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

Outline

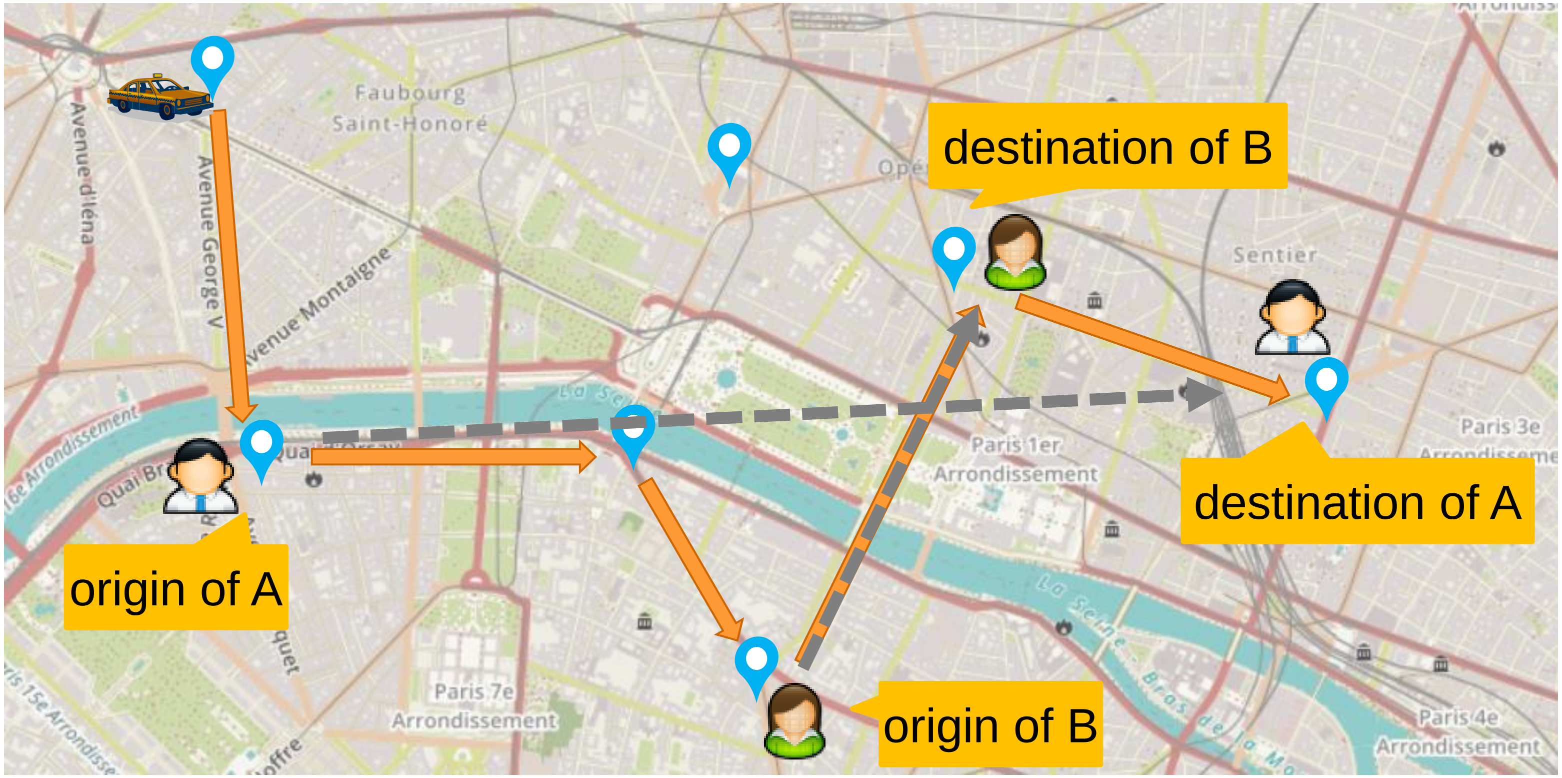
- Background
- Problem Statement
- Partition-based Framework
- Segment-based DP Algorithm
- Experiments
- Conclusion

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**,



DiDi

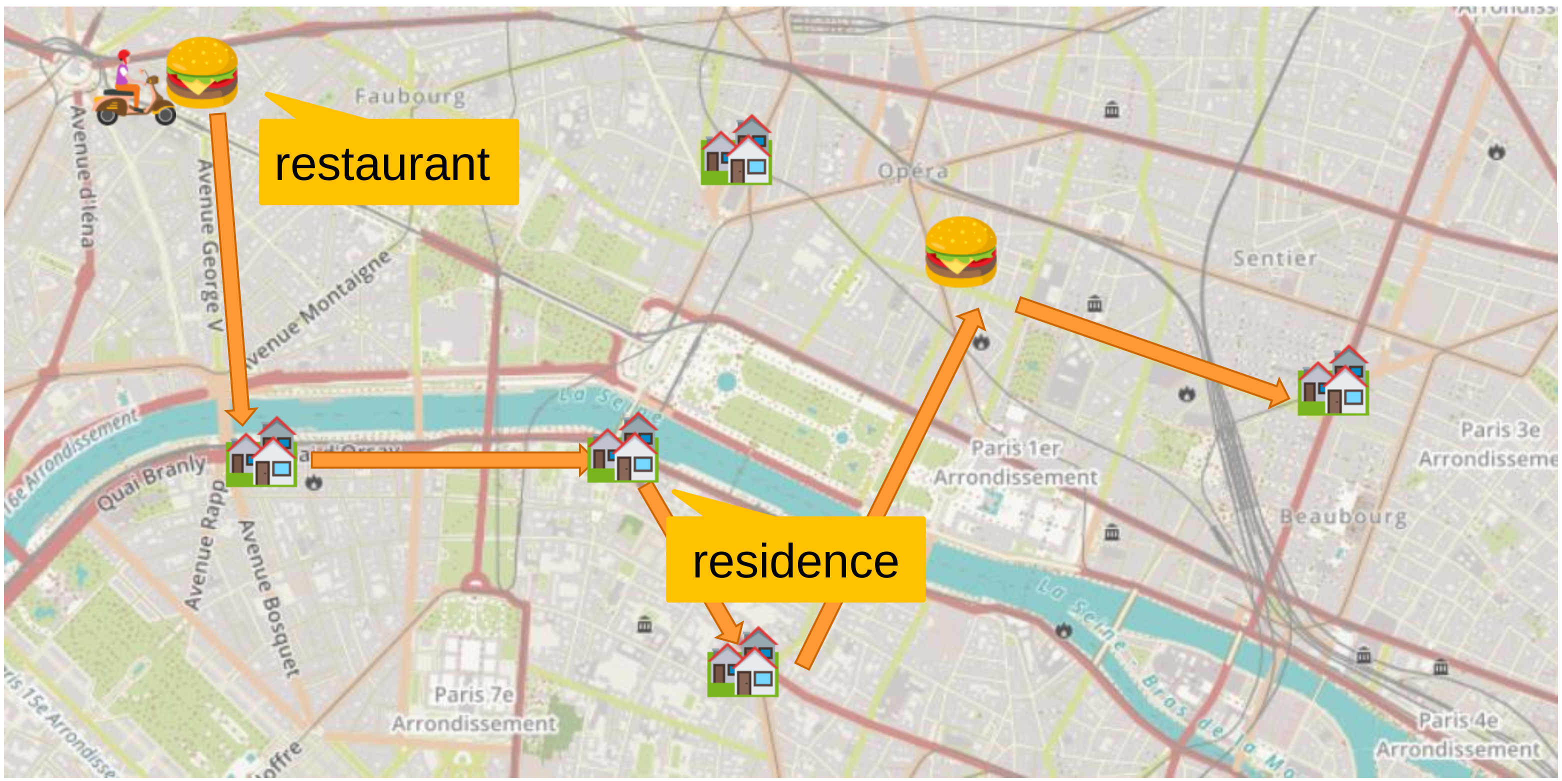
UBER

lyft

via

Dynamic Ridesharing Services

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



Delivery Hero

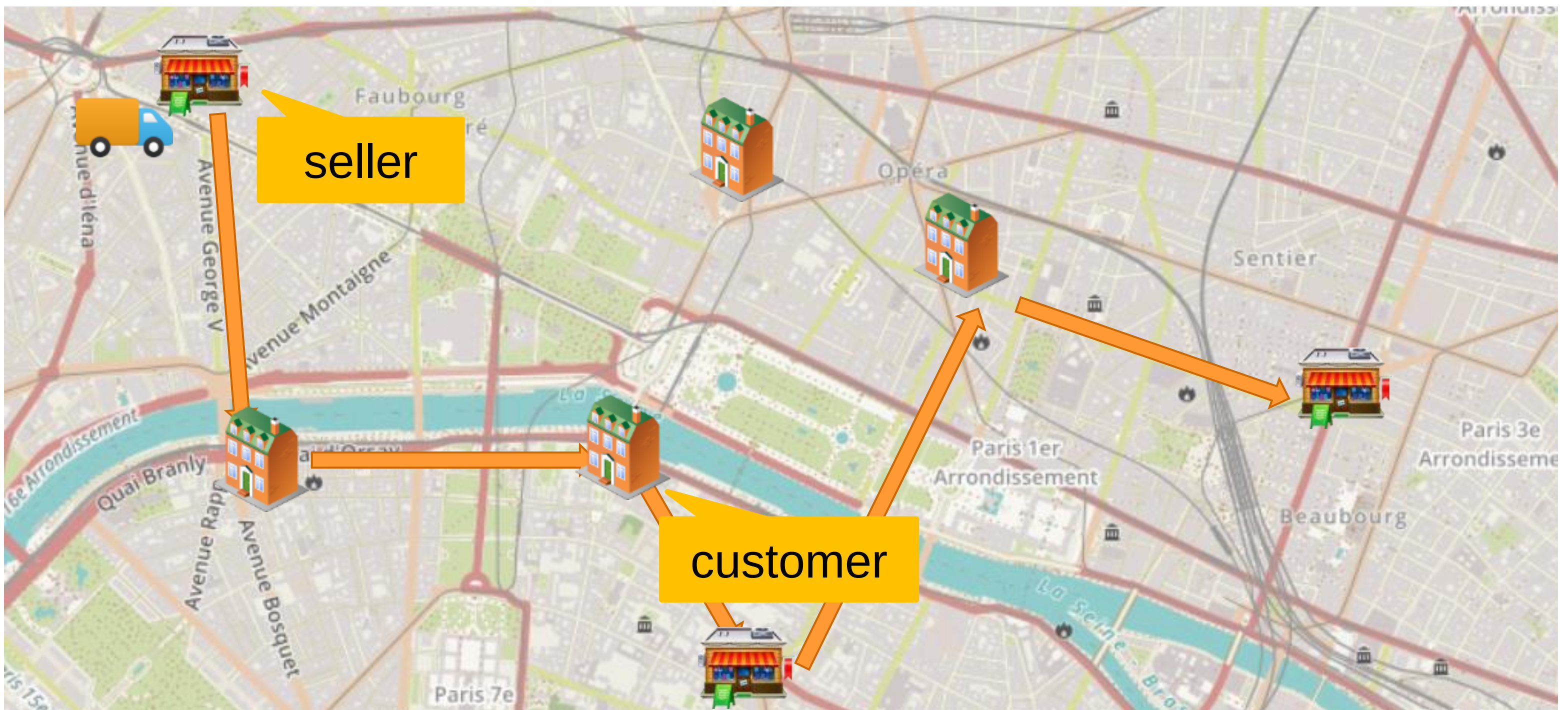
seamless

口碑 Koubei

美团 meituan.com

Dynamic Ridesharing Services

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

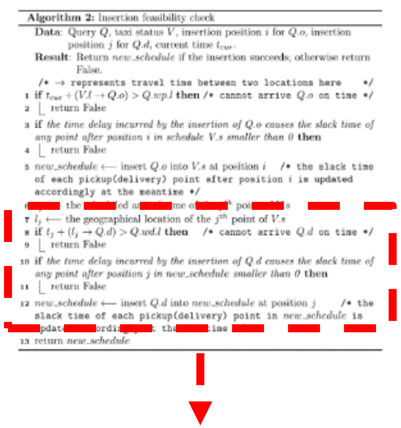


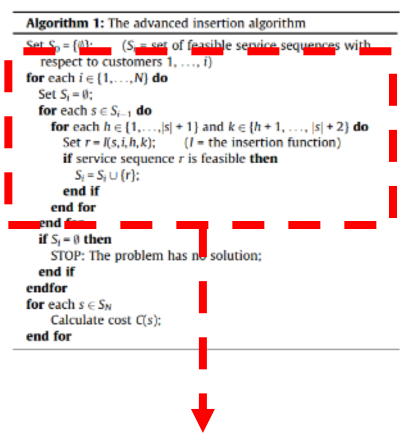
How to plan a route for the worker is central for DR

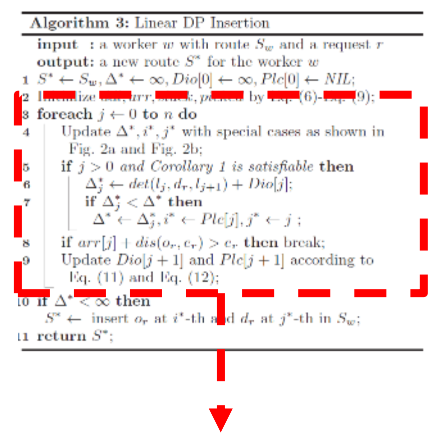
Dynamic Ridesharing Services

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

T-share	Adaptive insertion	Kinetic	Prune GreedyDP
[ICDE'13]	[EJOR'11]	[VLDB'14]	[VLDB'18]

