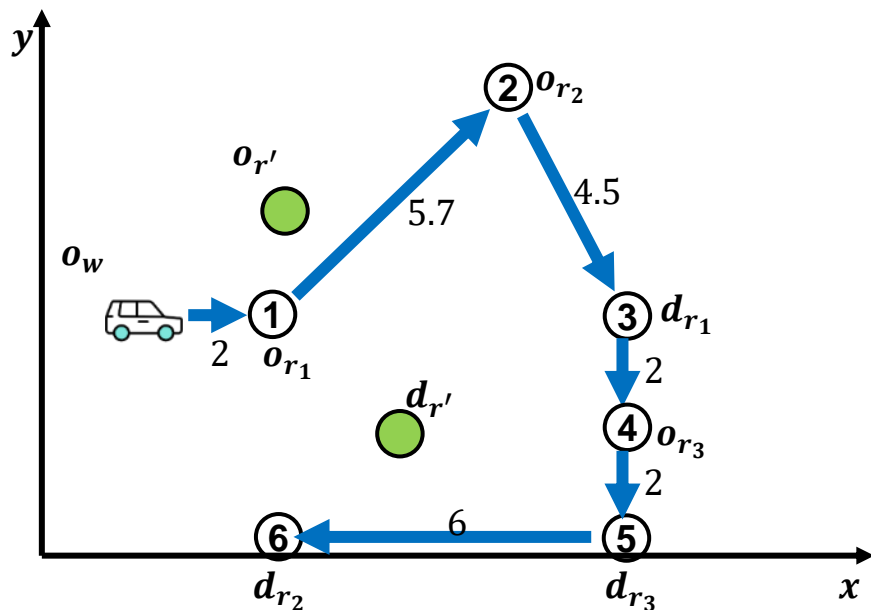
- Initialization

- Initialize $thr(\cdot)$, $B(\cdot)$

objective relevant to j

index	threshold of j			$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$



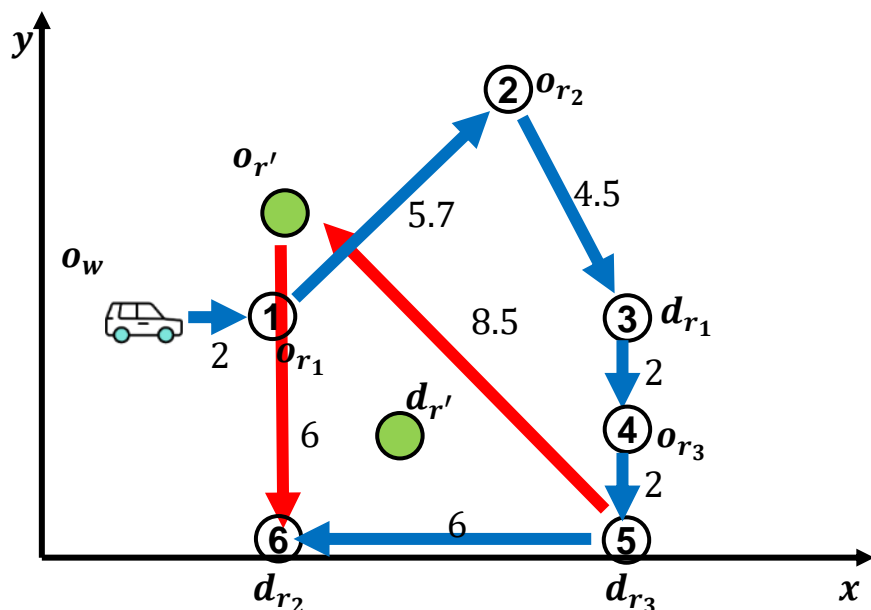
Example

- Enumerate $i = 5$ ($A(i) = 8.5$)
 - Update $B(6) = 25$ at point $thr(6) = -1$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

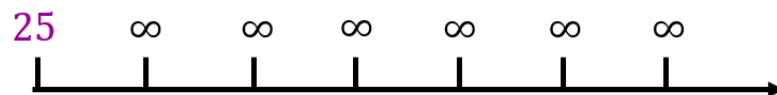
Optimal solution: ∞



Update value of this point as 25

$i = 5$

$par(\cdot)$



$thr(\cdot)$

-1 3.3 4.5 5.5 5.8 6.3 7.4

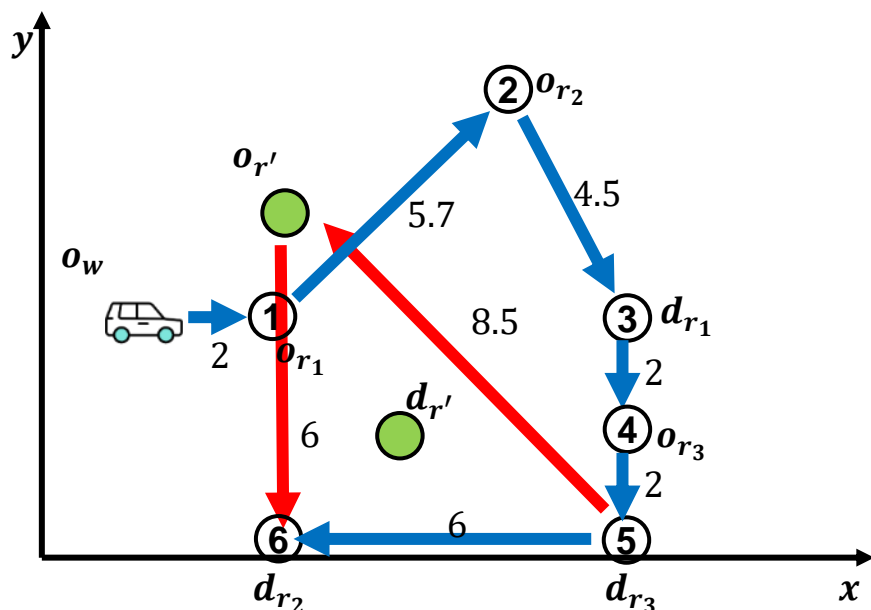
Example

- Enumerate $i = 5$ ($A(i) = 8.5$)
 - Query the segment $[det(5, o_{r'}), \infty)$

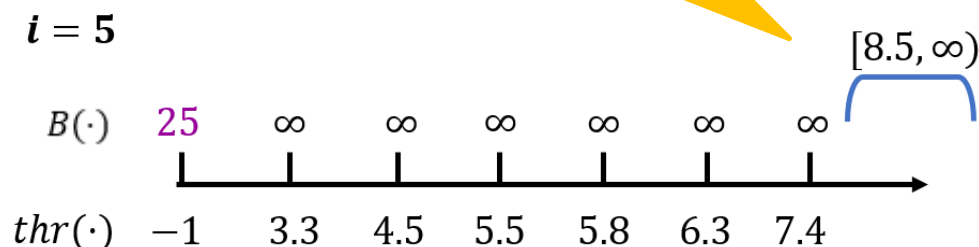
index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



