Online Ridesharing with Meeting Points

Jiachuan Wang[1], Peng Cheng[2], Libin Zheng[3],Lei Chen[1], Wenjie Zhang[4]
[1]Hong Kong University of Science and Technology, Hong Kong, China
[2]East China Normal University, Shanghai, China
[3]Sun Yat-sen University, Guangzhou, China
[4]The University of New South Wales, Australia

Outline

- Background and Motivation
- The Meeting-Point-based Online Ridesharing Problem
- Framework Overview
- Methods
- Experimental Evaluation
- Summary

Ridesharing in the World

- Online platforms for ridesharing grows rapidly.
    - Each driver can serve more than one request when their routes have common sub-routes

Uber

Lyft

DiDi

Route Planning for Ridesharing

- Effective/efficient route planning strategy is highly demanded due to:
  - A large number of dynamically arriving requests
  - A large number of drivers
  - A large number of possible routes allowing share
  - Limited response time

New Mode: Meeting Points

Traditional route planning

- Requests are posted with source locations and destination locations
- Platform organizes drivers to pass these locations and serve riders

New Mode: Meeting Points

Traditional route planning

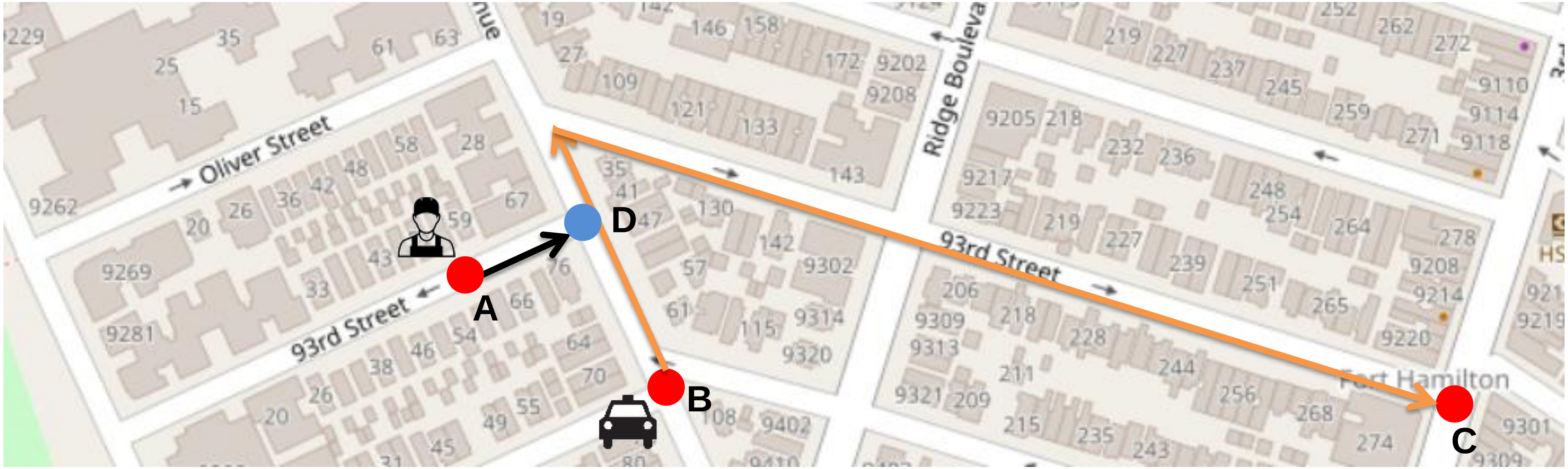
- Requests are posted with source locations and destination locations
- Platform organizes drivers to pass these locations and serve riders

However, due to the complex topology of the city road network, some locations (e.g., **A** and **B**) are **spatially close** to each other but **hard to access** for drivers.

New Mode: Meeting Points

Route planning with Meeting Points

- Meeting points (MP for short) are introduced as alternative locations for pick-up/drop-off locations of requests.
- E.g., driver and riders now **meet at D**.
- Short walk (**A**→D) of riders, large overall profit!



Problem of Ridesharing with Meeting Point (MP)

- Existing researches [1, 2] for MP are offline
  - **Inefficiency**: cannot serve **large-scale online** applications

- MP is not well-explored in the industry
  - Express Pool (Uber) encourages riders to walk to Express spots (meeting points) for efficient routing
  - **Inflexible**: **wait** until a group of requests has a shareable route and pick up them **together** like at a bus station [3]

[1] Mitja Stiglic , et al. 2015
[2] Meng Zhao, et al. 2018
[3] Uber Express Just like a Bus. https://gizmodo.com/i-tried-uber-snew-pool-express-service-and-honestly-j-1823190462

Motivation

- Some vertices are more convenient to come and go and thus "**popular**"
  - E.g., vertices close to highways and downtown

- With flexible MPs, it is possible to **serve more** requests at or near those "**popular**" vertices, which makes them even **more frequently** used.