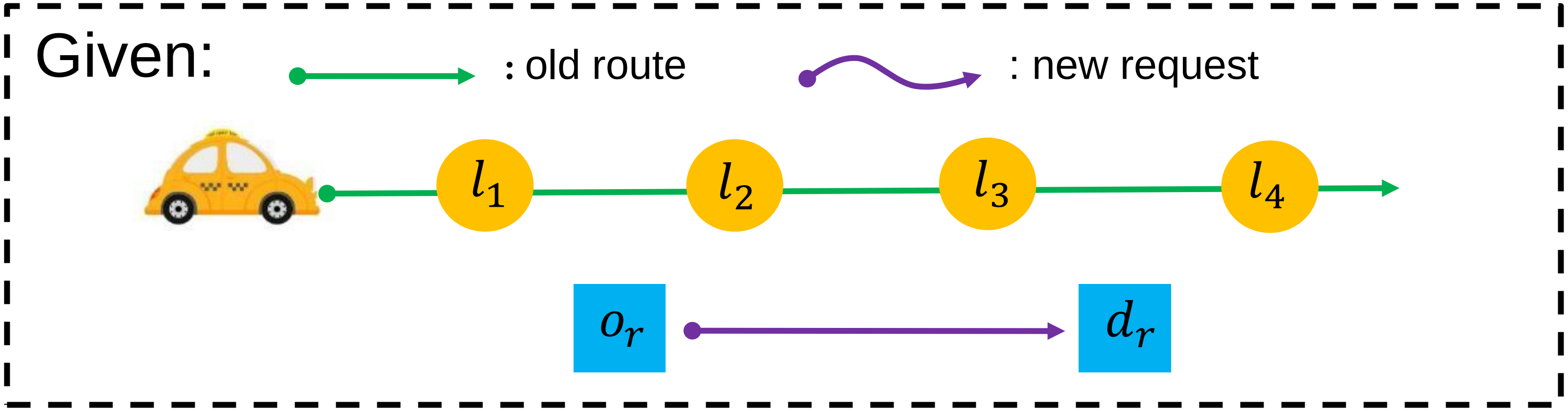




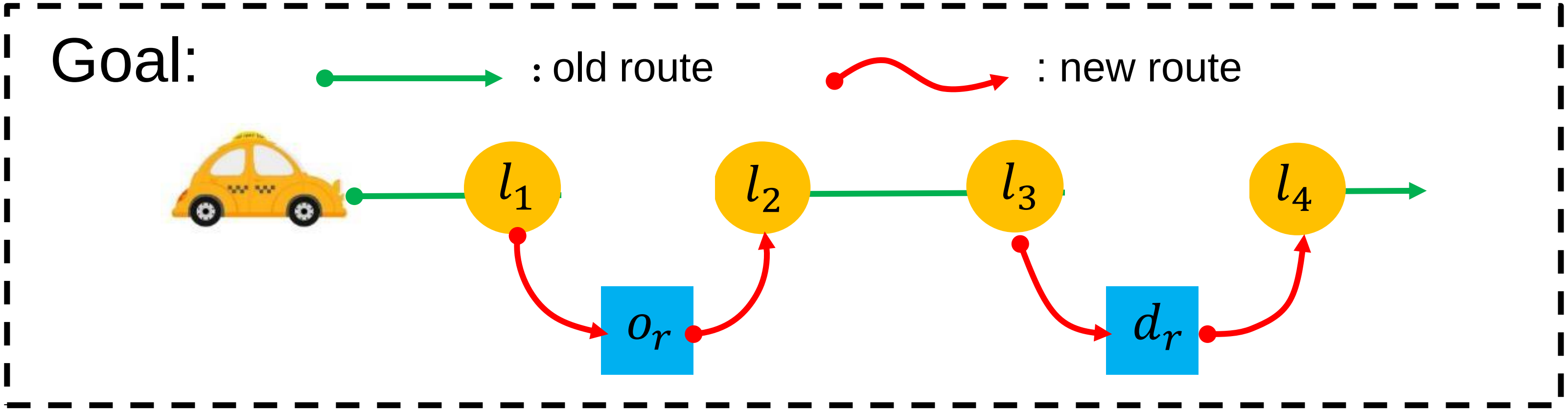


Insertion Operator: insert a new request into the current route

Insertion Operator



Insertion:
finding the appropriate location to
Insert the new request



Outline

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

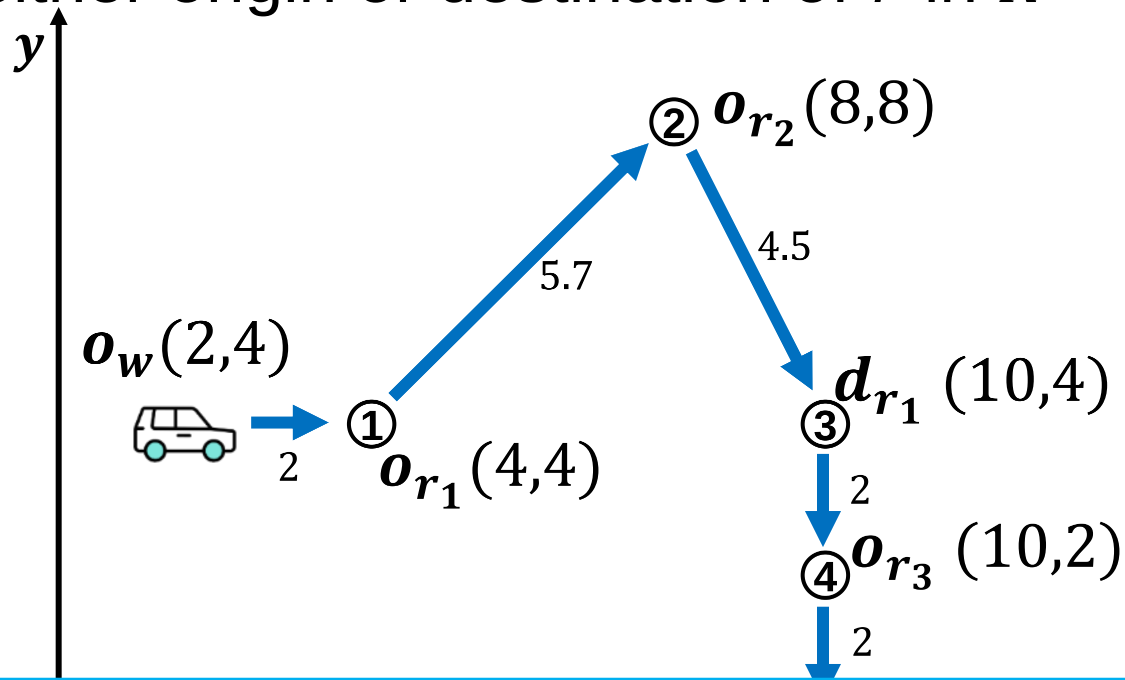
Worker

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



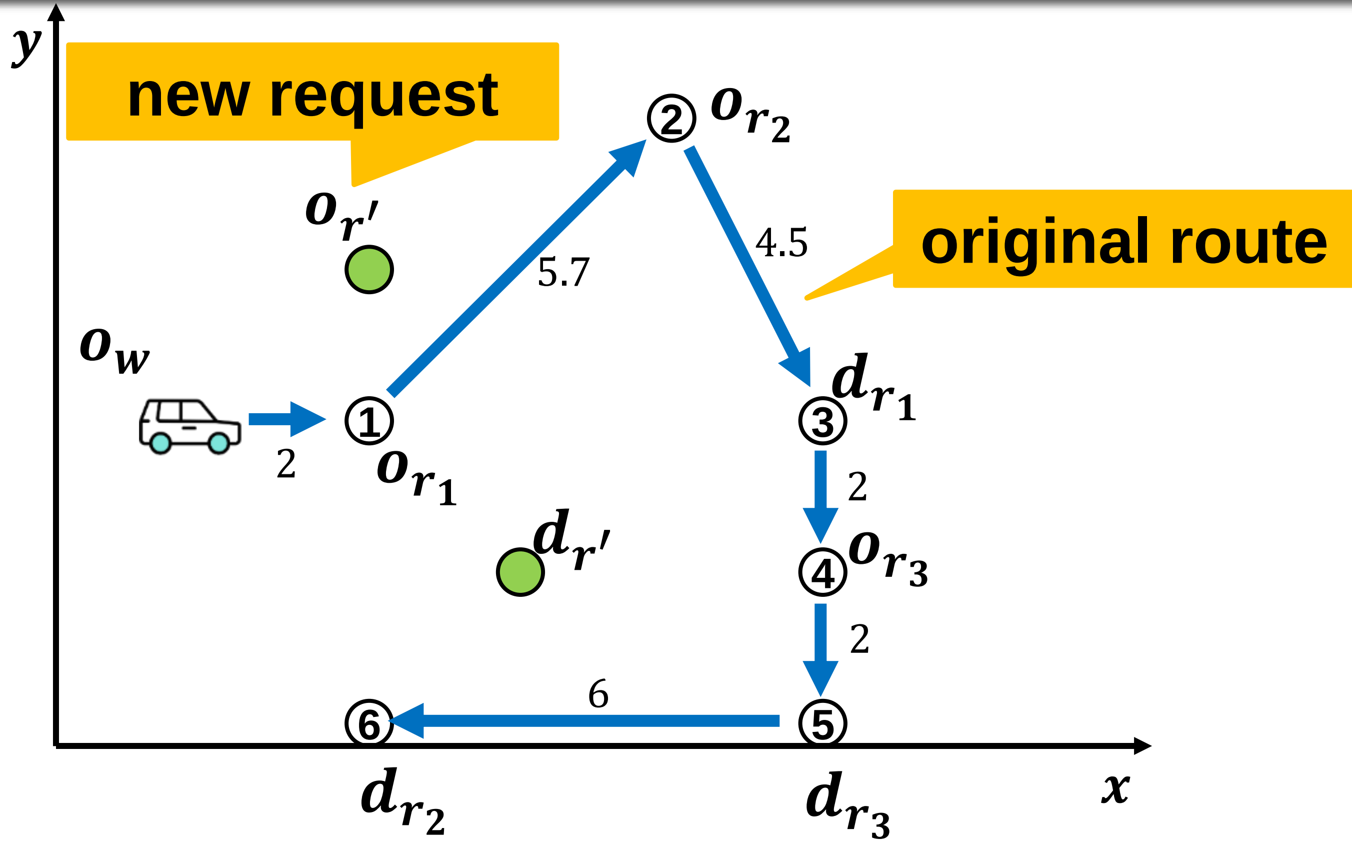
Feasible: (1) Capacity constraint; (2) Deadline constraint

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

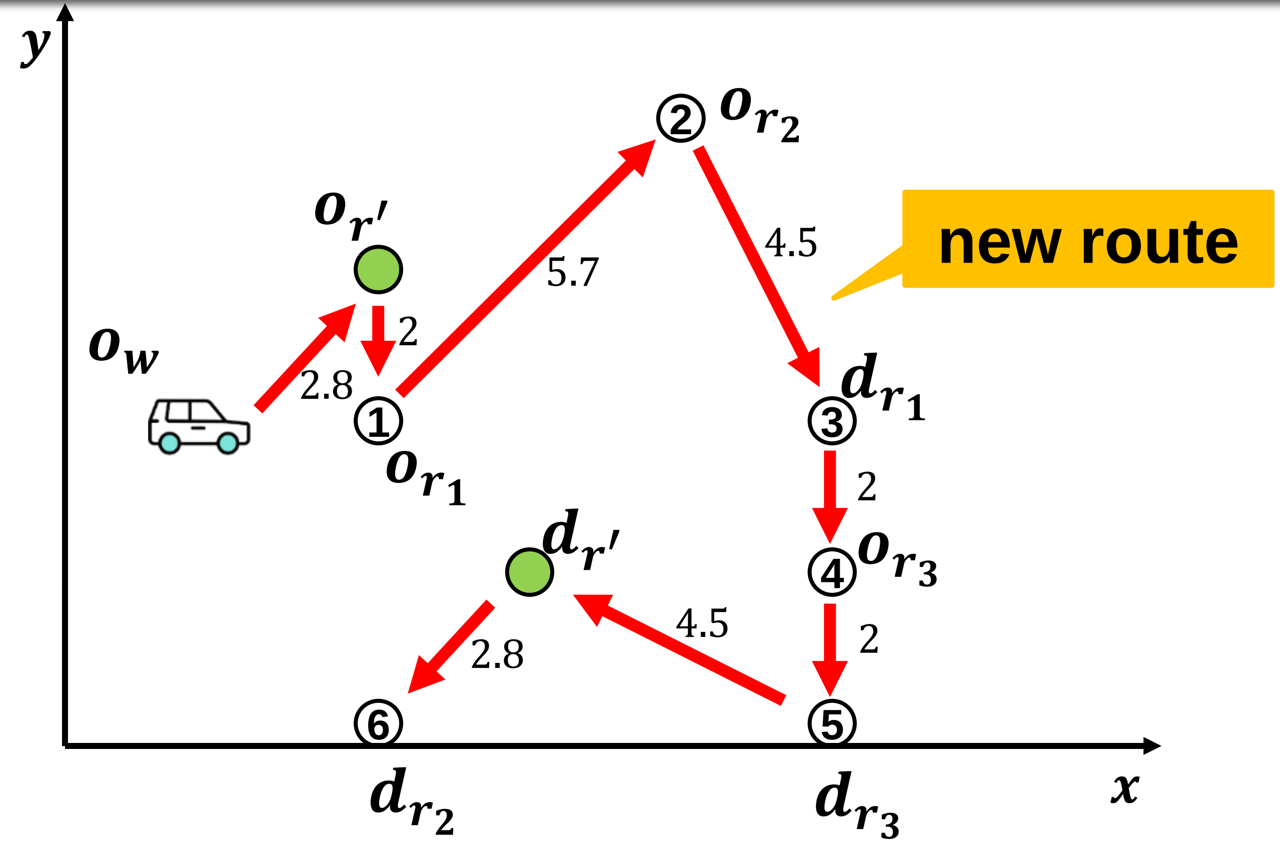
Focus of This talk.

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

Basic Algorithm

How to reduce the time complexity?