$det(5, o_{r'}) = 8.5$
The answer is ∞



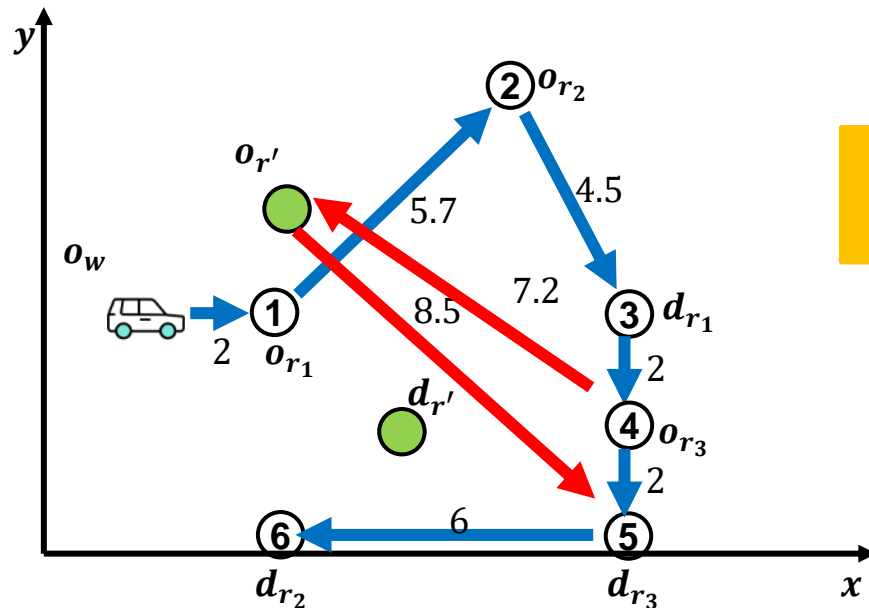
Example

- Enumerate $i = 4$ ($A(i) = 13.7$)
 - Update $B(5) = 23.5$ at point $thr(5) = 3.3$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



Update value of this point as 23.5

$i = 4$



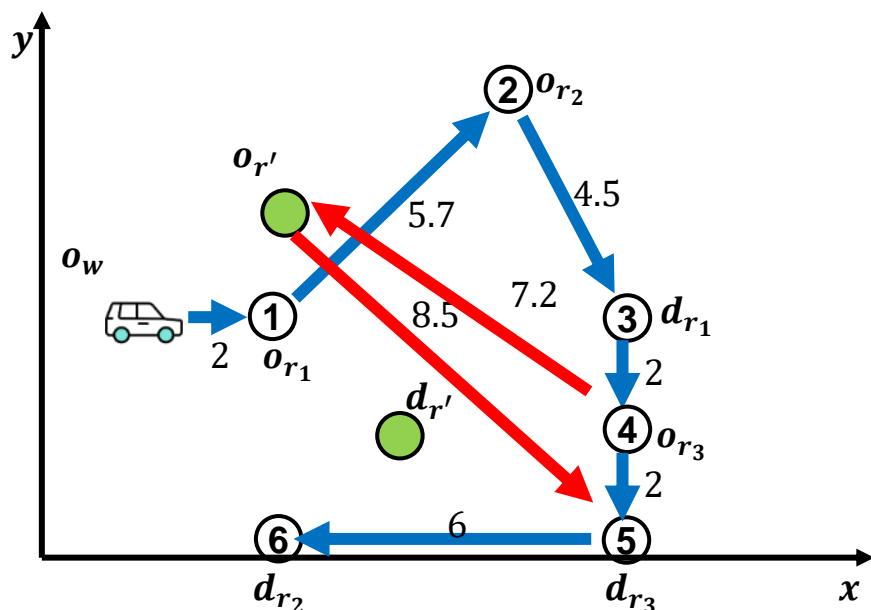
Example

- Enumerate $i = 4$ ($A(i) = 13.7$)
 - Query the segment $[det(4, o_{r'}), \infty)$

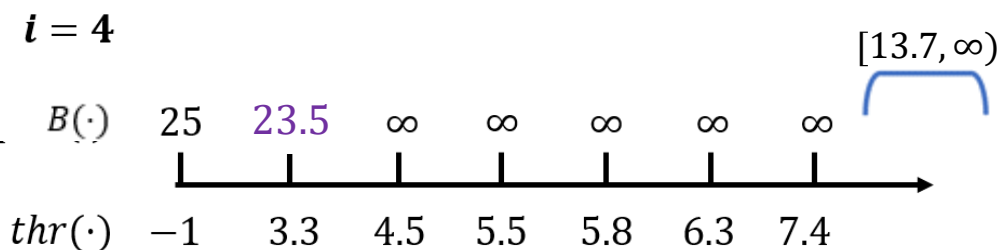
index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



$det(4, o_{r'}) = 13.7$
The answer is ∞



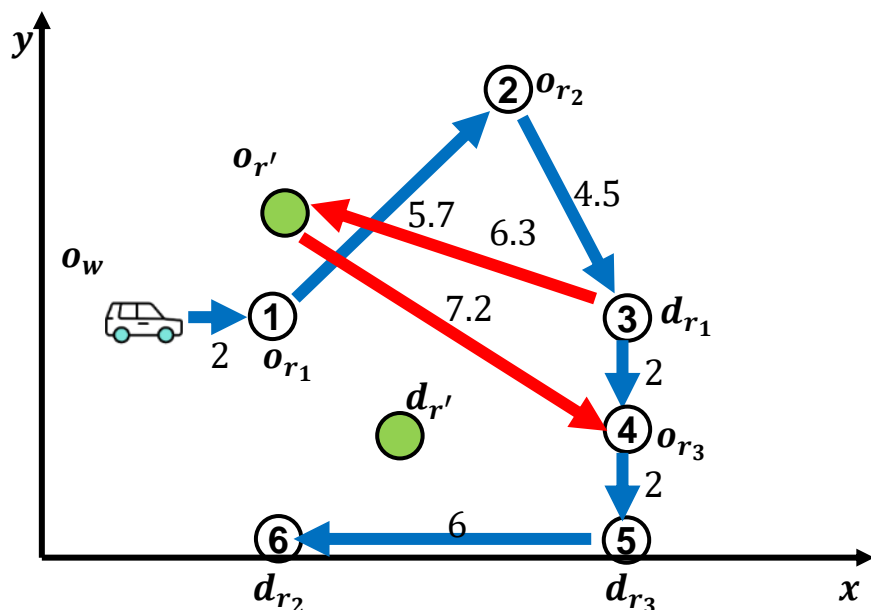
Example

- Enumerate $i = 3$ ($A(i) = 11.5$)
 - Update $B(4) = 28.7$ at point $thr(4) = 5.8$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



Update value of this point as 28.7

$i = 3$



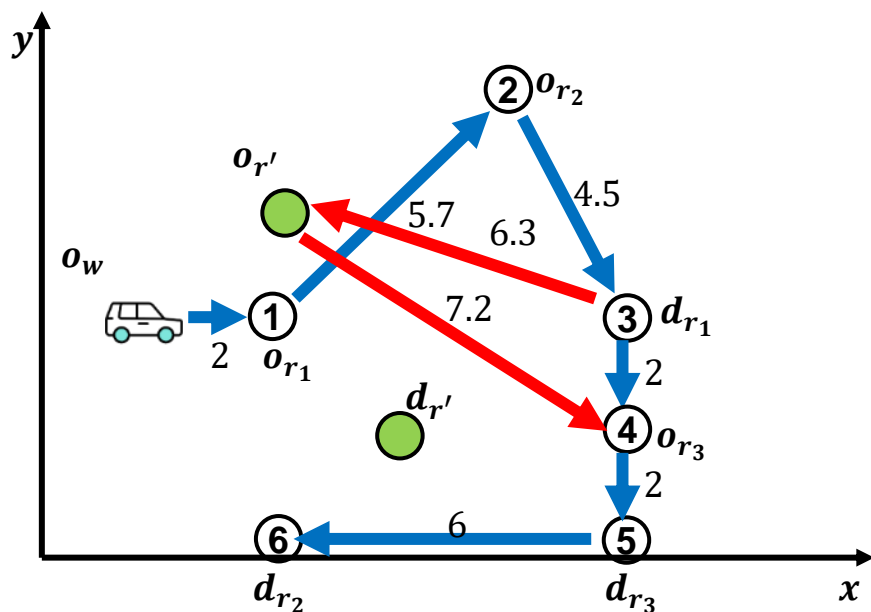
Example

- Enumerate $i = 3$ ($A(i) = 11.5$)
 - Query the segment $[det(3, o_{r'}), \infty)$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