- These vertices serve as the **skeleton** of the road network
  - **Effectiveness**: estimate and select these popular vertices
  - **Efficiency**: fast algorithms especially on popular vertices

Motivation

- The requirement for a **road network skeleton** motivates us to take advantage of k-skip cover V^* [1], which is a selected subset of vertices to be the **skeleton** of a graph G.

> - We call a vertex set V^* ***k-skip cover*** if for any shortest path of length k on a graph, there is at least one of its vertices $\in V^*$.

- To minimize the size of V^*, we need to find the most "popular" and convenient vertices, which frequently appear in shortest paths, which coincide with our requirement for meeting points.

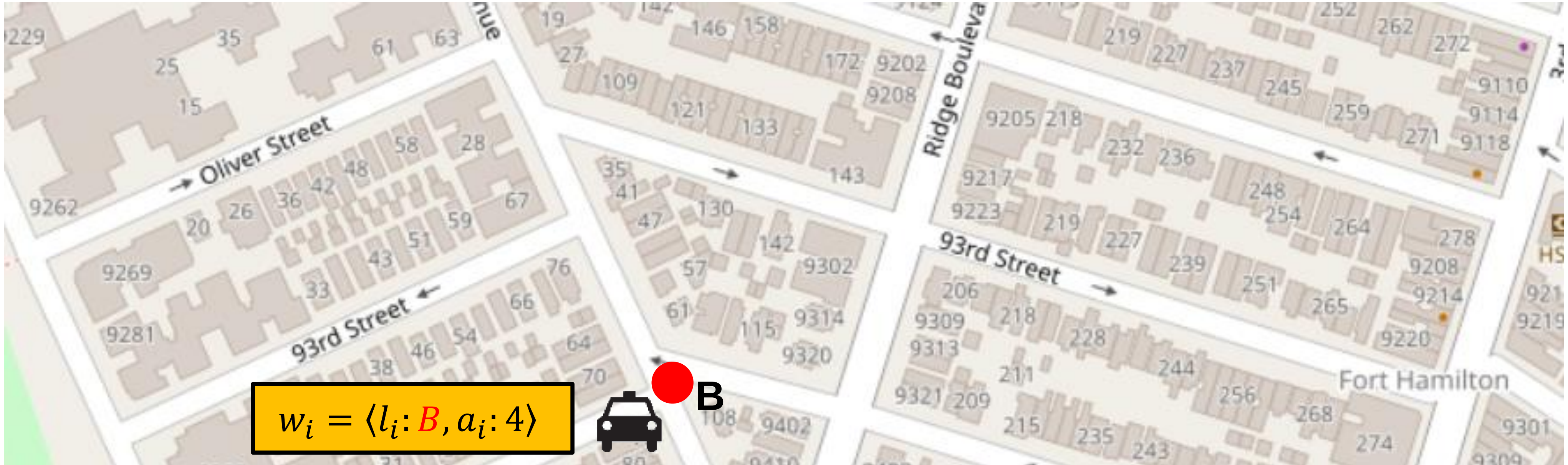
[1] Yufei Tao, et al. 2011

Outline

- Background and Motivation
- The Meeting-Point-based Online Ridesharing Problem
- Framework Overview
- Methods
- Experimental Evaluation
- Summary