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 $O(1)$

×

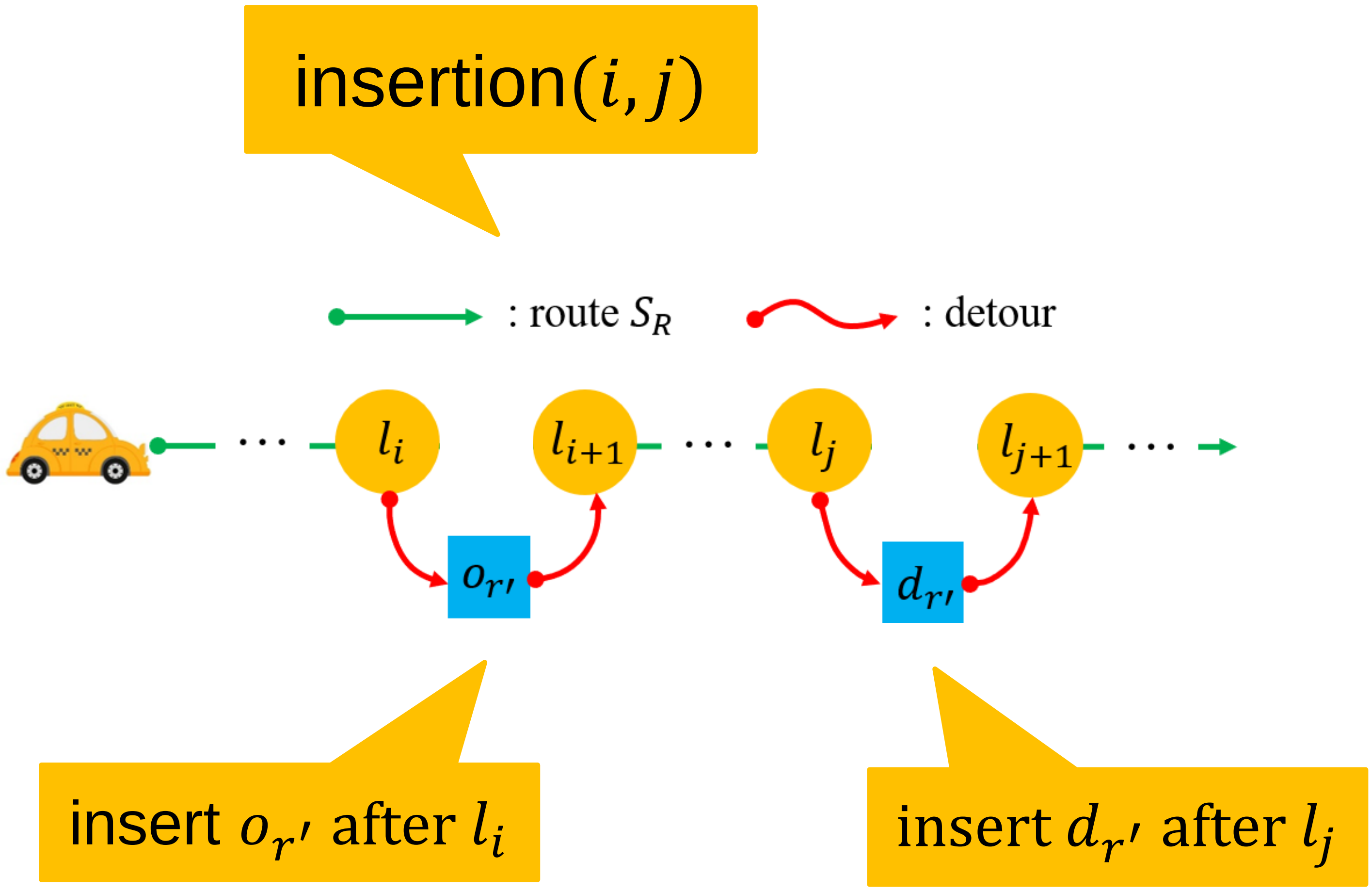
Enumerate all possible insertion pairs in $O(n^2)$

+

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

Partition-based Framework

Insertion(i, j)



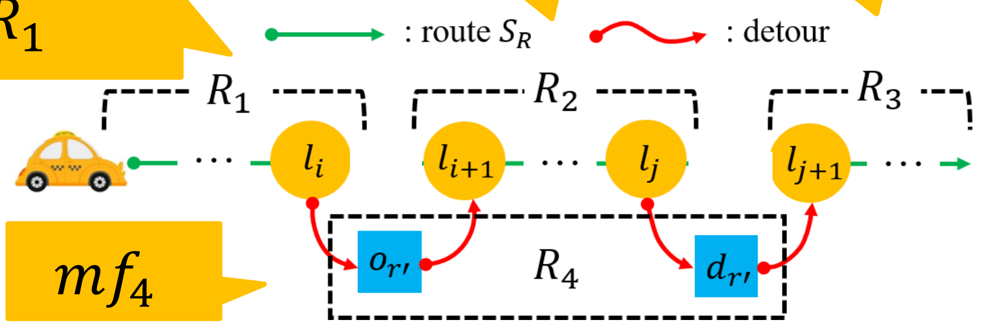
Partition-based Framework

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

mf_1: the maximum flow time of all requests in R_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 $O(n^2)$

+

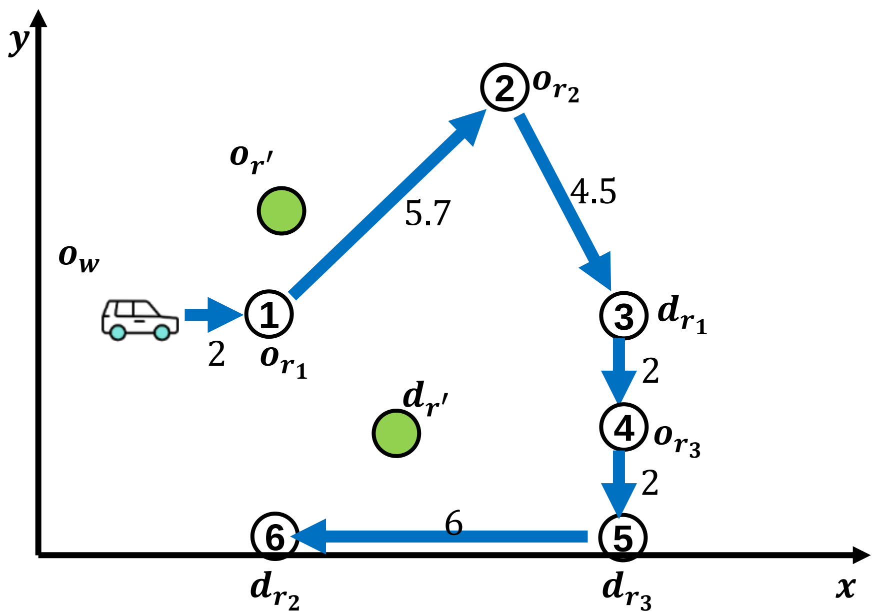
Pre-processing in $O(n^2)$

Pre-processing

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

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

$O(n^2)$



i \ 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

$mobj$

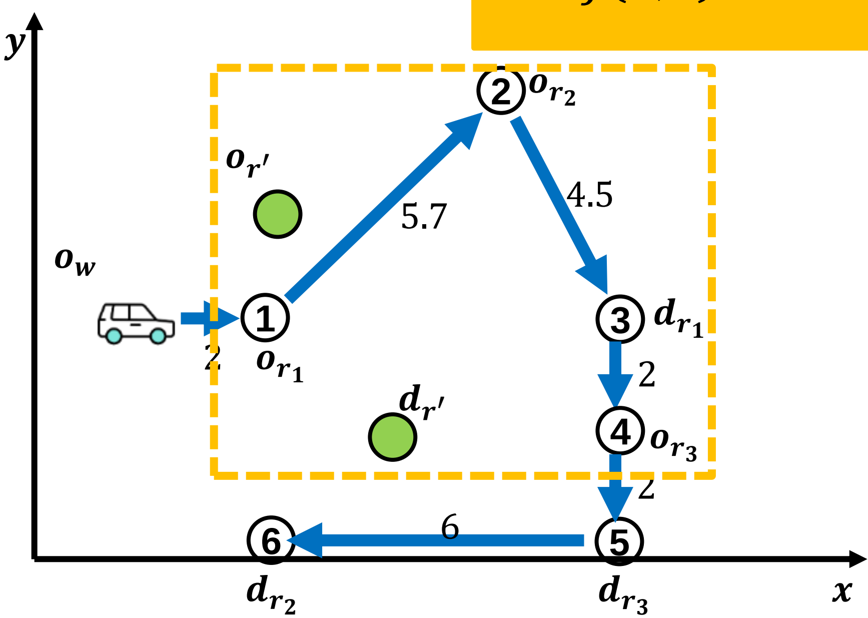
Pre-processing

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

$O(n^2)$

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

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



i \ 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 $O(1)$

×

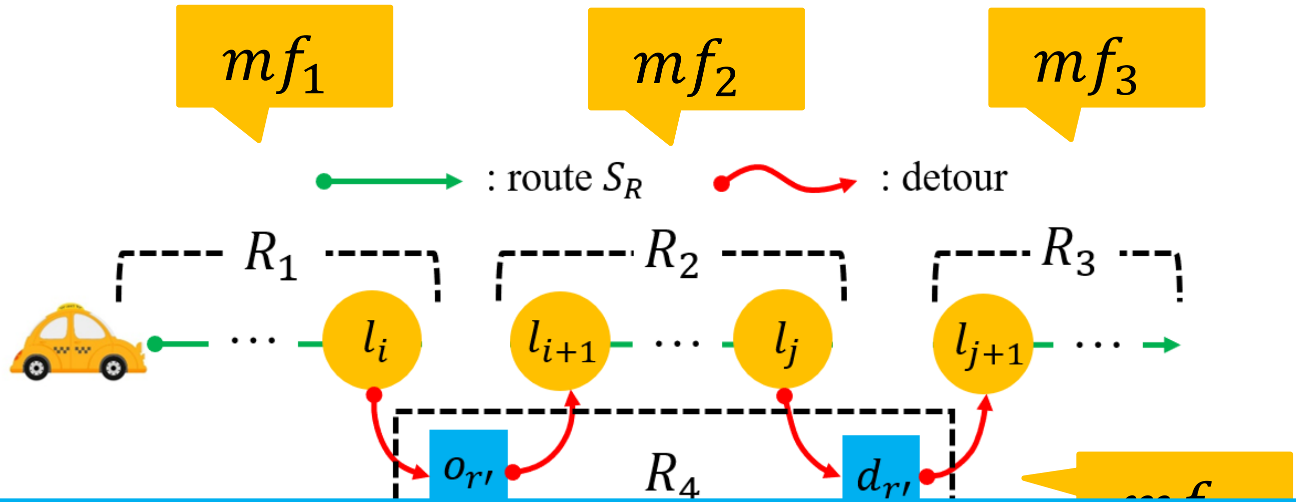
Enumerate possible insertion pairs in $O(n^2)$

+

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
  - $mf_2 = \boxed{det(i, o_{r'})} + \boxed{mobj(i+1, j)}$ matrix in pre-processing
  - $mf_3 = det(i, o_{r'}) + det(j, d_{r'}) + mobj(j, n)$
  - $mf_4 = arr(j) + det(i, o_{r'}) + dis\left(l_j, d_{r'}\right) + (\alpha - 1)t_{r'}$

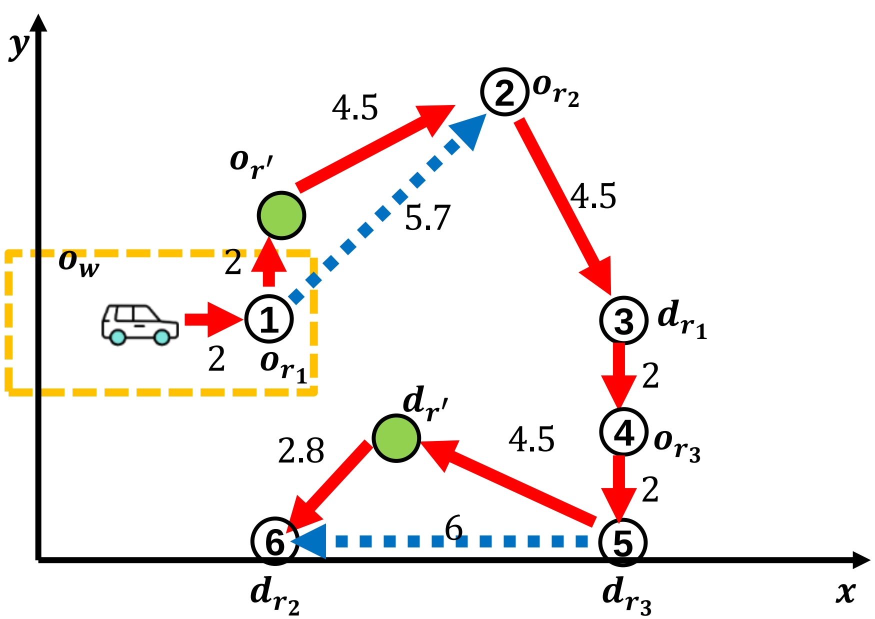


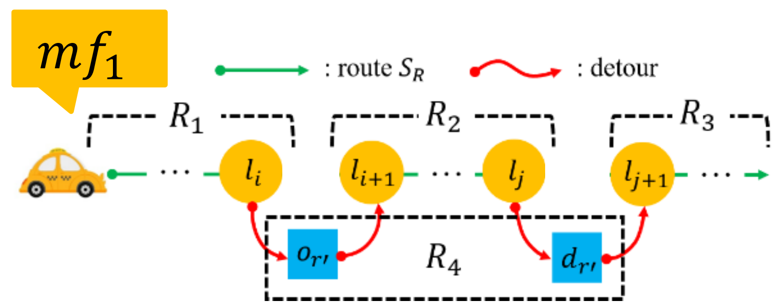
We can calculate the objective in $O(1)$ time with formulas

Objective Calculation

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

$R_1 = \emptyset$





i \ 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

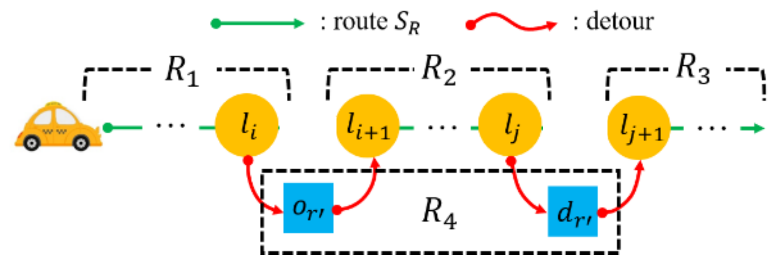
$mobj$

Objective Calculation

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

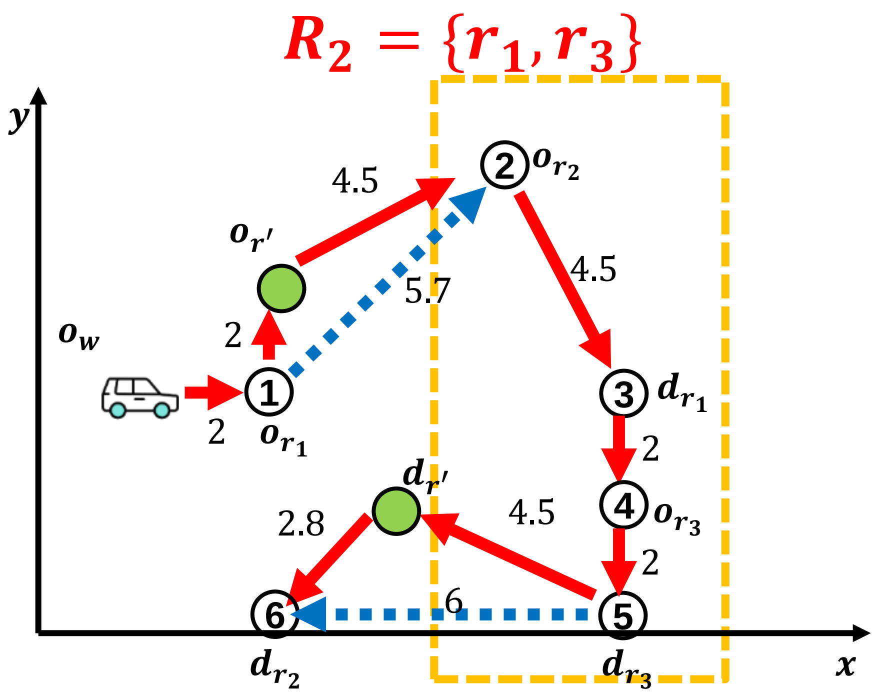
$det(k,p)$: the detour of inserting p after k-th location