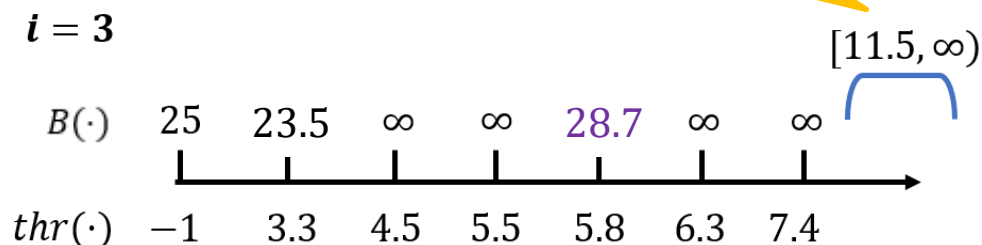
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



$$det(3, o_{r'}) = 11.5$$

The answer is ∞



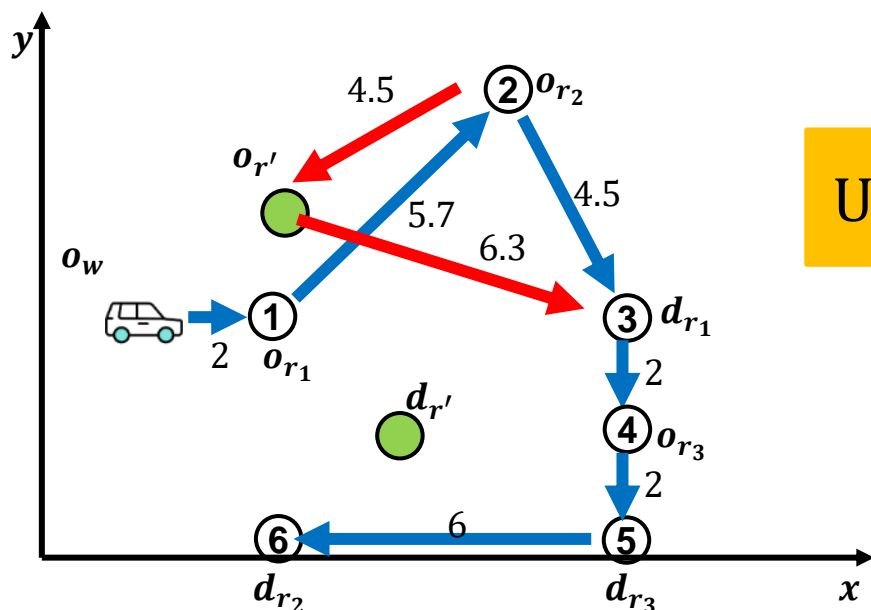
Example

- Enumerate $i = 2$ ($A(i) = 6.3$)
 - Update $B(3) = 28.7$ at point $thr(3) = 6.3$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

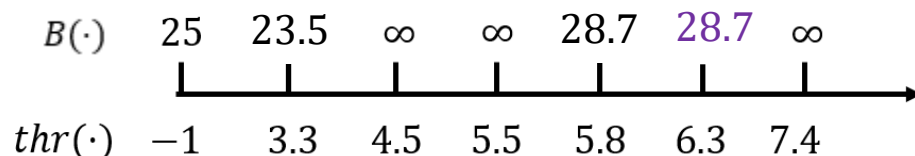
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



Update value of this point as 28.7

$i = 2$



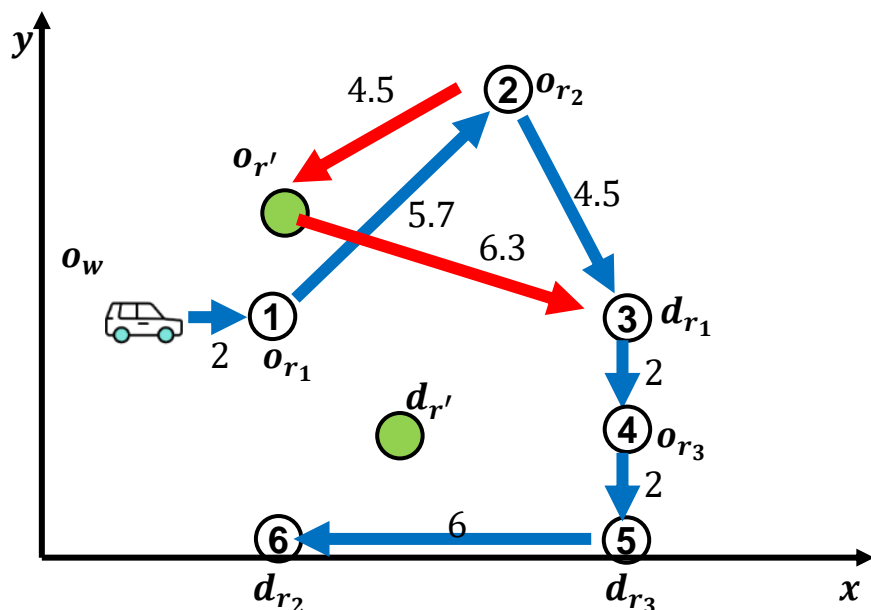
Example

- Enumerate $i = 2$ ($A(i) = 6.3$)
 - Query the segment $[det(2, o_{r'}), \infty)$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

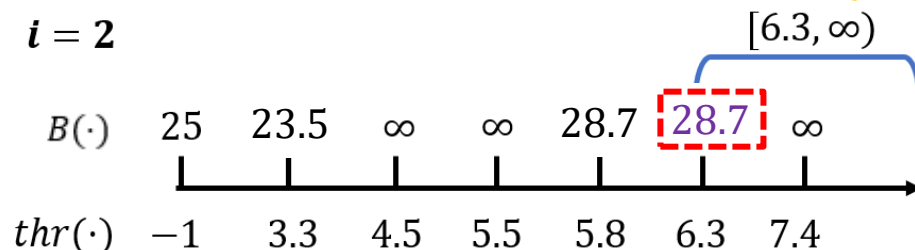
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: ∞



$det(2, o_{r'}) = 6.3$
The answer is 28.7

$i = 2$



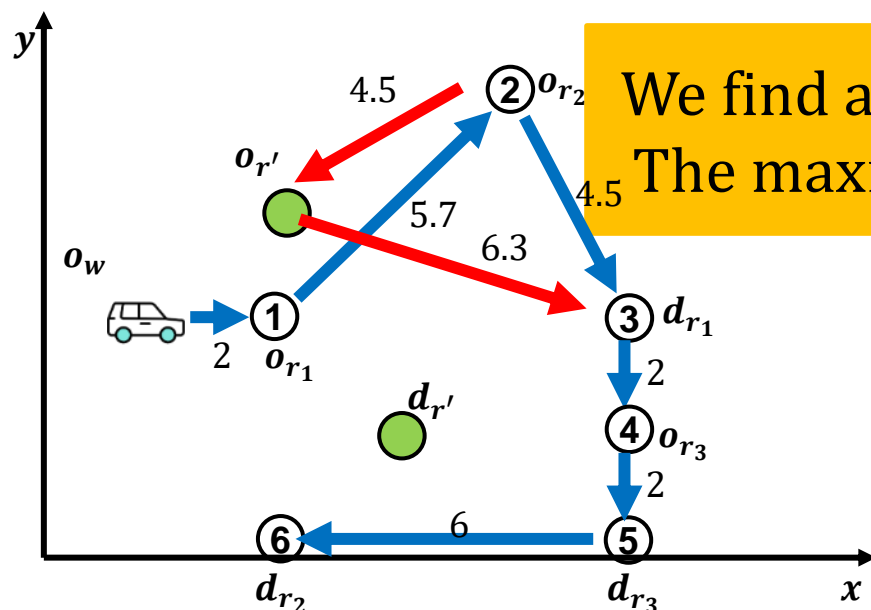
Example

- Enumerate $i = 2$ ($A(i) = 6.3$)
 - Query the segment $[det(2, o_{r'}), \infty)$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