The Meeting-Point-based Online Ridesharing Problem

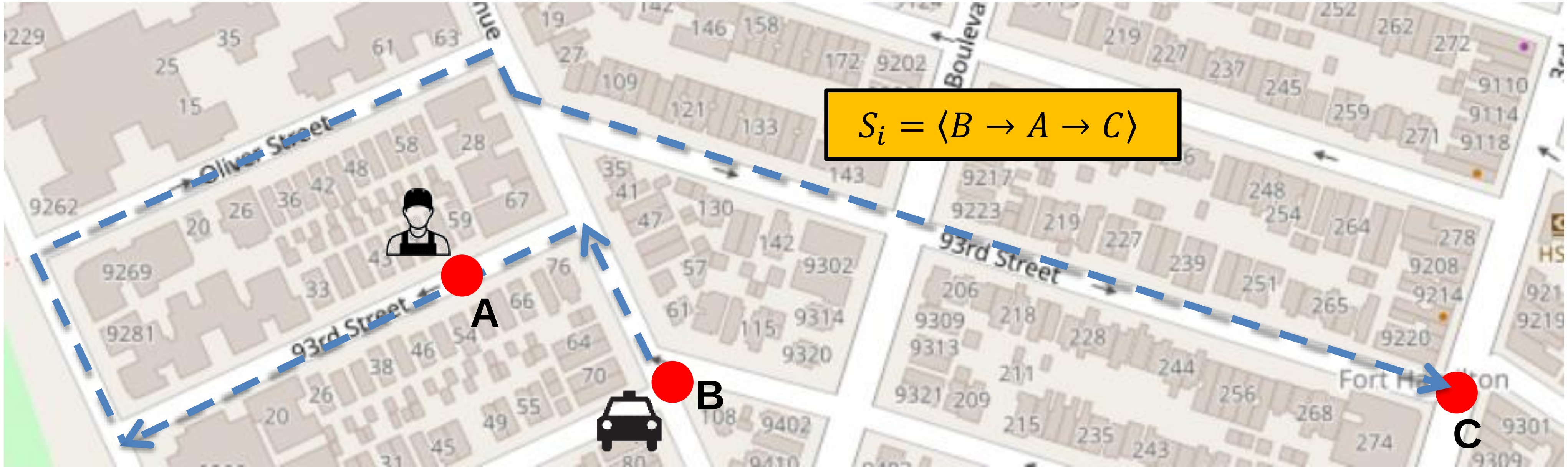
- Drivers
  - A set of n drivers $W = \{w_1, w_2, \dots, w_n\}$
  - Each is defined by $w_i = \langle l_i, a_i \rangle$ with current location l_i and capacity limitation a_i



The Meeting-Point-based Online Ridesharing Problem

- Drivers
  - A set of n drivers $W = \{w_1, w_2, \dots, w_n\}$
  - Each is defined by $w_i = \langle l_i, a_i \rangle$ with current location l_i and capacity limitation a_i
- Requests
  - A set of m requests $R = \{r_1, r_2, \dots, r_n\}$
  - Traditionally, each is defined by $r_j = \langle s_i, e_i, tr_j, tp_j, td_j, p_j, a_j \rangle$, where:
    - s_i / e_i for source/destination locations;
    - $tr_j / tp_j / td_j$ for time of release/pick-up deadline/drop-off deadline;
    - p_j for rejection penalty;
    - a_j for capacity.

The Meeting-Point-based Online Ridesharing Problem

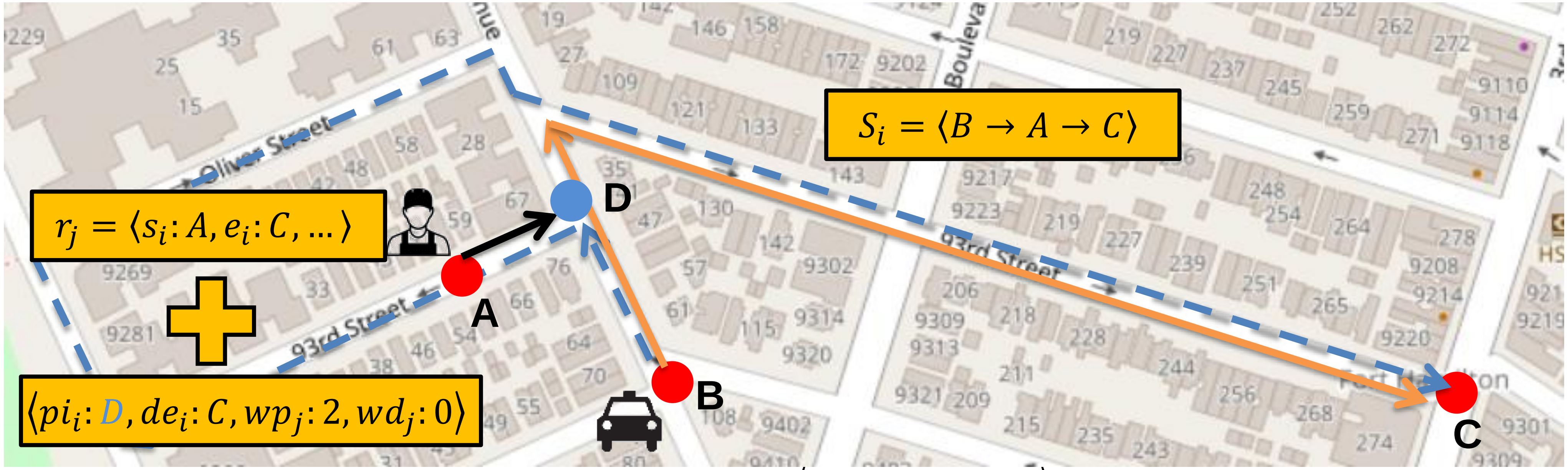


- Traditional route planning:
  - Assign each driver w_i a **route S_i, which is a sequence of s_j/e_j** under the time and capacity constraints.
  - Minimizing a unified cost of:

$$\alpha \sum_{w_i \in W} D(S_{w_i}) + \sum_{r_j \in \bar{R}} p_j$$

The driving cost of routes — The penalty for rejected requests

The Meeting-Point-based Online Ridesharing Problem



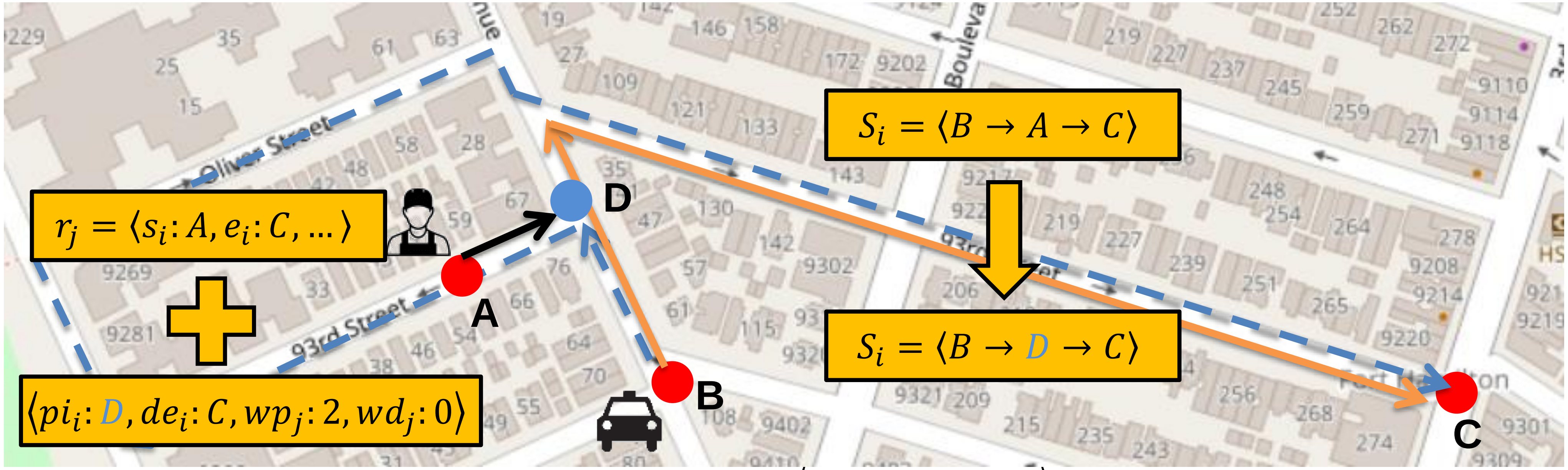
- With **meeting point**, an assigned request has $\langle pi_j, de_j, wp_j, wd_j \rangle$ in addition, where:
  - pi_j / de_j for pick-up and drop-off locations
  - wp_j, wd_j for time of riders **walking** before picked up and after dropped off.

$$\alpha \sum_{w_i \in W} D(S_{w_i}) + \sum_{r_j \in \bar{R}} p_j$$

The driving cost of routes

The penalty for rejected requests

The Meeting-Point-based Online Ridesharing Problem



- With **meeting point**, an assigned request has $\langle pi_j, de_j, wp_j, wd_j \rangle$ in addition, where:
  - $\boldsymbol{pi_j/de_j}$ for pick-up and drop-off locations
  - wp_j, wd_j for time of riders walking before picked up and after dropped off.
- **Meeting-Point**-based route planning:
  - Assign each driver w_i a route S_i, which is a sequence of ~~s_i/e_i~~ $\boldsymbol{pi_j/de_j}$.
  - Minimizing a unified cost of:

$$\alpha \sum_{w_i \in W} D(S_{w_i}) + \sum_{r_j \in \bar{R}} p_j + \beta \sum_{r_j \in \hat{R}} (wp_j + wd_j)$$

The driving cost of routes | The penalty for rejected requests | The walking cost of requests

The Meeting-Point-based Online Ridesharing Problem

We prove the MORP problem is **NP-hard** by reducing it from the basic route planning problem[1] for shareable mobility services.

We further show that **no** deterministic nor randomized algorithm can guarantee a **constant Competitive Ratio.**

[1] Yongxin Tong et al. 2018

- **Meeting-Point**-based route planning:
  - Assign each driver w_i a route S_i, which is a sequence of ~~s_i/e_i~~ $\boldsymbol{pi_j/de_j}$.
  - Minimizing a unified cost of:

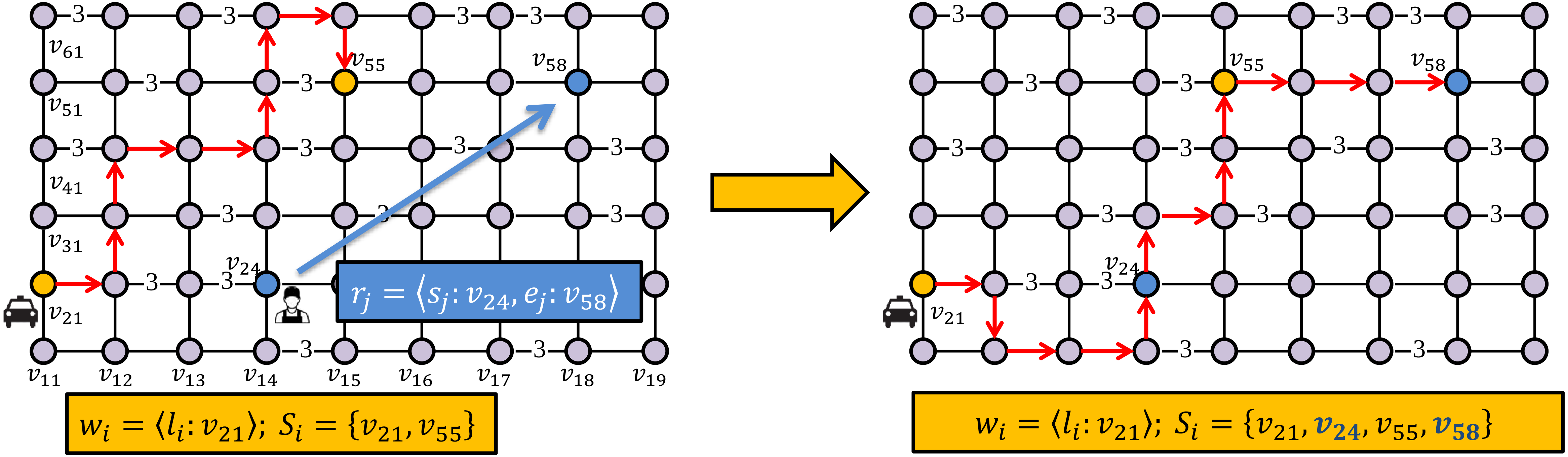
$$\alpha \sum_{w_i \in W} D(S_{w_i}) + \sum_{r_j \in \bar{R}} p_j + \beta \sum_{r_j \in \hat{R}} (wp_j + wd_j)$$

The driving cost of routes | The penalty for rejected requests | The walking cost of requests

Outline

- Background and Motivation
- The Meeting-Point-based Online Ridesharing Problem
- Framework Overview
- Methods
- Experimental Evaluation
- Summary

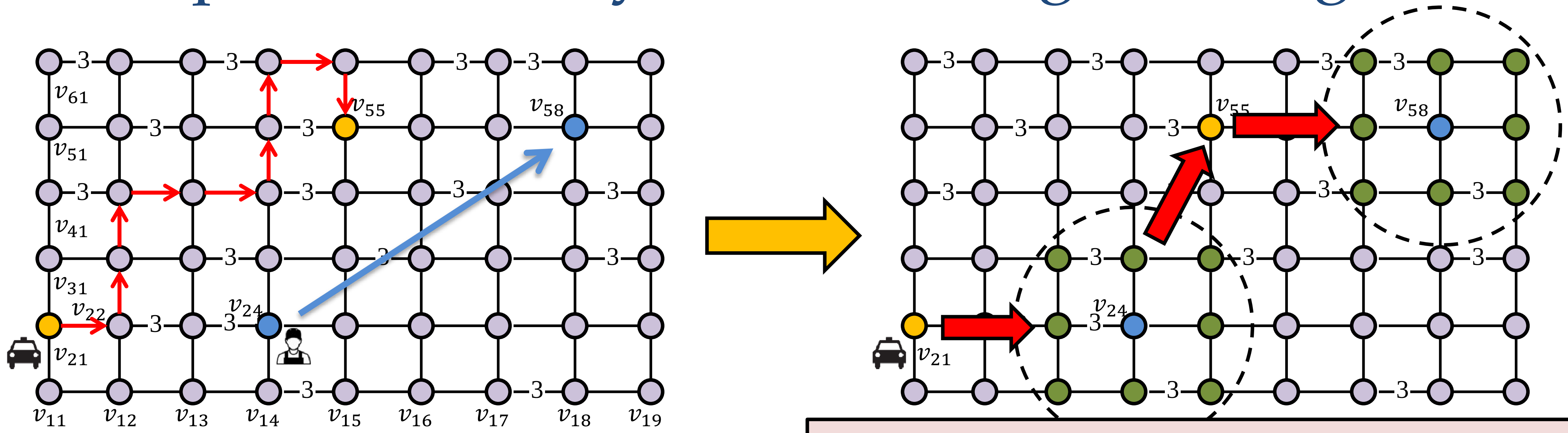
Existing method - Insertion



Insertion is an effective local optimal algorithm for route planning, which has linear ($O(|S_i|)$) time complexity [1].
It involves **shortest path queries** between the current route (S_i) and inserted locations (s_j, e_j).