mf_2



i \ 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

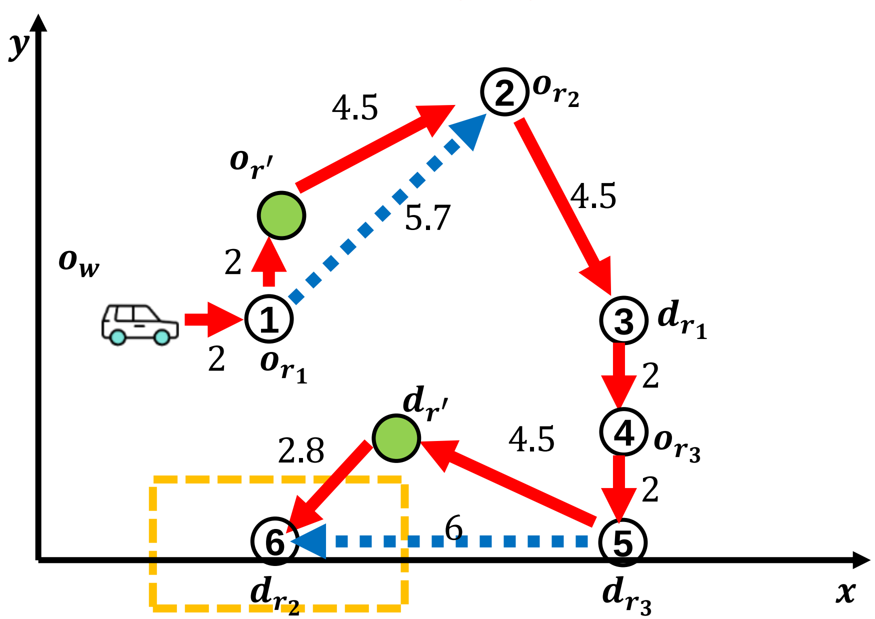
$mobj$



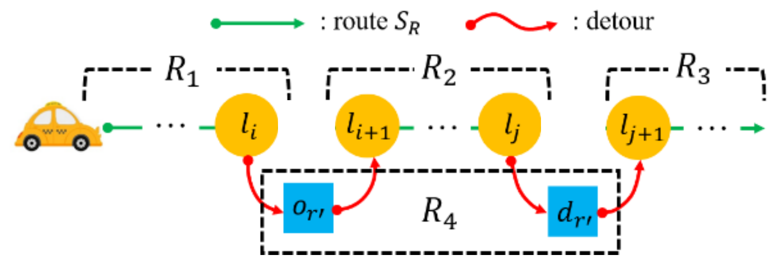
Objective Calculation

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

$R_3 = \{r_2\}$



mf_3



i \ 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

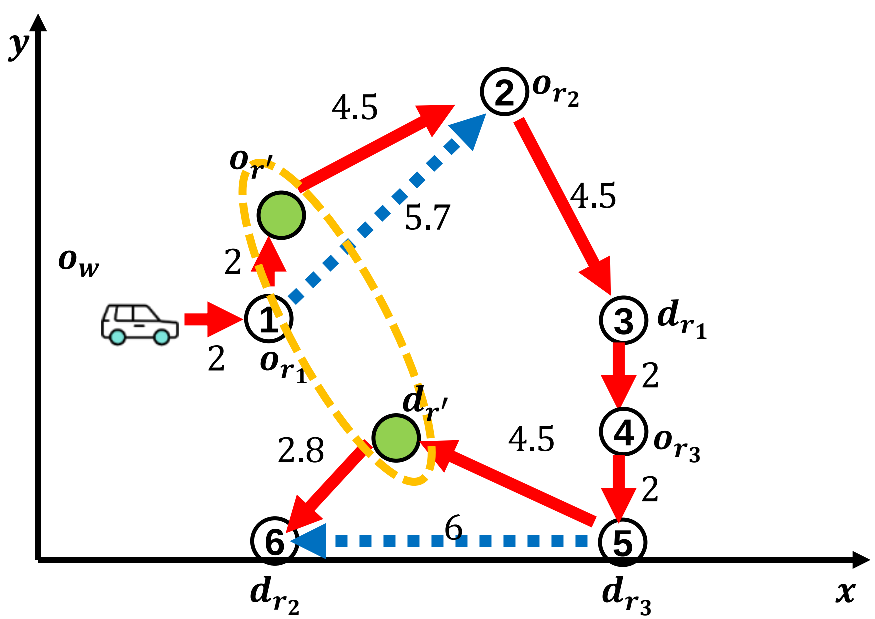
Objective Calculation

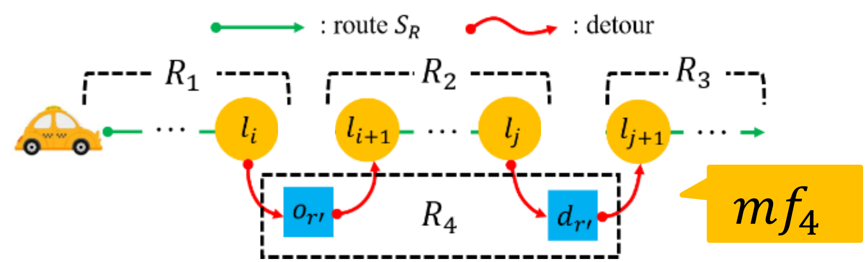
- Insertion $(1,5)$
  - mf_4 is the flow time of r': $21.5 - 0 = 21.5$

arrival time of $d_{r'}$

release time of r'

$R_4 = \{r'\}$



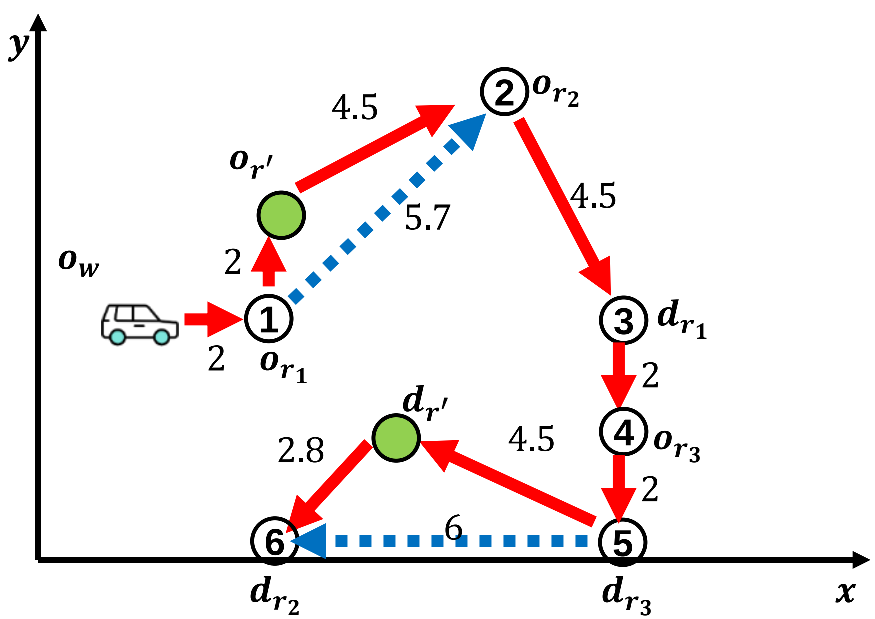


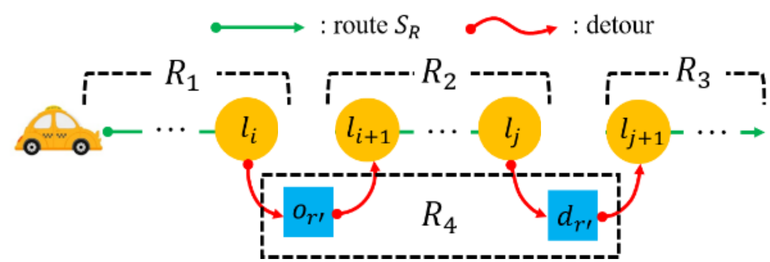
i \ j	0	1	2	3	4	5	6
0	0	0	0	12.2	12.2	16.2	22.2
1	-	0	0	12.2	12.2	16.2	22.2
2	-	-	0	12.2	12.2	16.2	22.2
3	-	-	-	12.2	12.2	16.2	22.2
4	-	-	-	-	0	16.2	22.2
5	-	-	-	-	-	16.2	22.2
6	-	-	-	-	-	-	22.2

$mobj$

Objective Calculation

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





i \ 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:

$$pck(i) \le c_w - c_{r'}$$
$$pck(j) \le c_w - c_{r'}$$

Deadline constraint:

$$det(i, o_{r'}) \le slk(i)$$
$$det(i, o_{r'}) + det(j, d_{r'}) \le slk(j)$$
$$arr(j) + det(i, o_{r'}) + dis\left(l_j, d_{r'}\right) \le e_{r'}$$

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 $O(1)$

$\times$

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 $O(1)$

$\times$

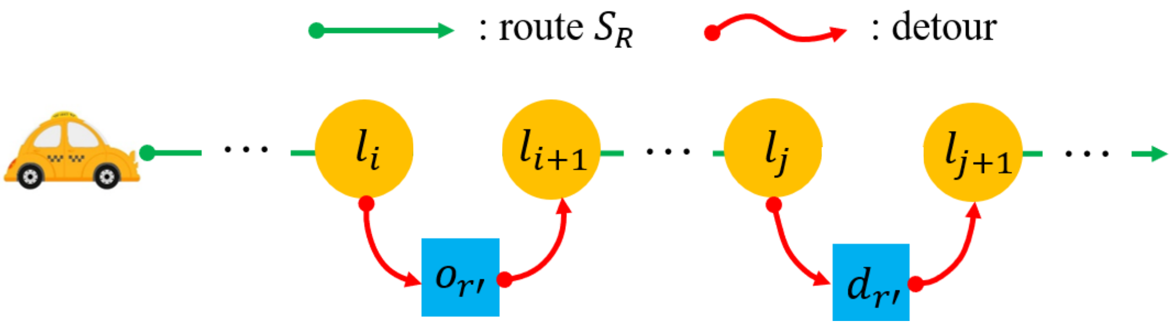
Enumerate locations to insert origin in $O(n)$

$+$

Pre-processing in $O(n)$

Segment-based DP Algorithm

Segment-based DP Algorithm

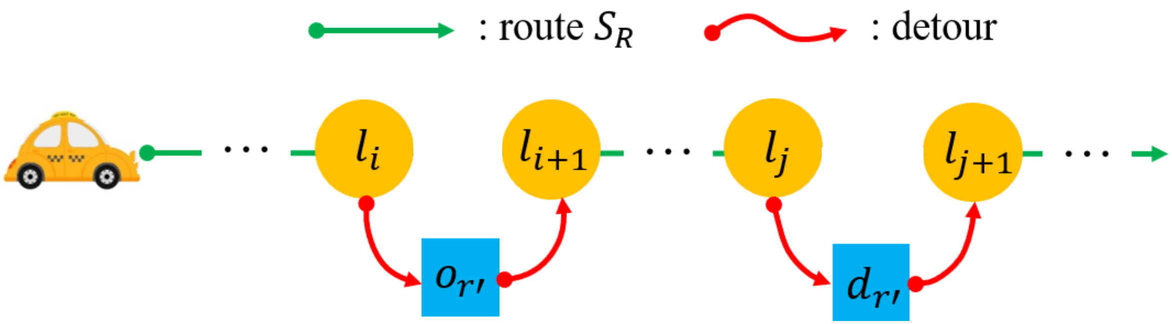


Enumerate i and j

$O(n^2)$

Partition-based Framework

Segment-based DP Algorithm



Enumerate i and j

Enumerate i,
find optimal j^*

$O(n^2)$

Reducing

$O(n)$

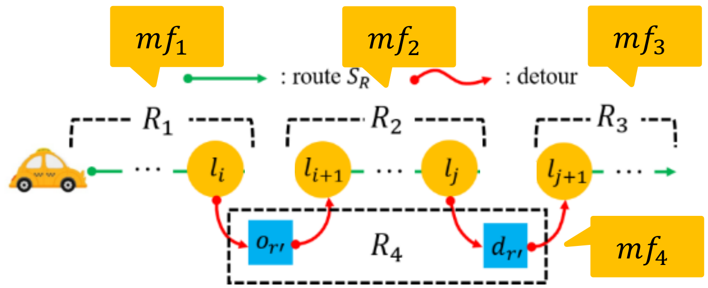
Partition-based Framework

Segment-based DP Algorithm

Segment-based DP Algorithm

- Objective Calculation

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



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

Partition-based Framework

partition i and j

$$A(i) = \det(i, o_{r'})$$
$$B(j) = \max\begin{cases} \det(j, d_{r'}) + mobj(j, n), \\ arr(j) + dis(l_j, d_{r'}) + (\alpha - 1)t_{r'} \}\end{cases}$$

relevant to i

relevant to j

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

Segment-based DP Algorithm

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

✕ 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 $O(?)$

$$B(j^*) = \min_{\substack{i<j<brk(i) \\ det(i,o_{r'}) \le thr(j)}} B(j)$$

Segment tree
$O(\log n)$

$O(n \log n)$

Fenwick tree (dynamic)
$O(1)$

$O(n)$

Example

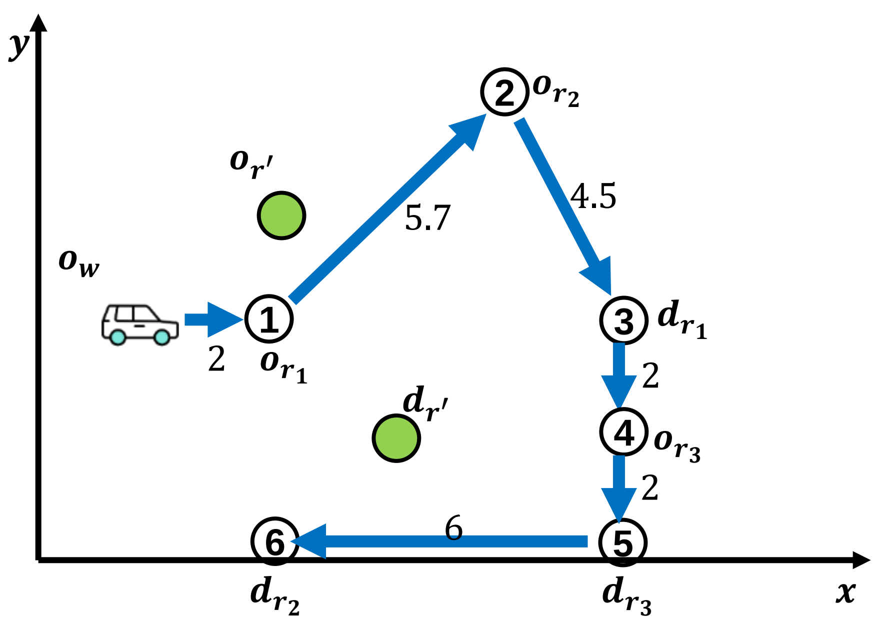
- Initialization
  - Initialize $thr(\cdot)$, $B(\cdot)$

objective relevant to j

threshold of j

$index$				$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	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