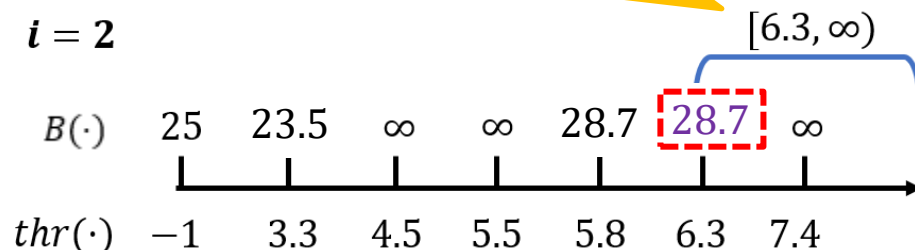
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 35



We find a feasible solution: insertion(2,3)
The maximum flow time is $6.3 + 28.7 = 35$

$i = 2$



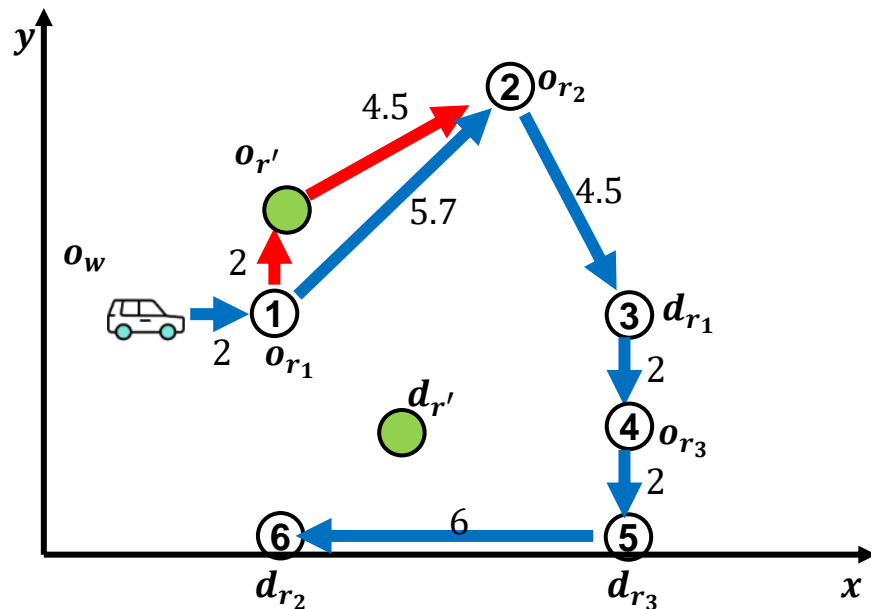
Example

- Enumerate $i = 1$ ($A(i) = 0.8$)
 - Update $B(2) = 28.5$ at point $thr(2) = 4.5$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

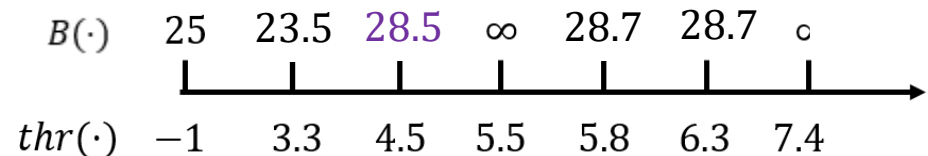
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 35



Update value of this point as 28.5

$i = 1$



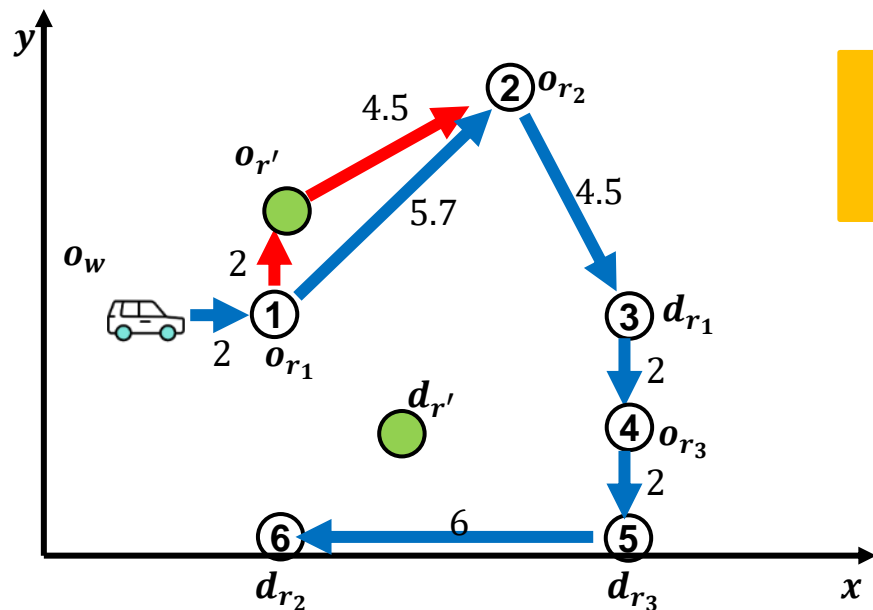
Example

- Enumerate $i = 1$ ($A(i) = 0.8$)
 - Query the segment $[det(1, o_{r'}), \infty)$

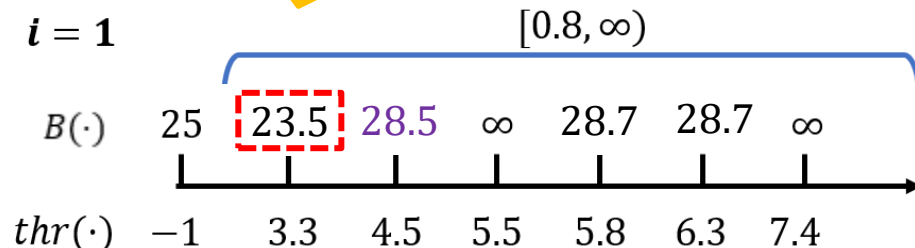
index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 35