[1] Yongxin Tong et al. 2018

Adapt Insertion by Enumerating Meeting Points

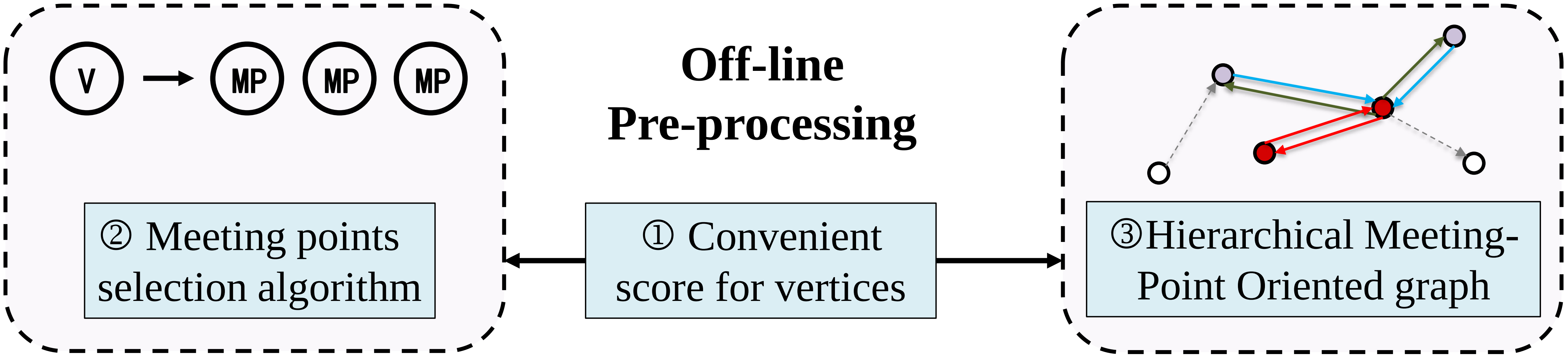


With Meeting Points, possible insertion location pairs are **greatly increased** (1 × 1 to 9 × 9) in the Figure.

By adapting insertion for MORP, it involves **shortest path queries** between the current route (S_i) and all possible pairs of meeting points near (s_j, e_j).

If a vertex has k meeting points on average, the computational cost increases by $k \times k$ times, which is unacceptable.