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



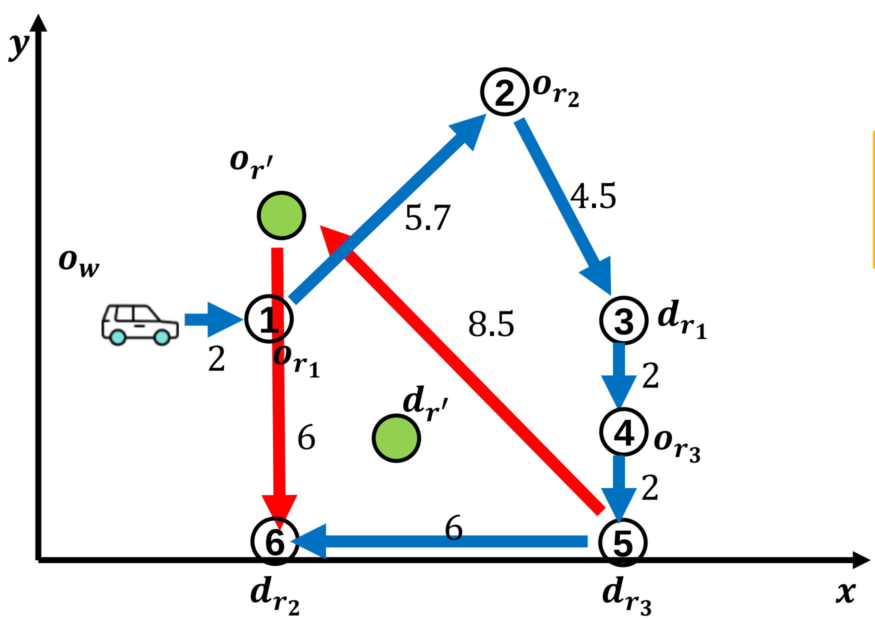
Example

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

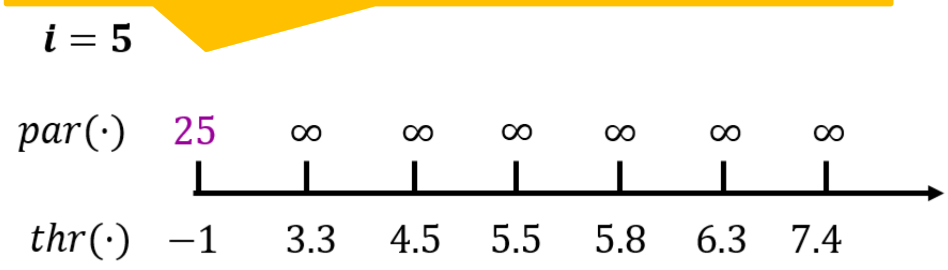
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



Update value of this point as 25



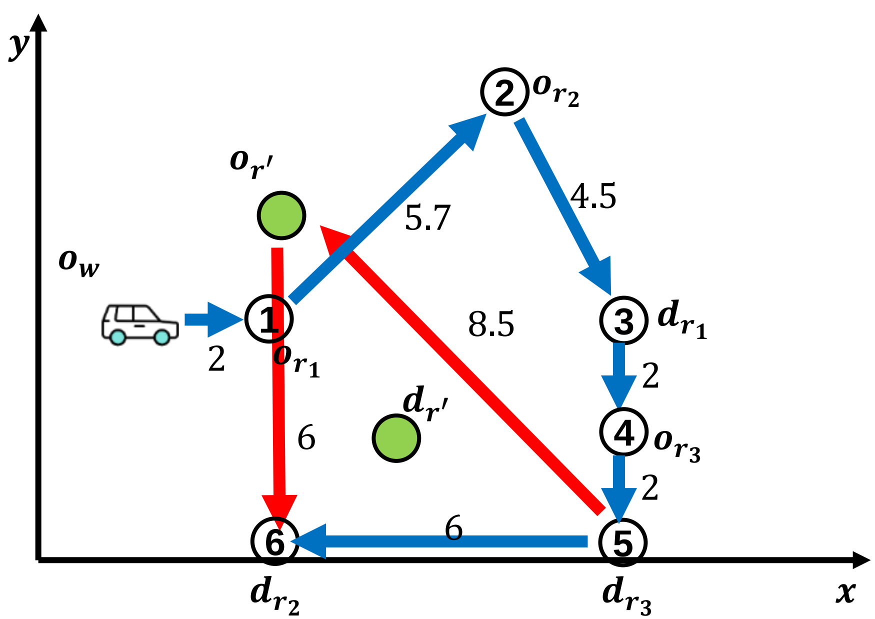
Example

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

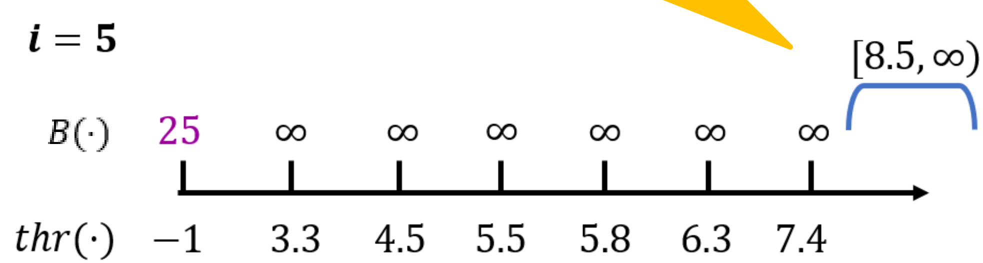
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



$det(5, o_{r'})$=8.5
The answer is ∞



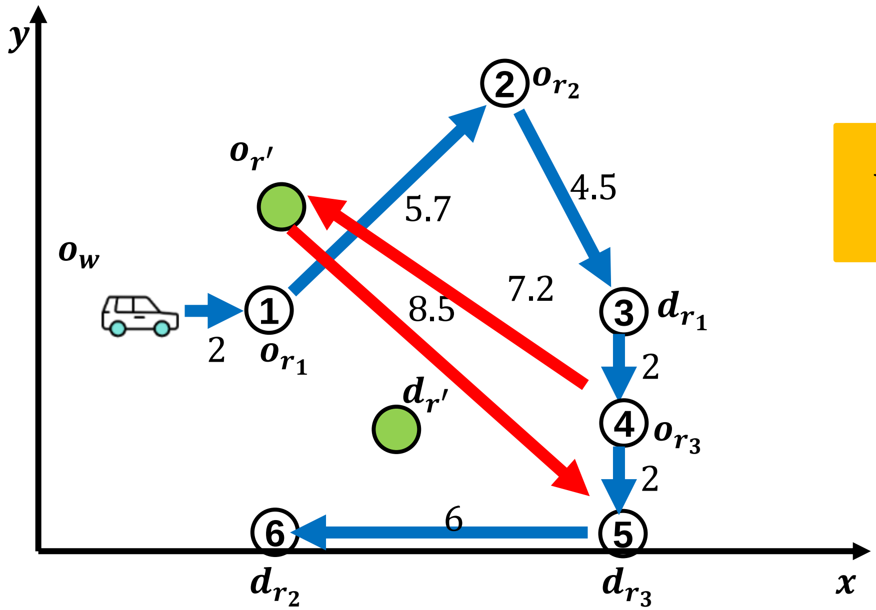
Example

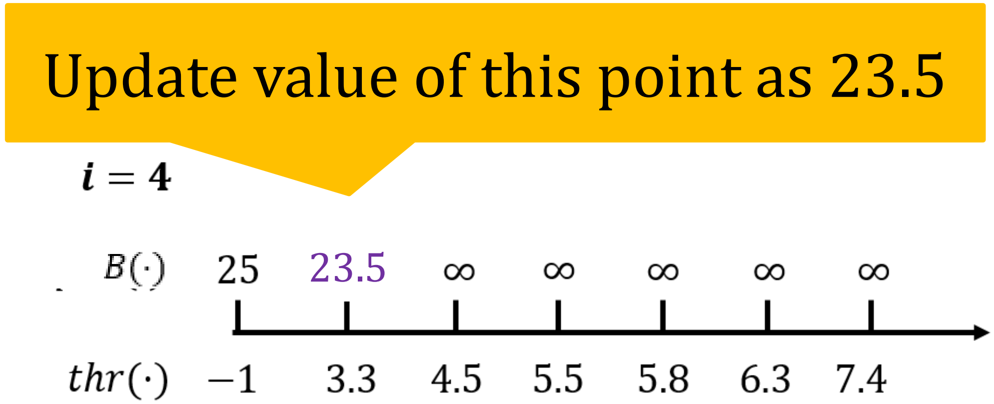
- Enumerate $i = 4$ ($A(i) = 13.7$)
  - Update $B(5) = 23.5$ at point $thr(5) = 3.3$

$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	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

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

Optimal solution: ∞





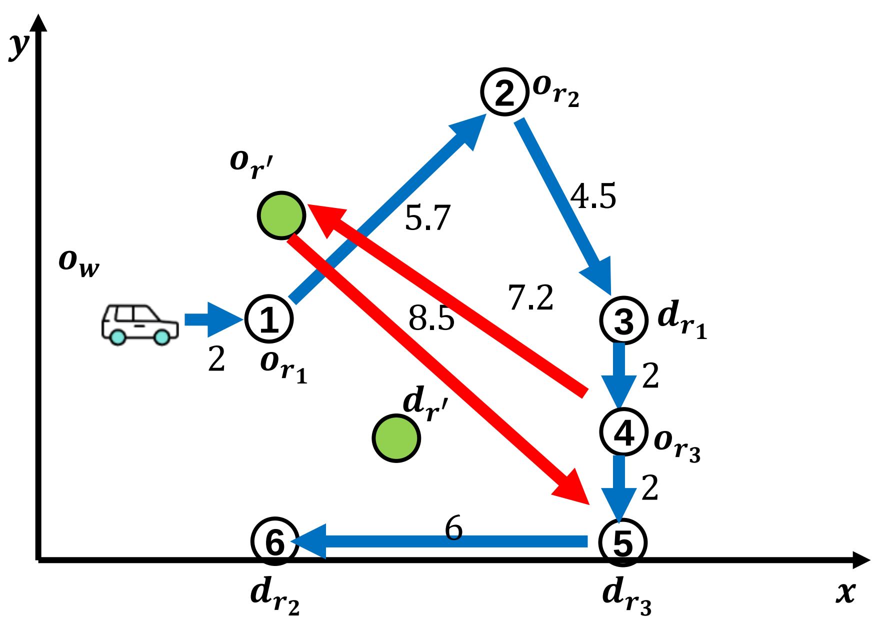
Example

- 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

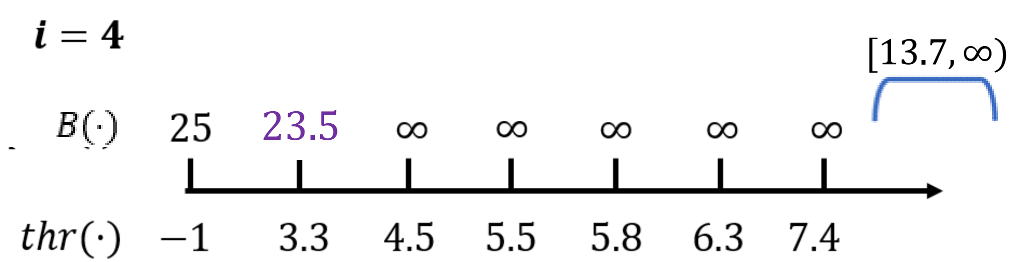
$$\min_{\substack{i<j<brk(i) \\ det(i,o_{r'})\le thr(j)}} B(j)$$

Optimal solution: ∞



$det(4, o_{r'}) = 13.7$

The answer is ∞



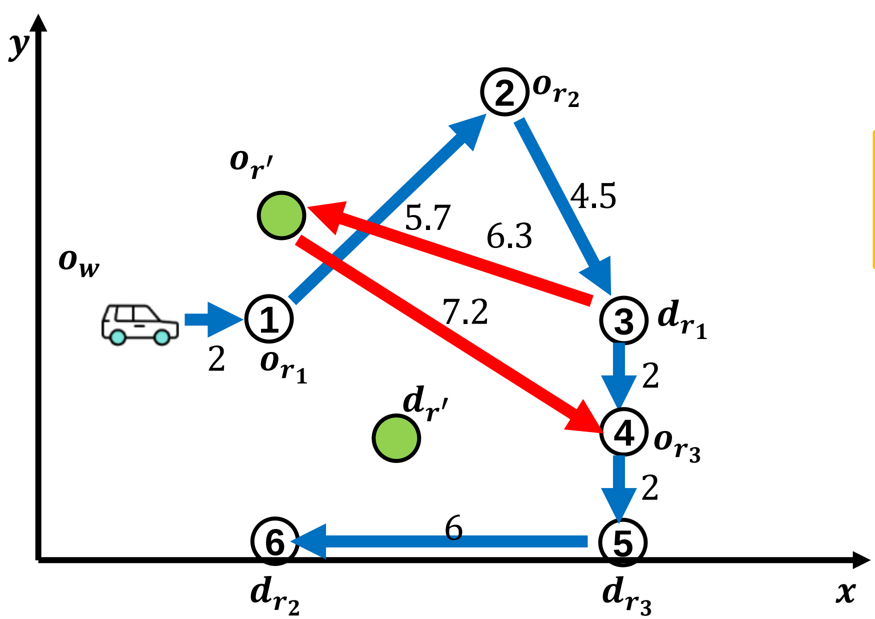
Example

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

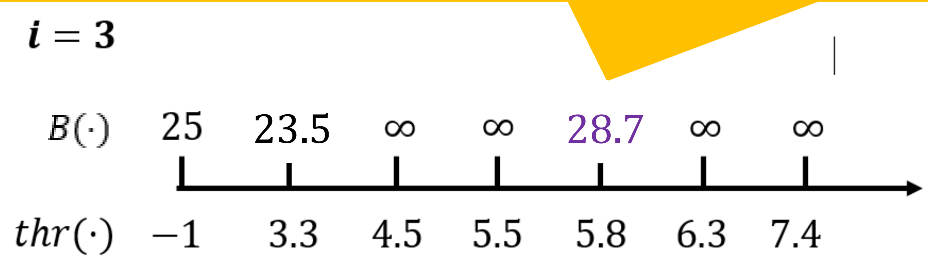
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