$det(1, o_{r'}) = 0.8$
The answer is 25.5



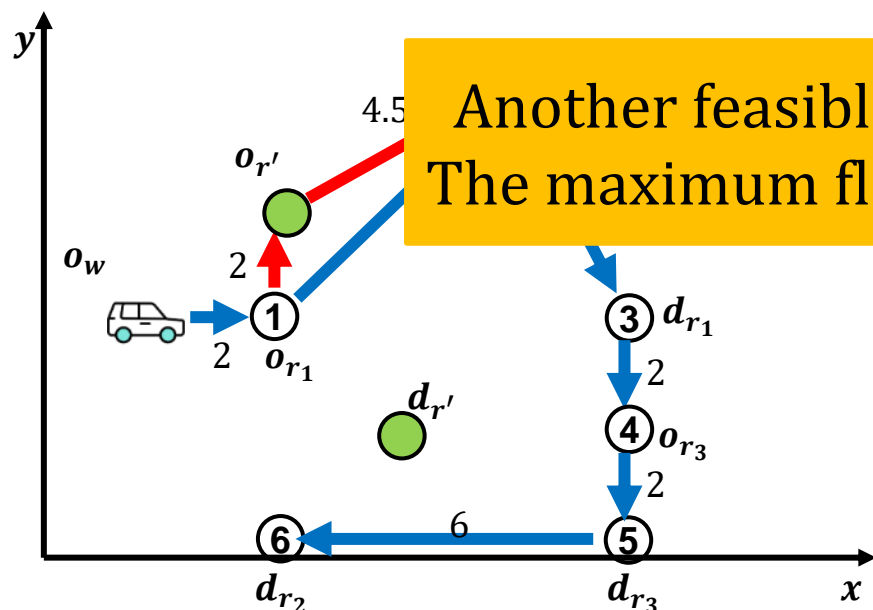
Example

- Enumerate $i = 1$ ($A(i) = 0.8$)
 - Query the segment $[det(1, o_{r'}), \infty)$

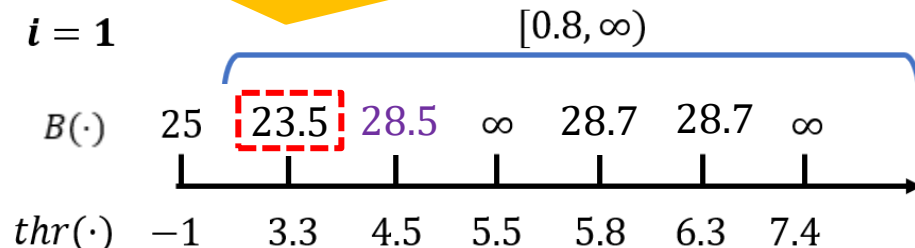
index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 24.3



Another feasible solution: insertion(1,5)
The maximum flow time is $0.8 + 23.5 = 24.3$



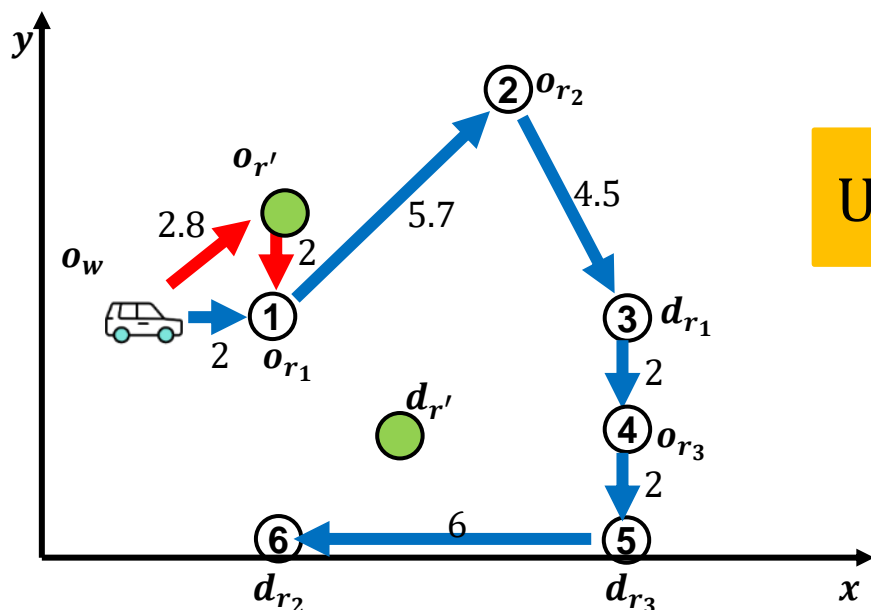
Example

- Enumerate $i = 0$ ($A(i) = 2.8$)
 - Update $B(1) = 25.6$ at point $thr(1) = 7.4$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

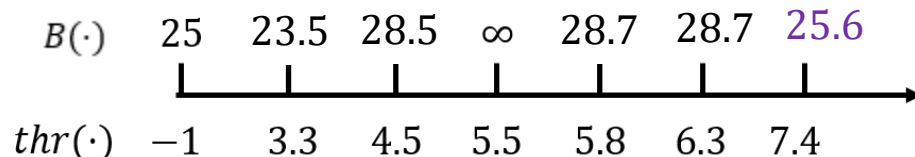
$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 24.3



Update value of this point as 25.6

$i = 0$



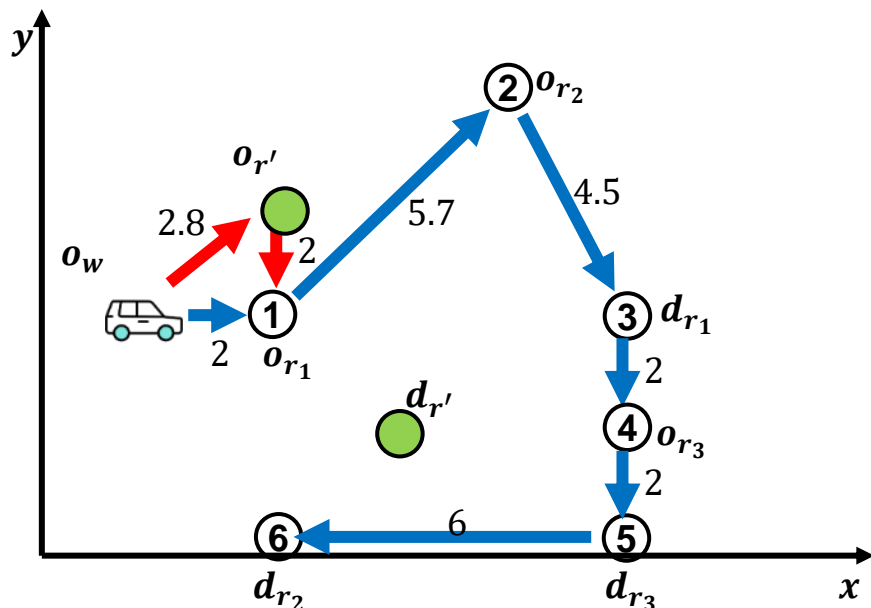
Example

- Enumerate $i = 0$ ($A(i) = 2.8$)
 - Query the segment $[det(0, o_{r'}), \infty)$

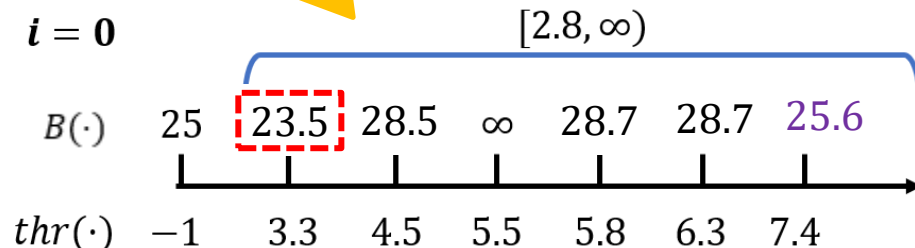
index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 24.3