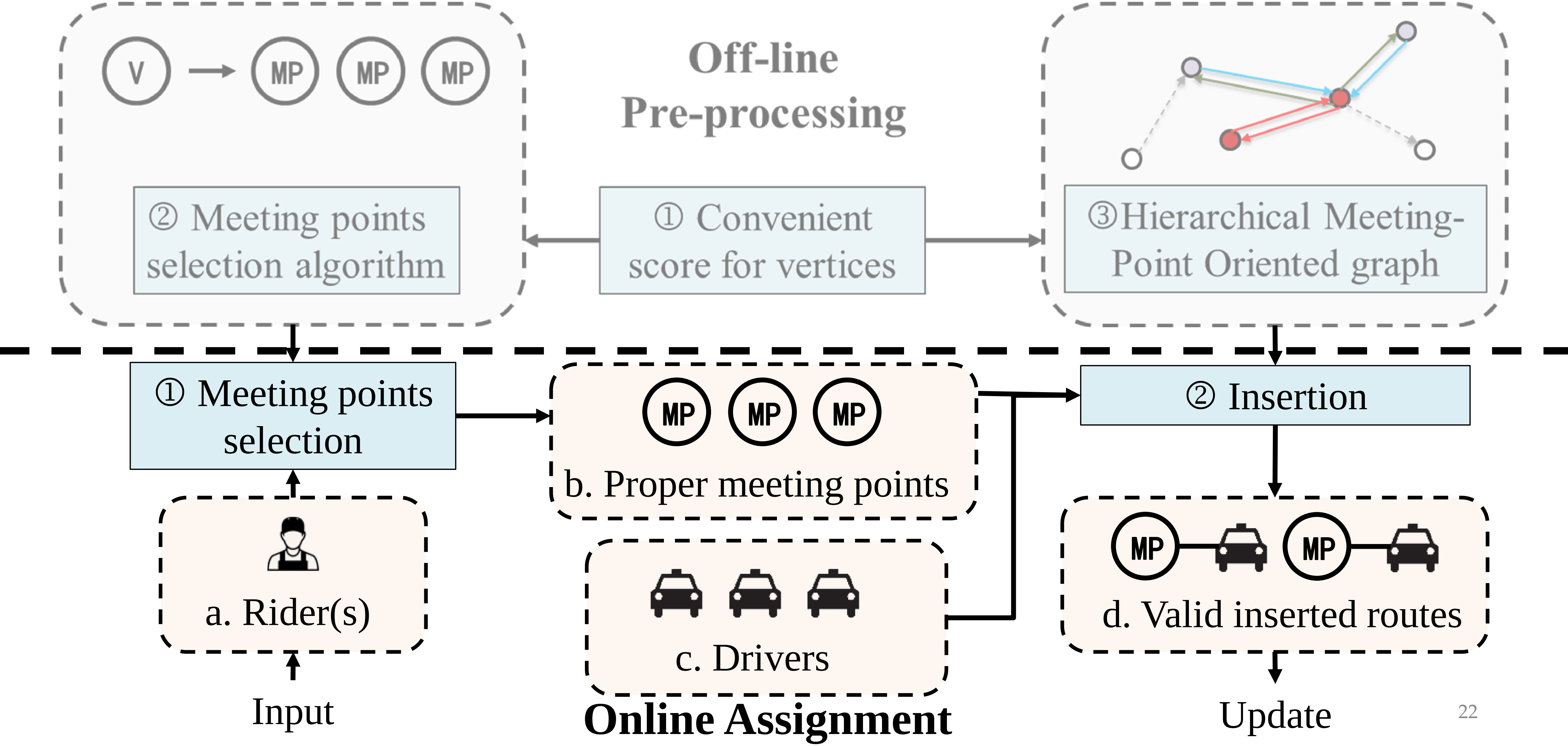
Framework



Selecting optimal meeting points (MPs) **online** is time-consuming

1) Prepare MPs for each vertex **offline** to reduce the search space
2) Design data structure for faster queries.

Framework



V
MP
MP
MP
Off-line
Pre-processing
② Meeting points selection algorithm
① Convenient score for vertices
③Hierarchical Meeting-Point Oriented graph
① Meeting points selection
MP
MP
MP
b. Proper meeting points
② Insertion
a. Rider(s)
c. Drivers
MP
MP
d. Valid inserted routes
Input
Online Assignment
Update

Outline

- Background and Motivation
- The Meeting-Point-based Online Ridesharing Problem
- Framework Overview
- Methods
- Experimental Evaluation
- Summary

Select Meeting Point Candidates

- Quantize how "convenient" a vertex is for transportation
  - MP candidates should easily get to and conveniently reach other vertices.

  - Given vertex u, define its n_r nearest vertices as reference vertices $n_o(u)$.
    - **Equivalent Out Cost** of u: the average distance towards its reference vertices

$$ECO(u) = \frac{\sum_{v \in n_o(u)} SP_c(u, v)}{n_r}$$

  - Similarly, reverse the graph we can have **Equivalent Inward Cost**

$$ECI(u) = \frac{\sum_{v \in n_i(u)} SP_c(v, u)}{n_r}$$

Select Meeting Point Candidates

- Quantize how "convenient" a vertex is for transportation
  - MP candidates should easily get to and conveniently reach other vertices.

  - Now we can rank the candidate MPs $\{v_1, v_2, \dots, v_n\}$ of a vertex u by Serving-Cost Score

$$SCS(u, v_i) = \boxed{\beta \cdot SP_p(u, v_i)} + \boxed{\alpha\big(ECI(v_i) + ECO(v_i)\big)}$$

Expected cost from walking

Expected cost from driving

Based on these statistics from shortest path queries, an $O(|V|)$ Local-Flexibility-Filter Algorithm is proposed to select MPs for each vertex **offline**.

Hierarchical Meeting-Point Oriented graph

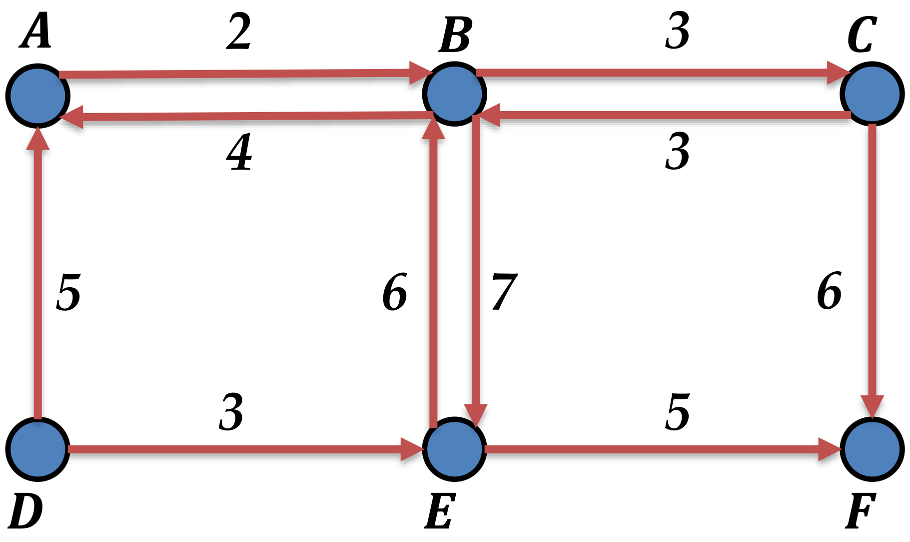
- With MPs, assigned routes can be concentrated on the convenient vertices