Update value of this point as 28.7



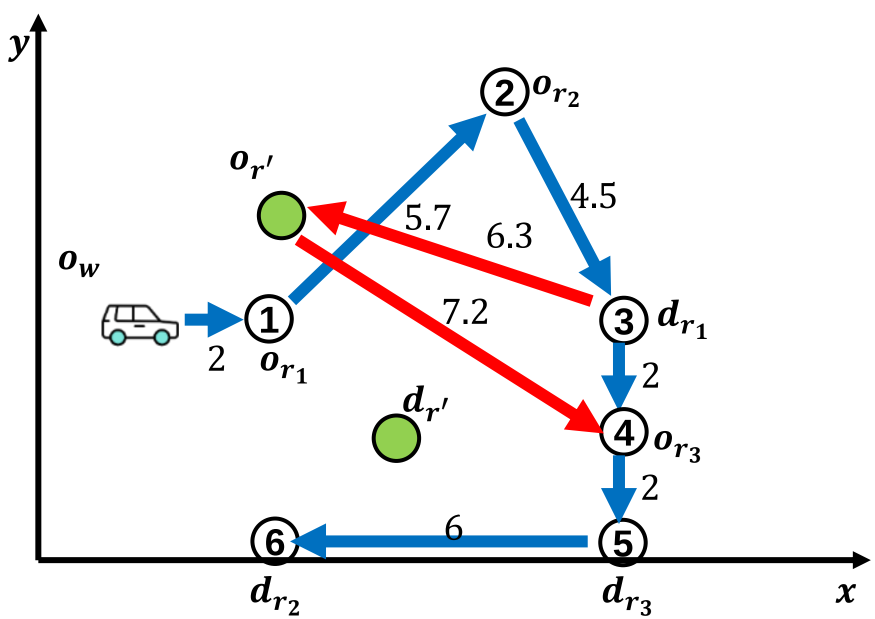
Example

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

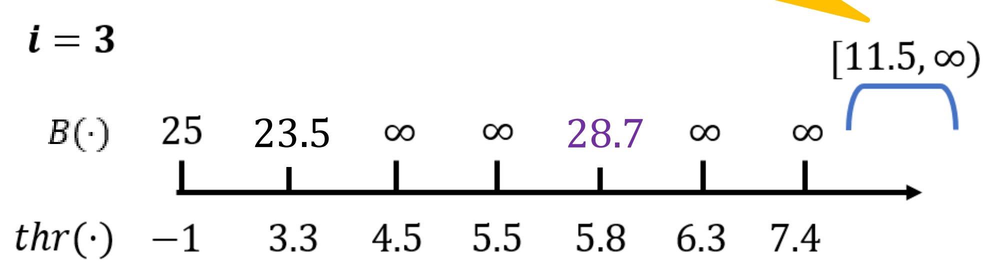
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



$det(3, o_{r'}) = 11.5$
The answer is ∞



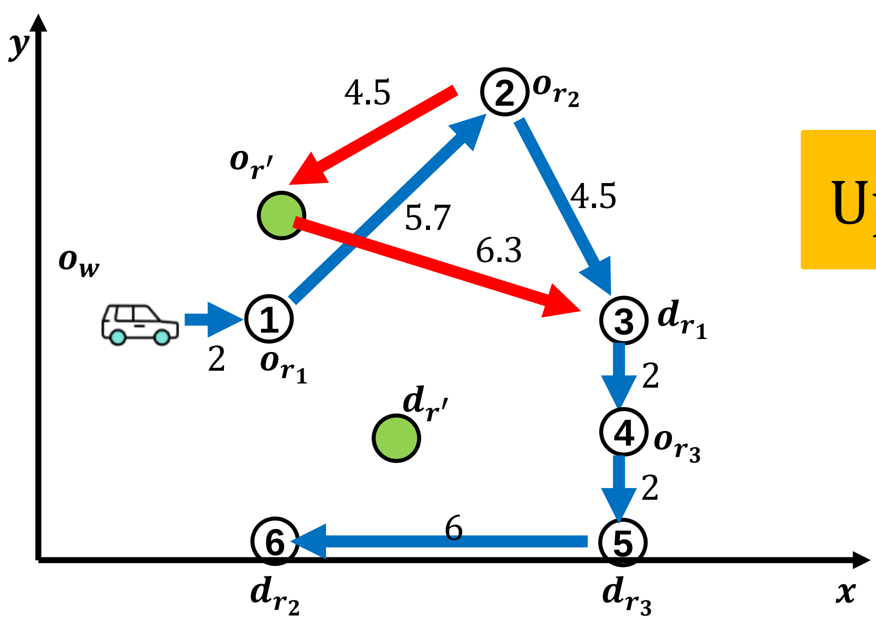
Example

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

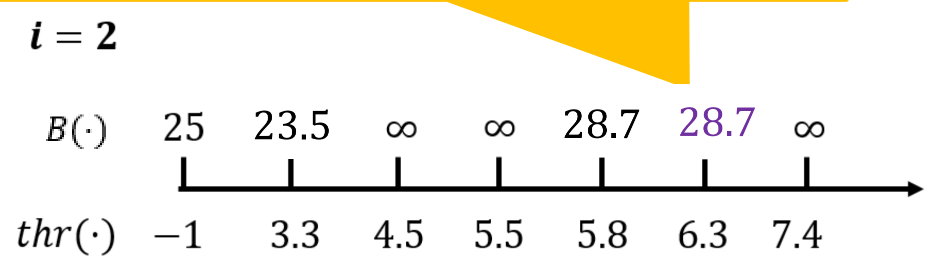
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



Update value of this point as 28.7



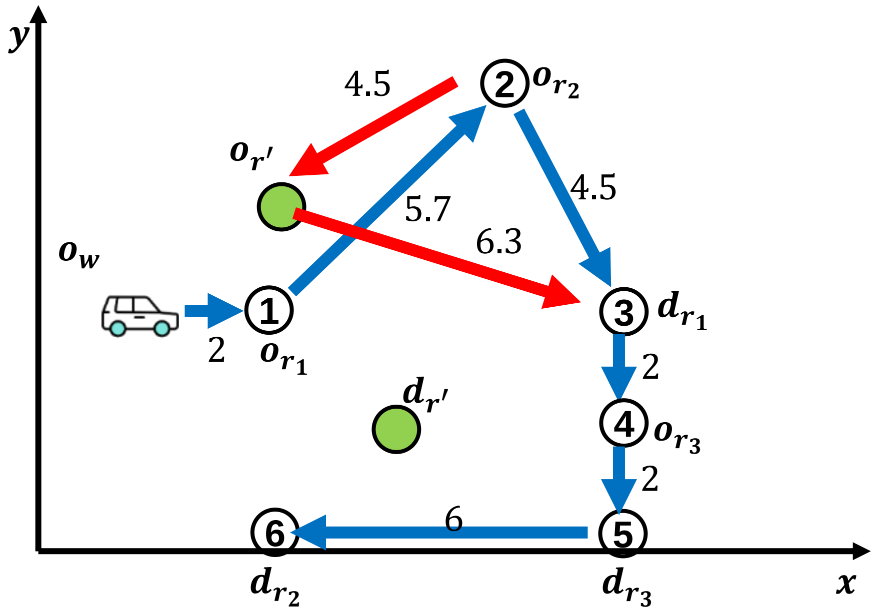
Example

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

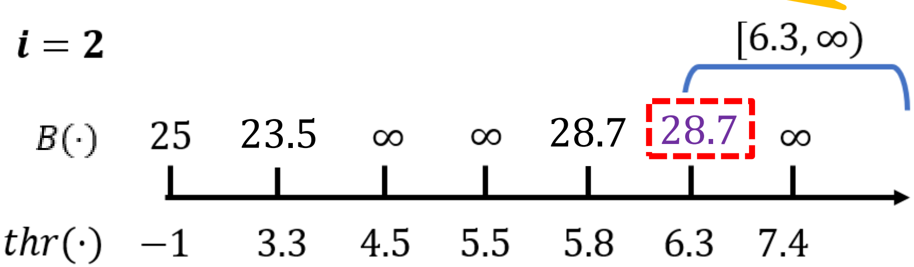
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: ∞



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



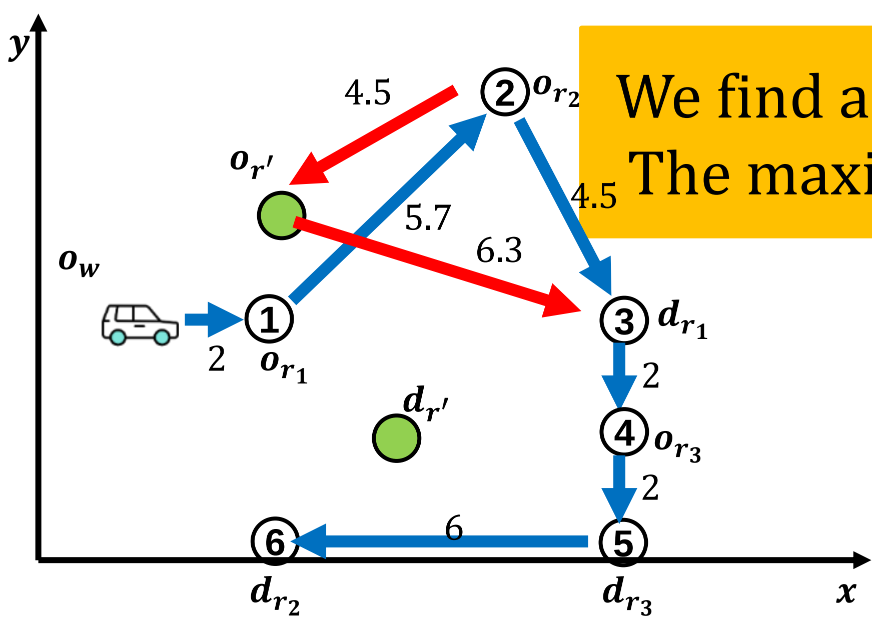
Example

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

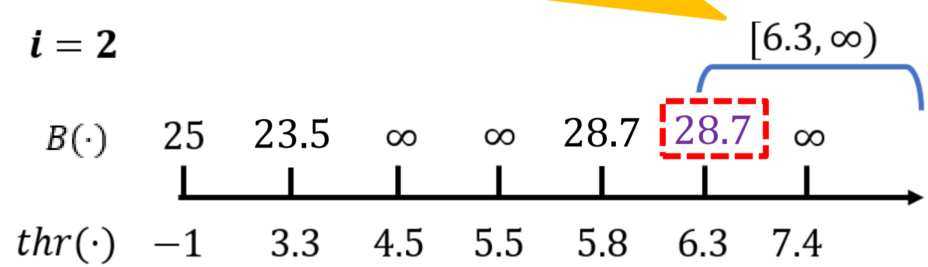
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: 35



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



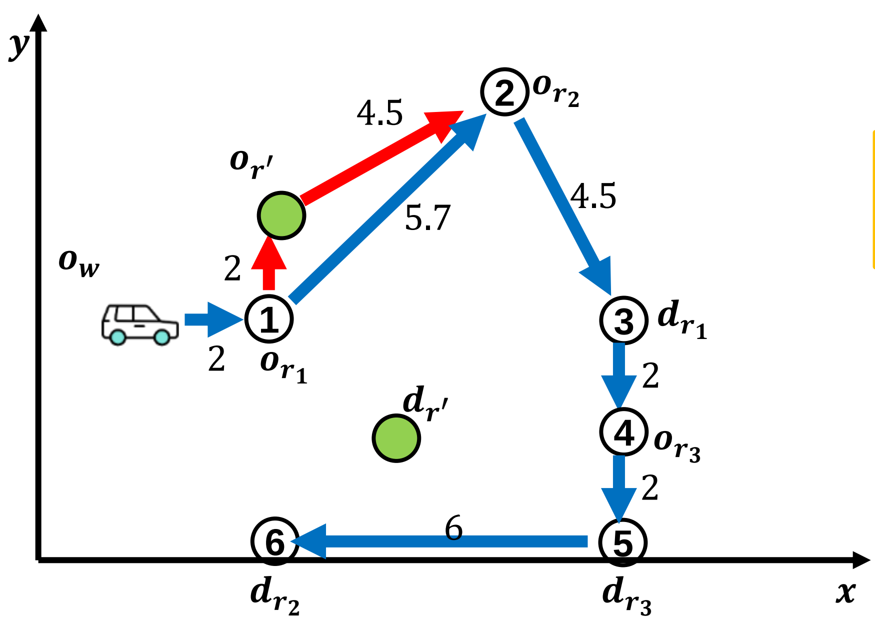
Example

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

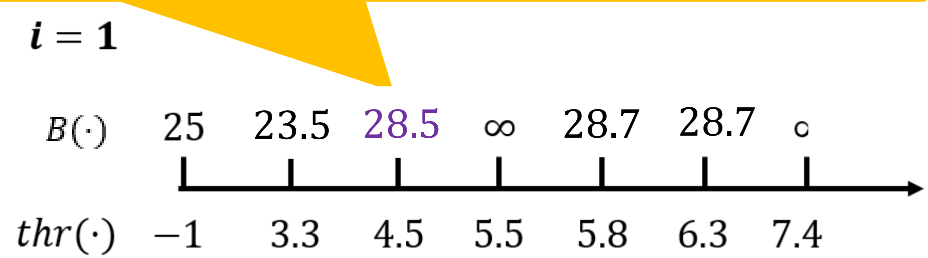
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: 35



Update value of this point as 28.5



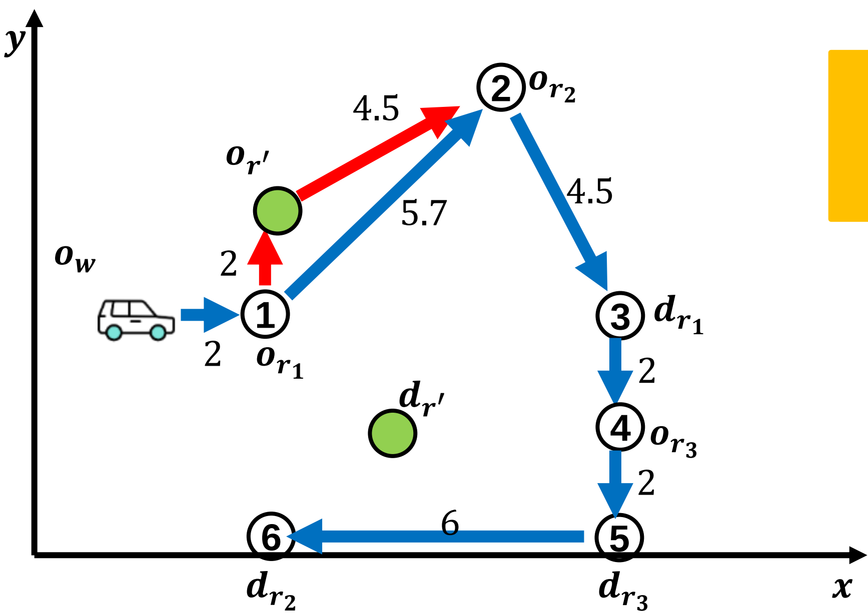
Example

- 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

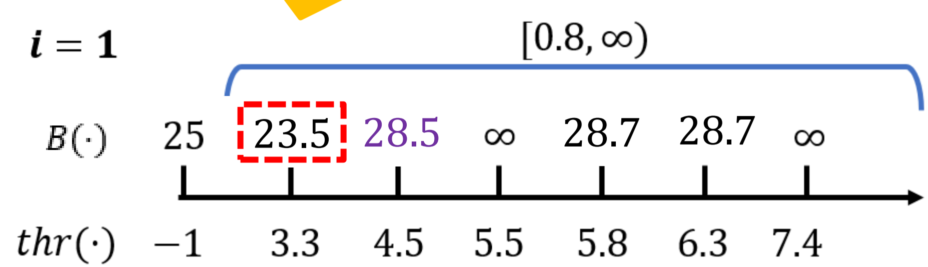
$$\min_{\substack{i<j<brk(i)\\ det(i,o_{r'})\le thr(j)}} B(j)$$