$det(0, o_{r'}) = 2.8$
The answer is 23.5



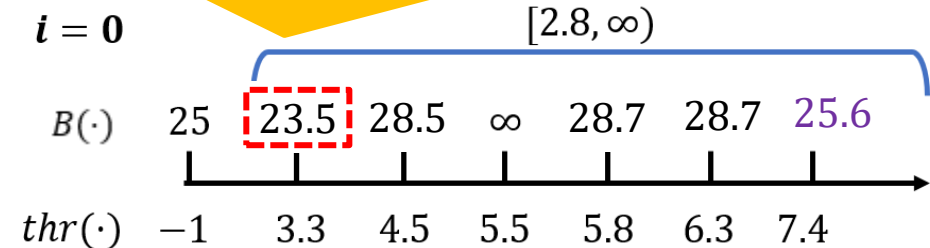
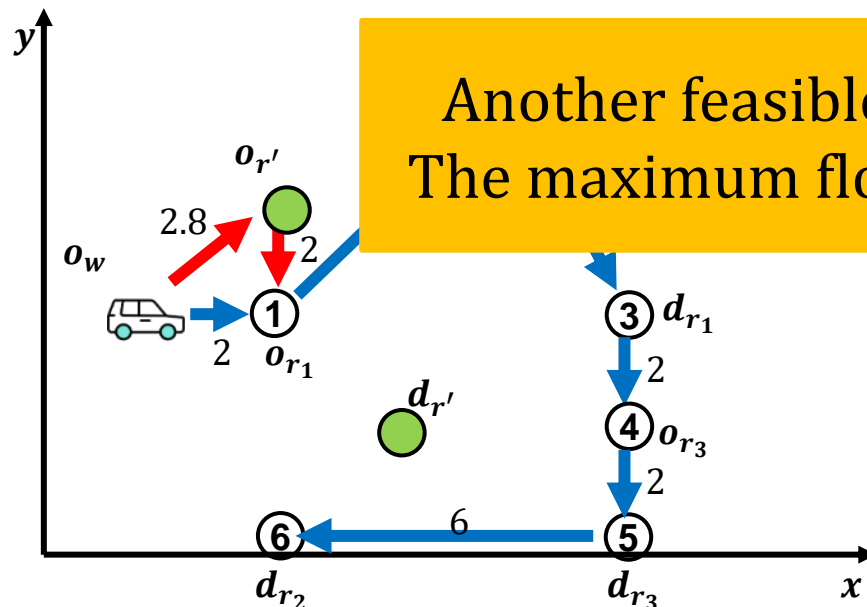
Example

- Enumerate $i = 0$ ($A(i) = 2.8$)
 - Query the segment $[det(0, o_{r'}), \infty)$

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$

Optimal solution: 24.3

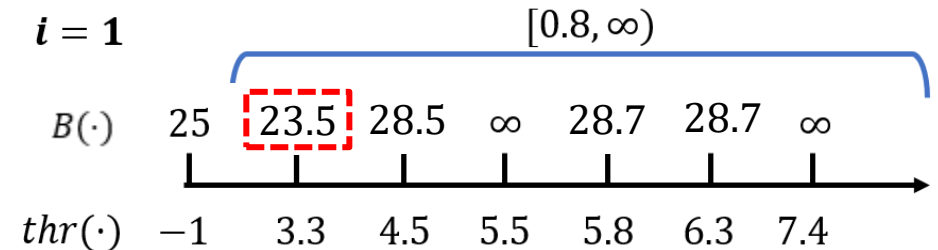
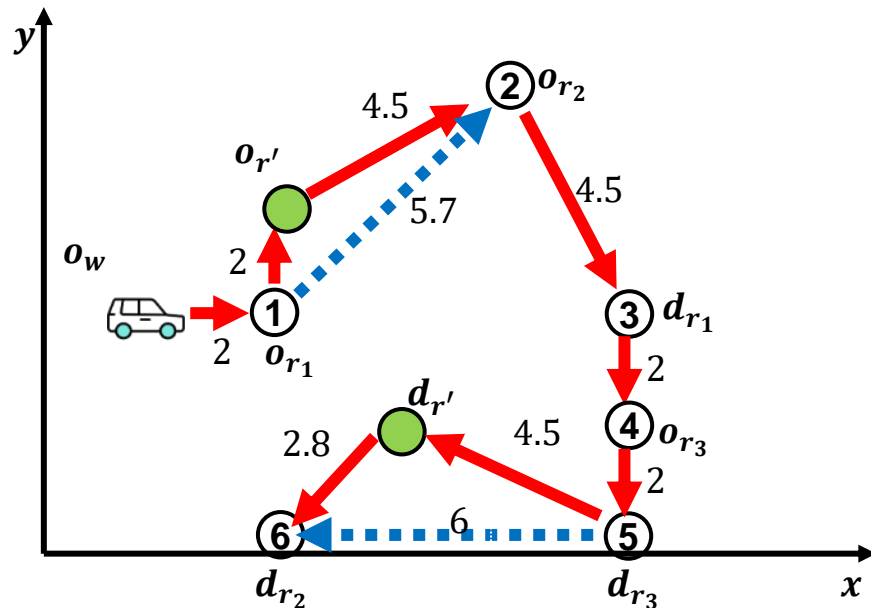


Example

- Finally, the optimal solution:
 - Insertion(1,5) with maximum flow time 24.3

index	0(o_w)	1(o_{r_1})	2(o_{r_2})	3(d_{r_1})	4(o_{r_3})	5(d_{r_3})	6(d_{r_2})
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

$$\min_{\substack{i < j < brk(i) \\ det(i, o_{r'}) \leq thr(j)}} B(j)$$



Outline

- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experimental Evaluations
- Conclusion

Experiments: Setup

- Two real datasets:
 - Taxi: collected in New York City, public dataset.
 - 517,100 requests
 - Worker's capacity is small (Default: 4)
 - Logistics: collected in Shanghai by Cainiao, an urban logistics platform in China.
 - 345,849 requests
 - Worker's capacity is large (Default: 120)

Experiments: Setup

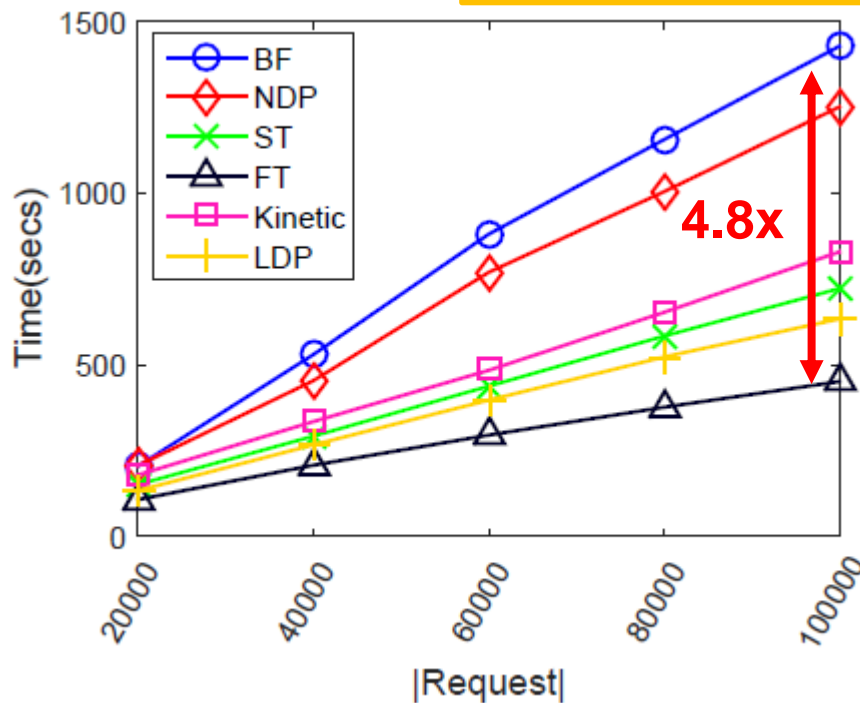
- Compared Algorithms:

Method		Complexity	Goal	
			Minimize total travel time	Minimize maximum flow time
Previous method	BF [ICDE'13] [EJOR'11]	$O(n^3)$	✓	✓
	Kinetic [VLDB'14]	$O(n^2)$	✓	✗
	LDP [VLDB'18]	$O(n)$	✓	✗
Our method	NDP	$O(n^2)$	✓	✓
	ST	$O(n \log n)$	✓	✓
	FT	$O(n)$	✓	✓

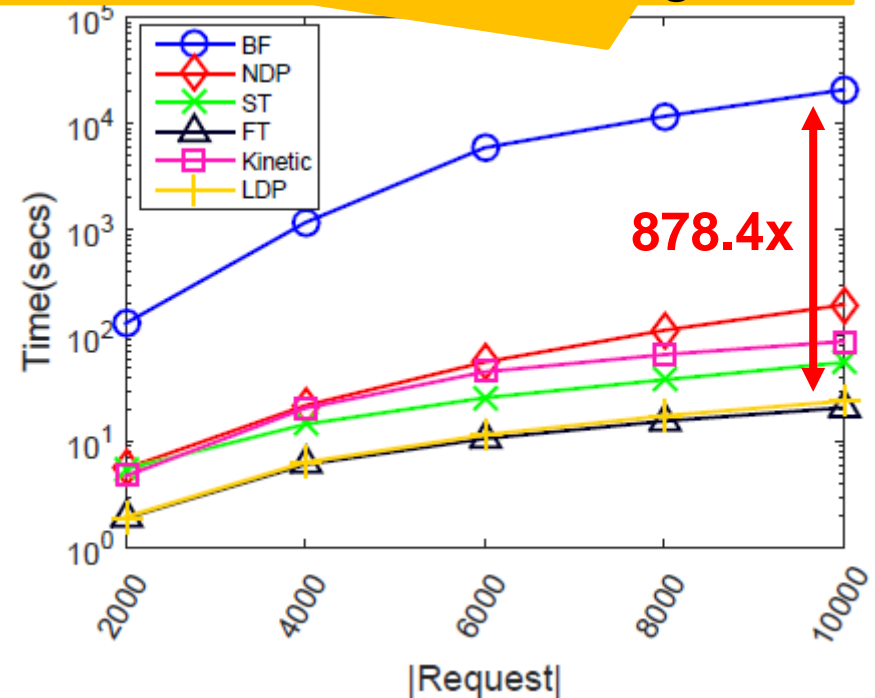
Experiments: Result

- Minimizing total travel time
 - FT is faster than LDP on Taxi and as fast as LDP on Logis

As for minimizing total travel time,
FT is 878.4 times faster than BF on Logistics