Optimal solution: 35



$det(1, o_{r'}) = 0.8$

The answer is 25.5



Example

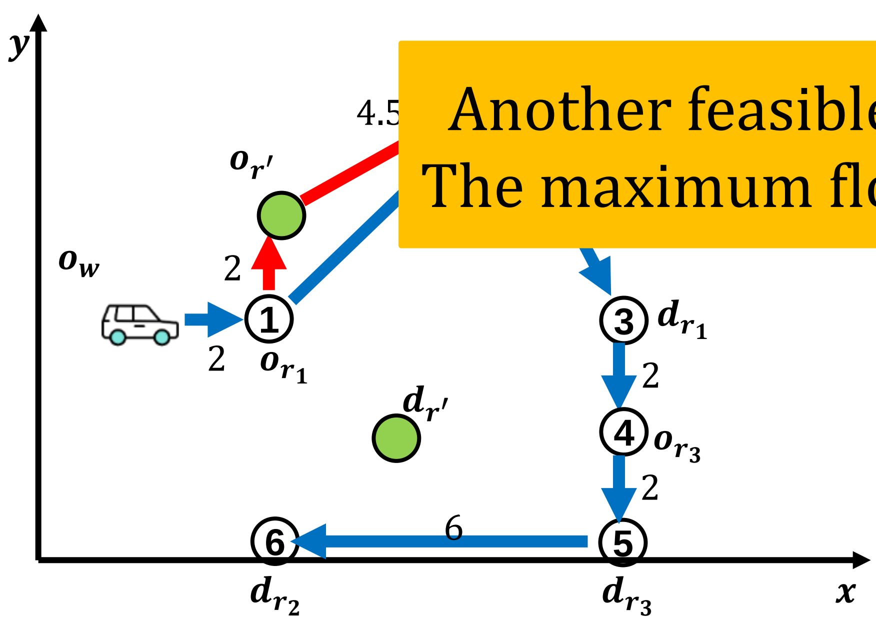
- Enumerate $i = 1$ $(A(i) = 0.8)$
  - Query the segment $[det(1, o_{r'}), \infty)$

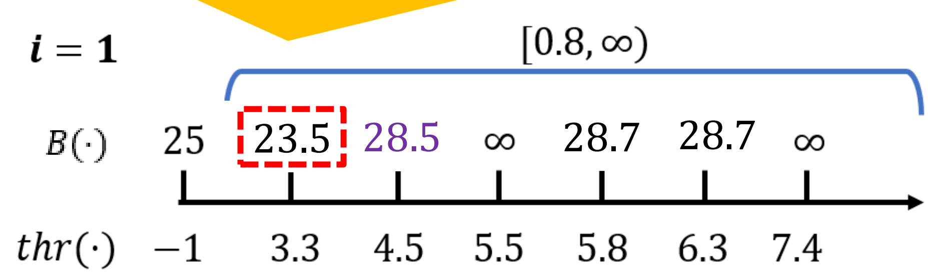
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	-1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: 24.3

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





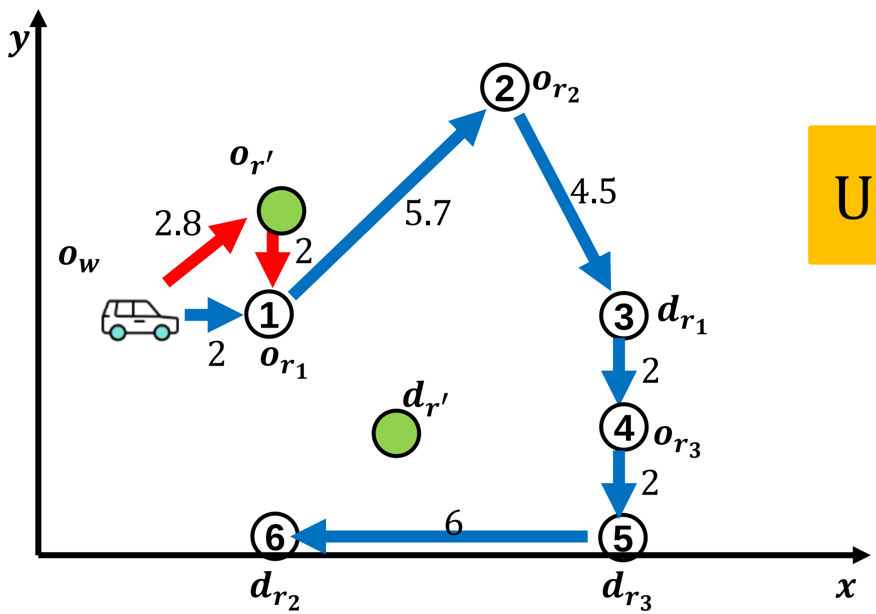
Example

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

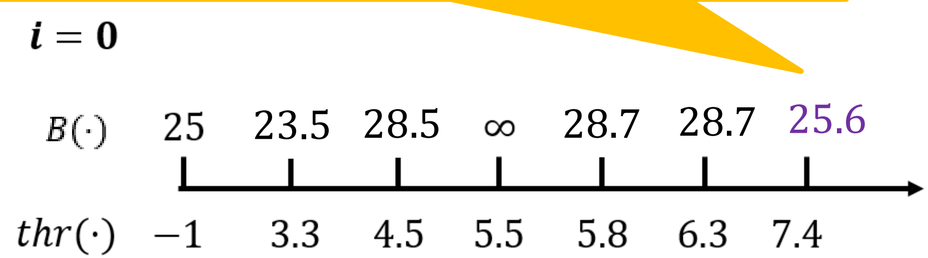
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: 24.3



Update value of this point as 25.6



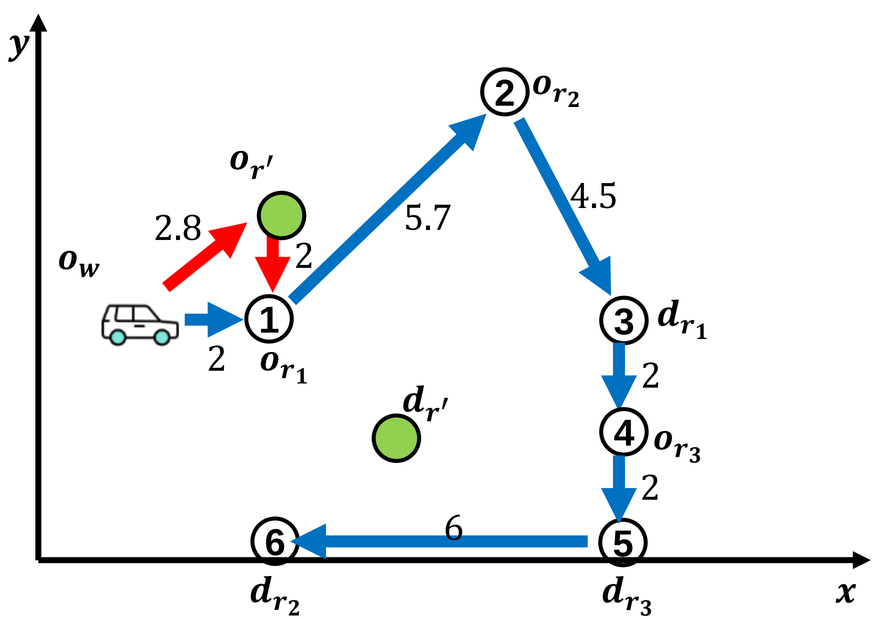
Example

- 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

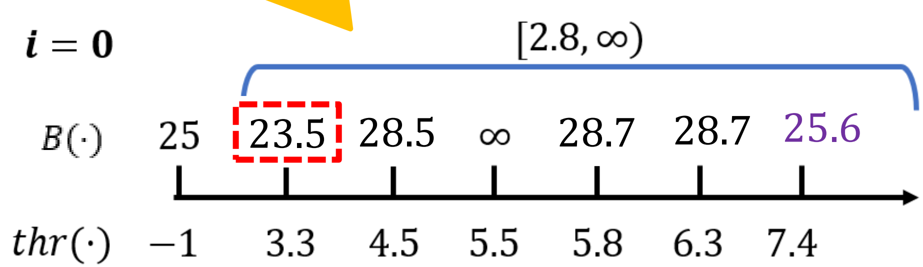
$$\min_{\substack{i<j<brk(i) \\ det(i,o_{r'})\le thr(j)}} B(j)$$

Optimal solution: 24.3



$det(0, o_{r'}) = 2.8$

The answer is 23.5



Example

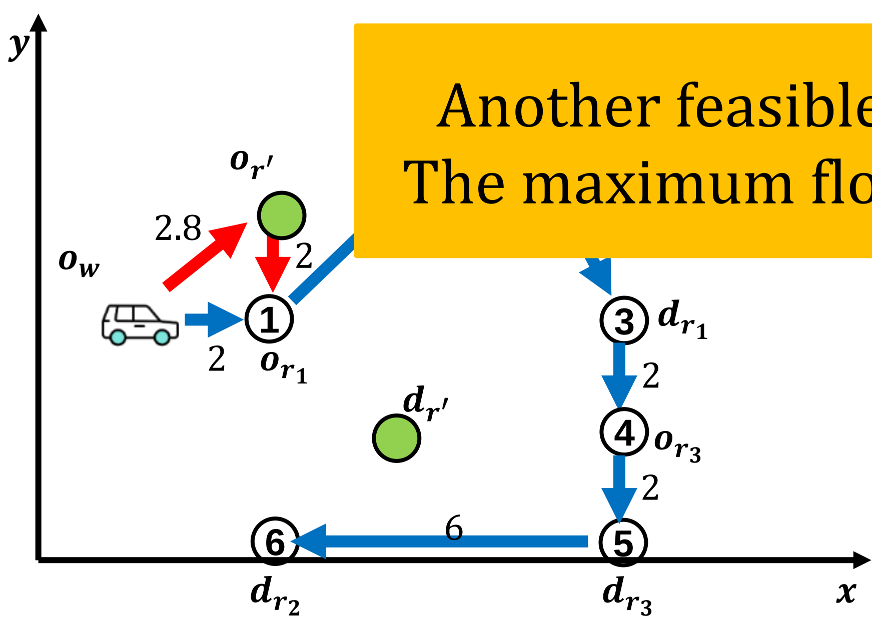
- Enumerate $i = 0$ $(A(i) = 2.8)$
  - Query the segment $[det(0, o_{r'}), \infty)$

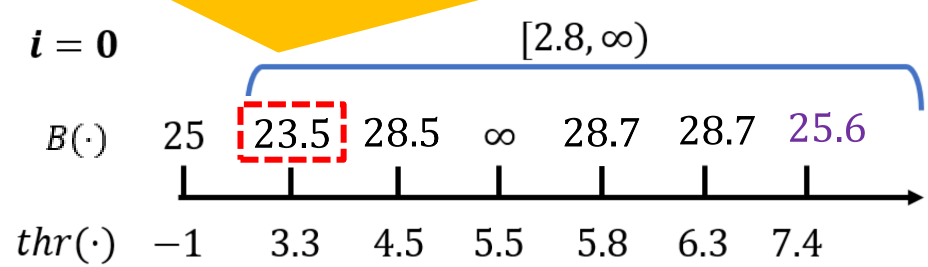
$index$	$0(o_w)$	$1(o_{r_1})$	$2(o_{r_2})$	$3(d_{r_1})$	$4(o_{r_3})$	$5(d_{r_3})$	$6(d_{r_2})$
$thr(\cdot)$	5.5	7.4	4.5	6.3	5.8	3.3	−1
$B(\cdot)$	27.5	25.6	28.5	28.7	28.7	23.5	25

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

Optimal solution: 24.3

Another feasible solution: insertion(0,5)
The maximum flow time is 2.8+23.5=26.3



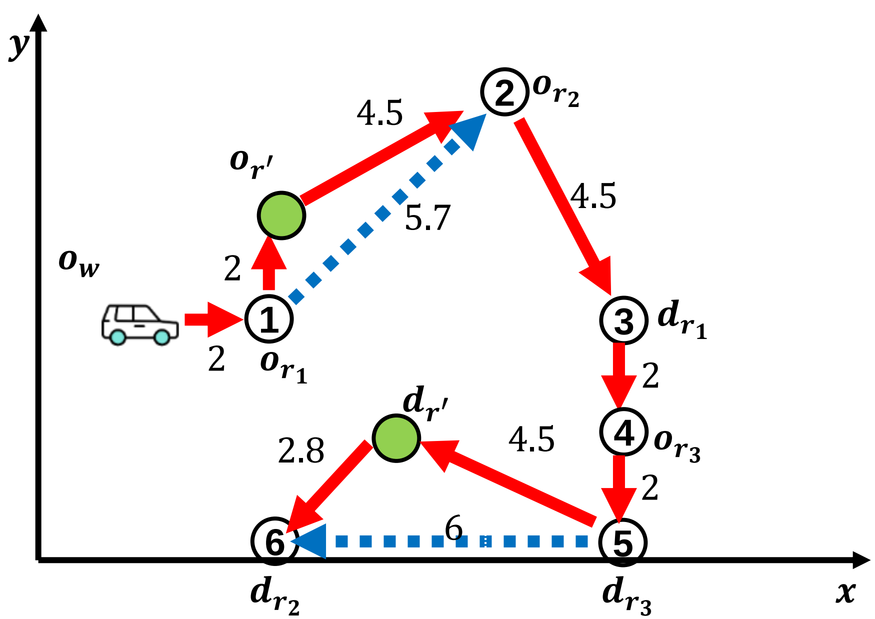


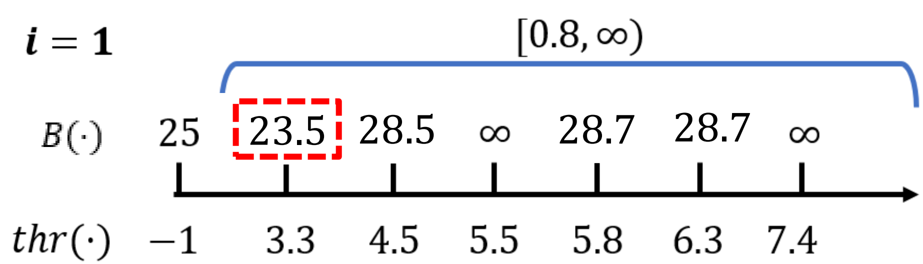
Example

- 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

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





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

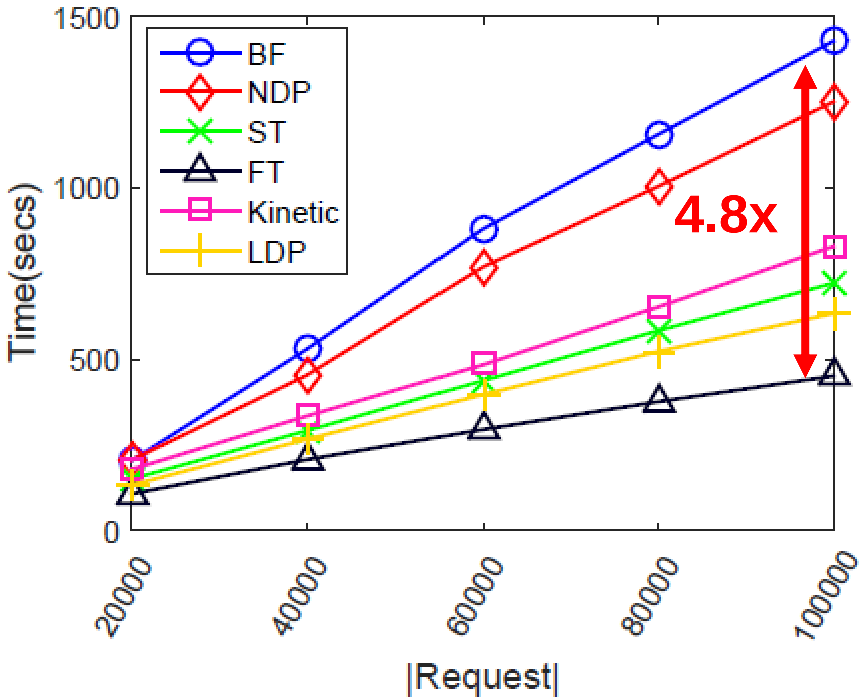
- Compared Algorithms:

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**	$\boldsymbol{O(n^2)}$	√	√
	ST	$\boldsymbol{O(n \log n)}$	√	√
	FT	$\boldsymbol{O(n)}$	√	√

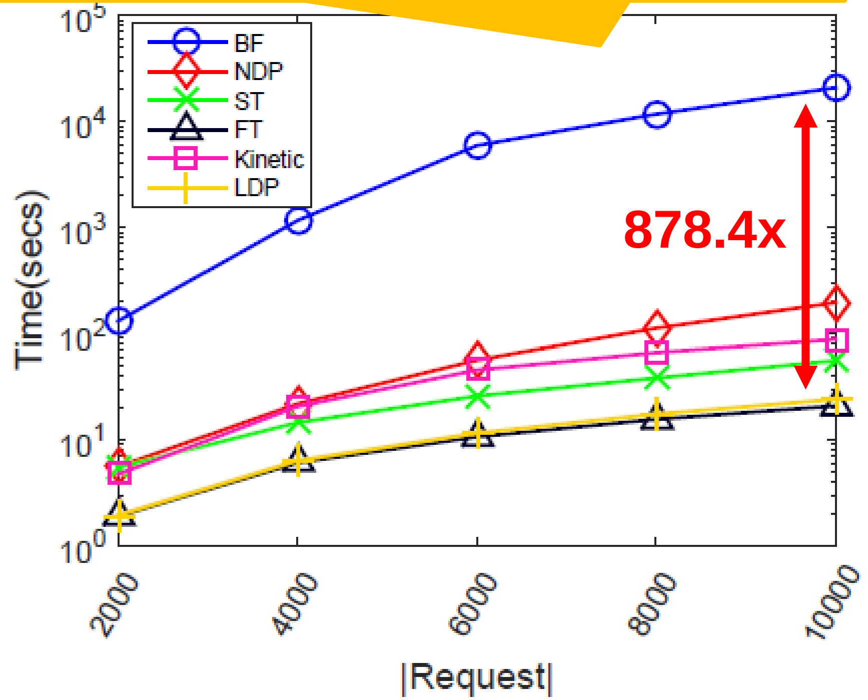
Experiments: Result

- 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

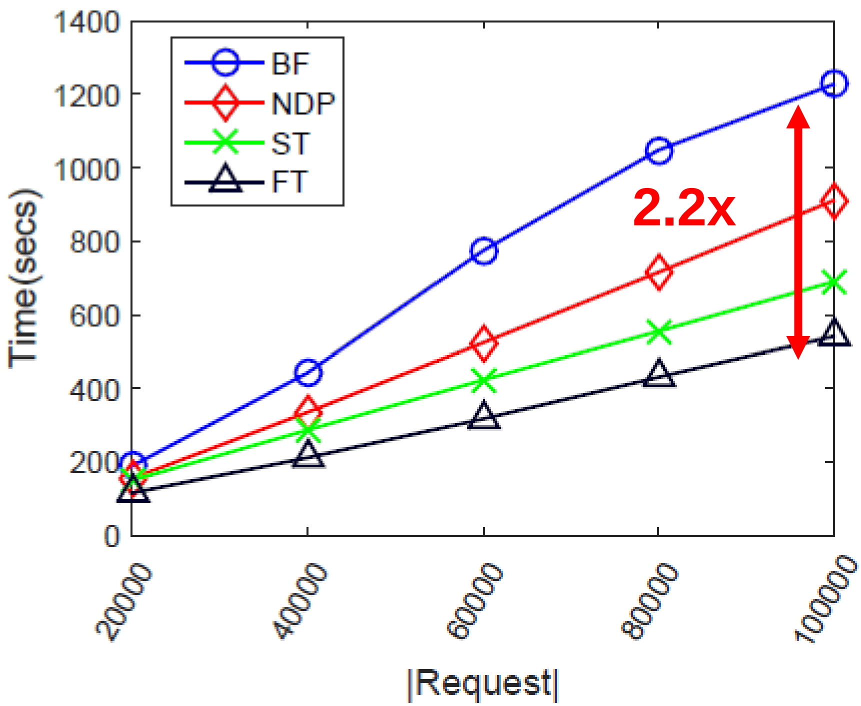


Dataset: logistics

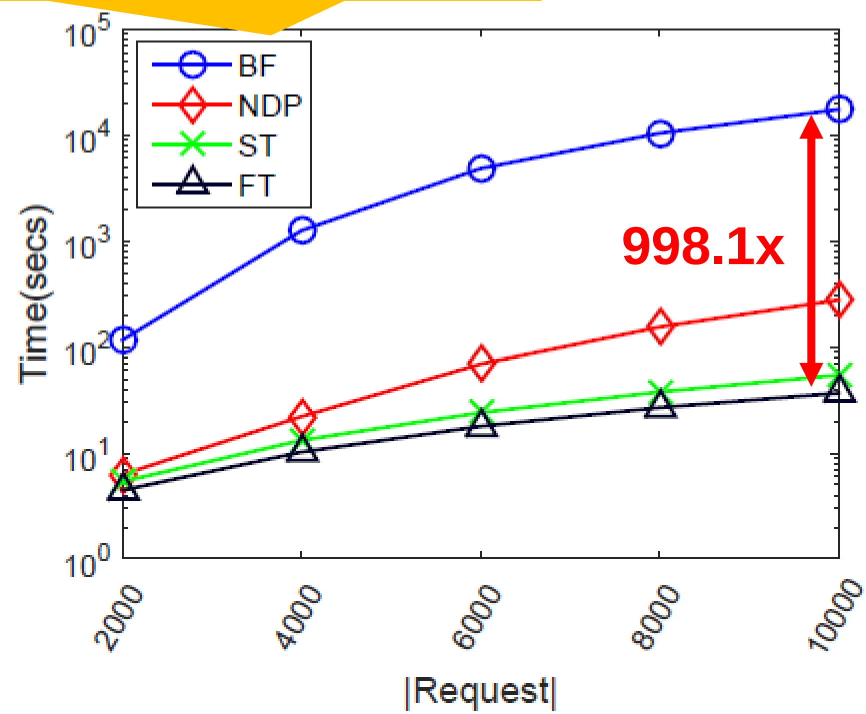
Experiments: Result

- 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

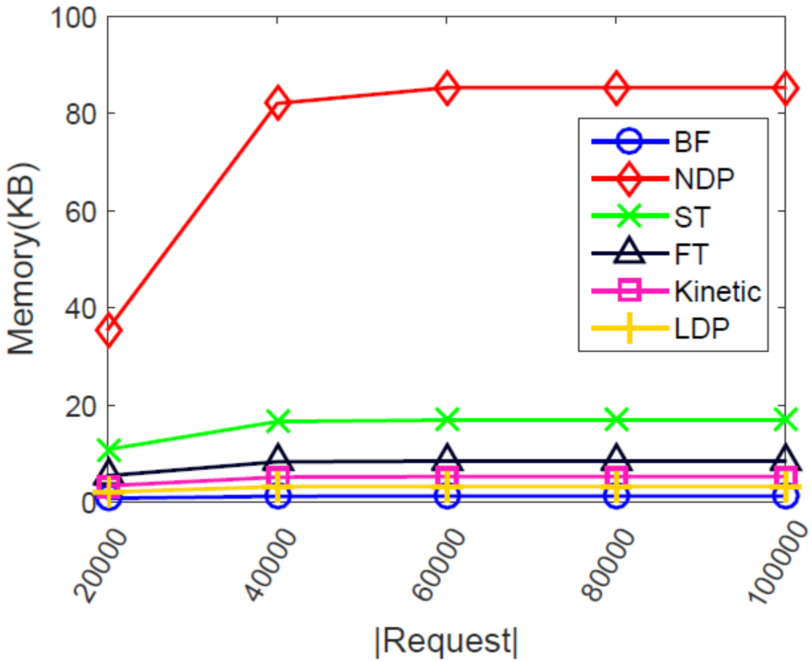


Dataset: logistics

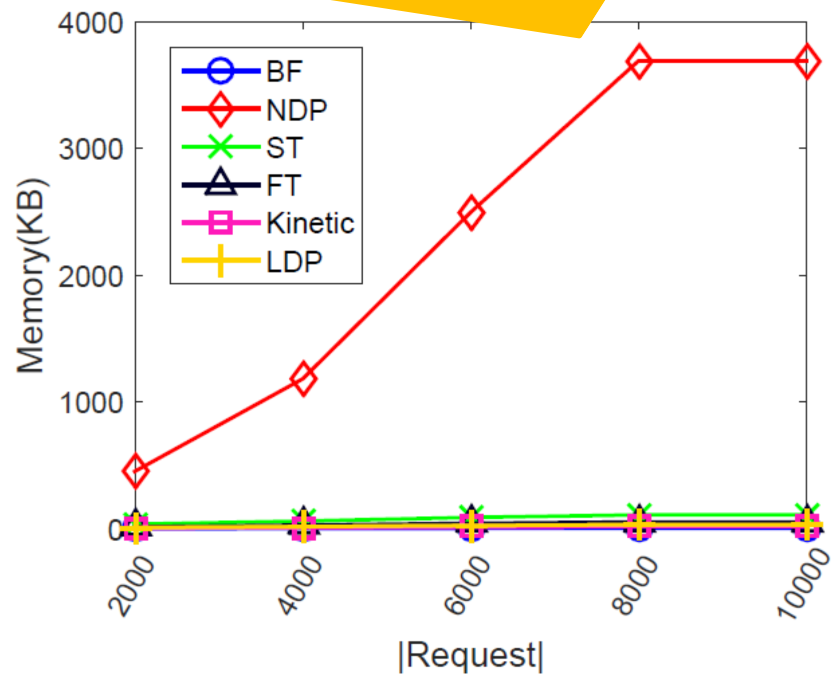
Experiments: Result

- Memory
  - The gap of memory cost among algorithms (except

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



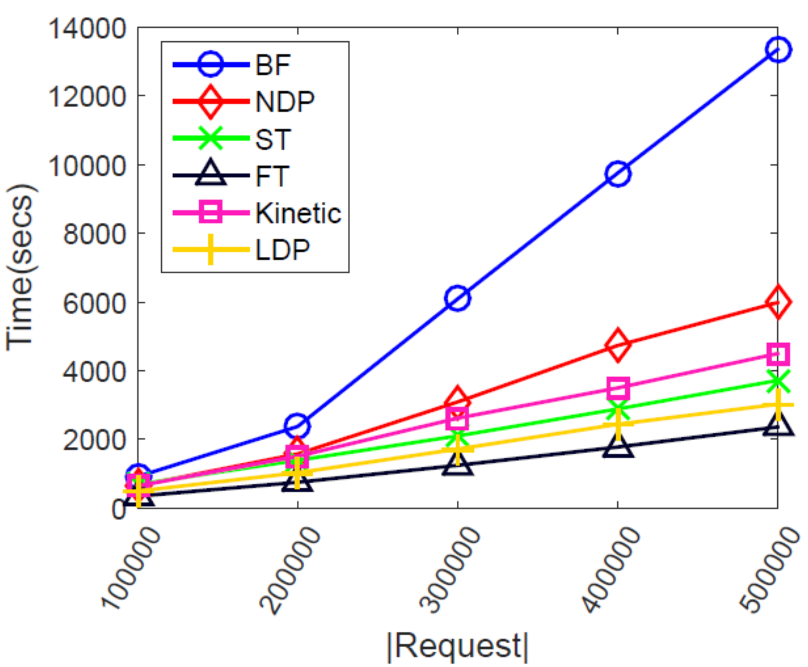
Dataset: Taxi



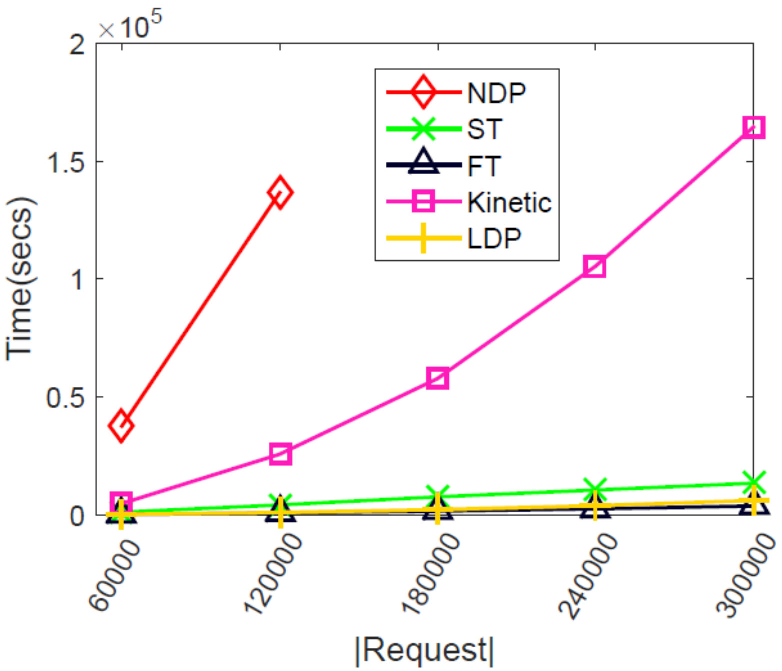
Dataset: logistics

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