Dataset: Taxi



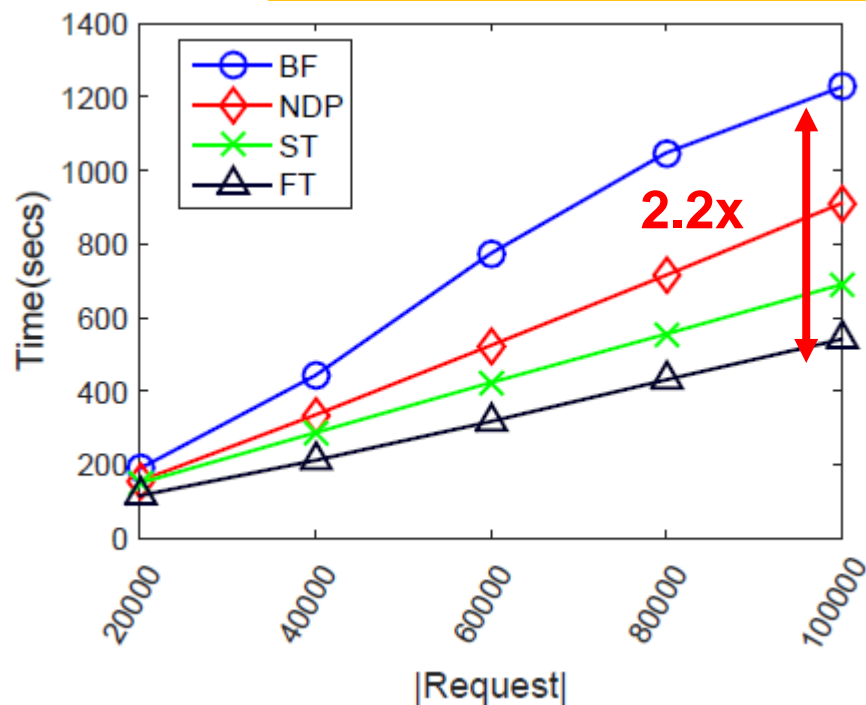
Dataset: logistics

Experiments: Result

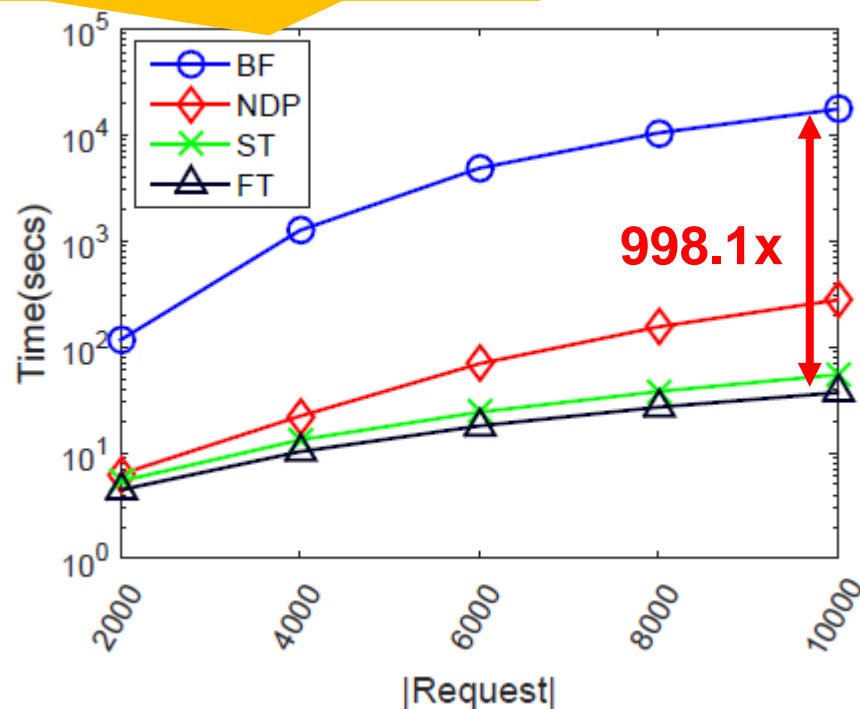
- Minimizing maximum flow time

- FT outperforms the other algorithms

FT is 2.2 times faster than BF on Taxi
and 998.1 times faster on Logistics



Dataset: Taxi



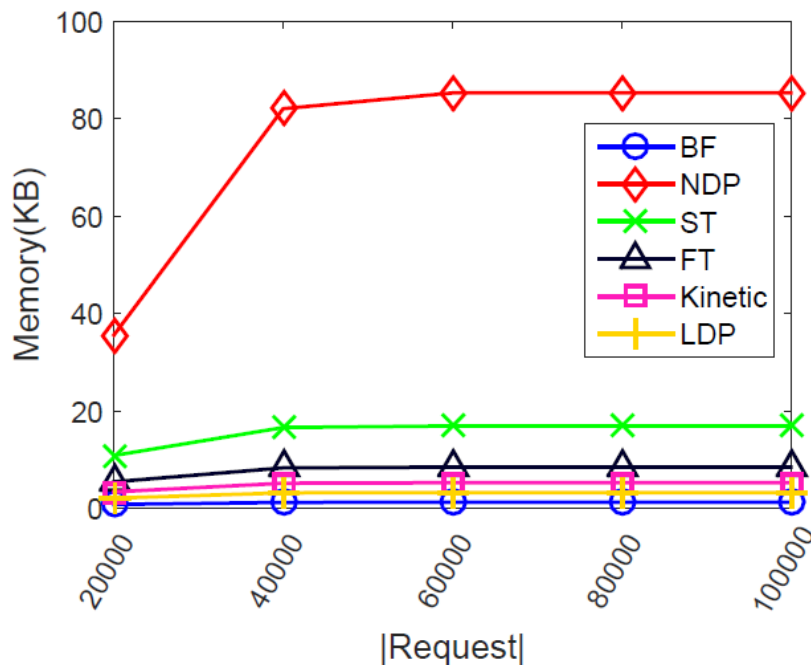
Dataset: logistics

Experiments: Result

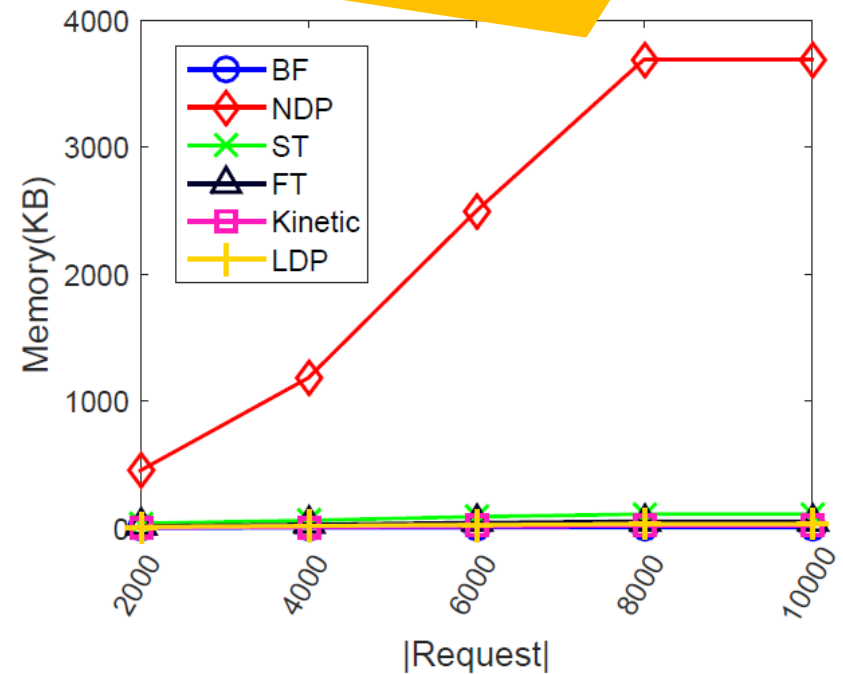
Memory

- The gap of memory cost among algorithms

(except The gap of memory cost between FT to other algorithms is less than 0.1MB



Dataset: Taxi

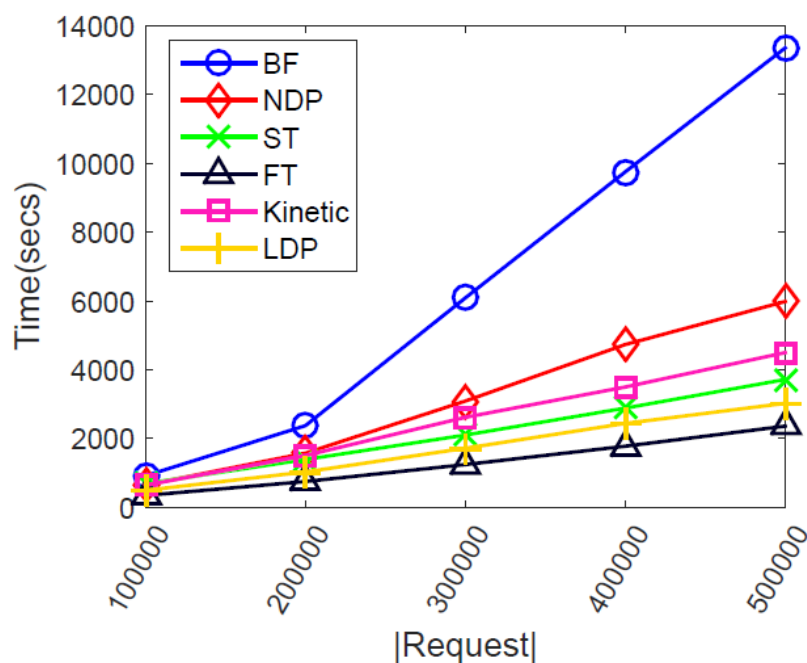


Dataset: logistics

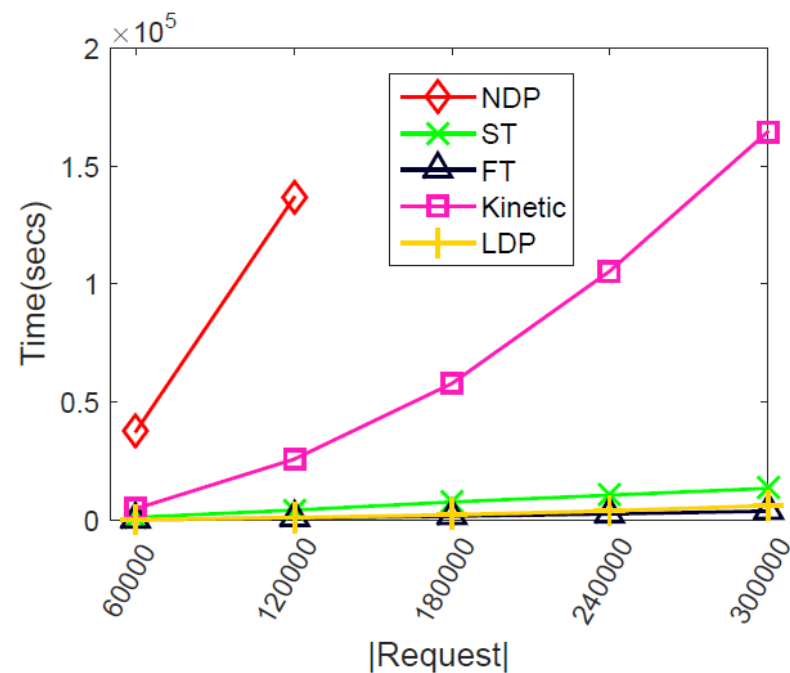
Experiments: Result

Scalability

- Our algorithms are fit for hundred thousands of requests.



Dataset: Taxi



Dataset: logistics

Outline

- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

Conclusion

- We **propose a partition-based framework** to reduce the time complexity of the generic insertion operator from $O(n^3)$ to $O(n^2)$.
- By utilizing some efficient index structures, we further **propose a linear insertion operator**.
- Experiments on real datasets show that the insertion operator can be **accelerated by up to 998.1 times** on urban-scale datasets.

Q & A



**Thank
You**