- We give the hierarchical order over the vertex set V:
  - **Defective vertices** $\boldsymbol{V_{de}}$ They are inconvenient to access.
  - **Core vertices** $\boldsymbol{V_{co}}$ They are used as MPs frequently.
  - **Sub-level vertices** $\boldsymbol{V_{su}}$ The remaining vertices are classified as sub-level vertices.

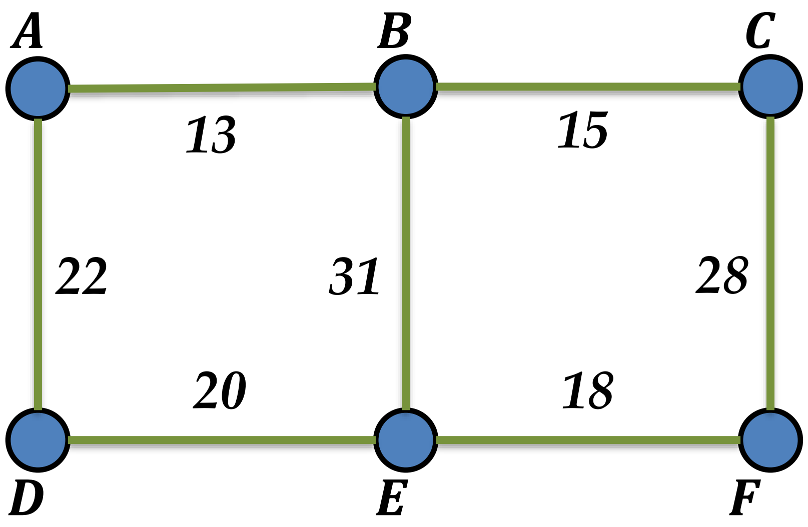
Hierarchical Meeting-Point Oriented graph

Defective vertices V_{de}

- We aim to remove the unwelcomed vertices in traditional ridesharing, which can be alternatively served by meeting points now.
- We propose a method to avoid two potential costs from vertex removal:
  - **The detour cost** A path containing removed vertex u no longer exists.
  - **The inaccessibility cost** The potential reject penalty of requests at the removed vertex.



(a) Graph for car



(b) Graph for passenger

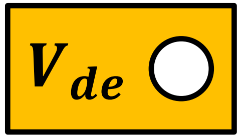
Hierarchical Meeting-Point Oriented graph

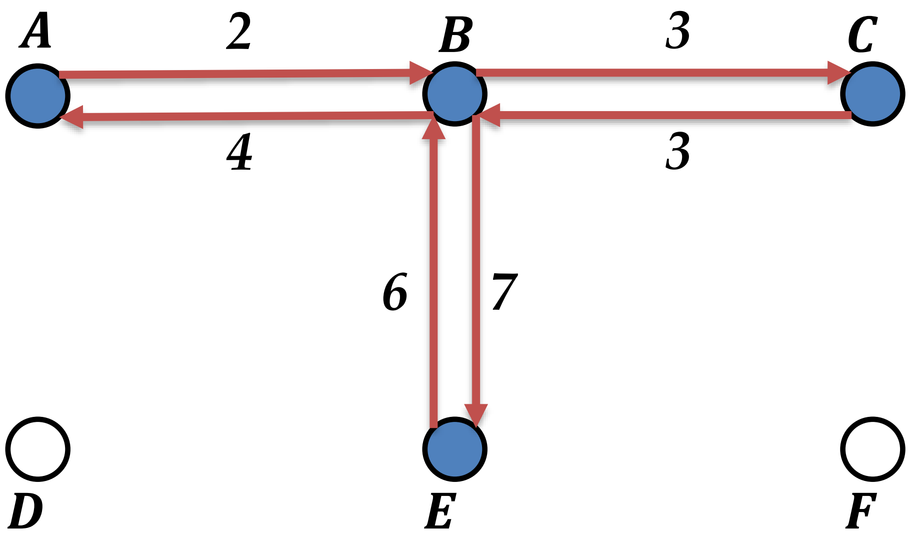
Defective vertices V_{de}

- A 3-phase $O(NlogN)$ DVS algorithm is proposed with theoretical guarantee

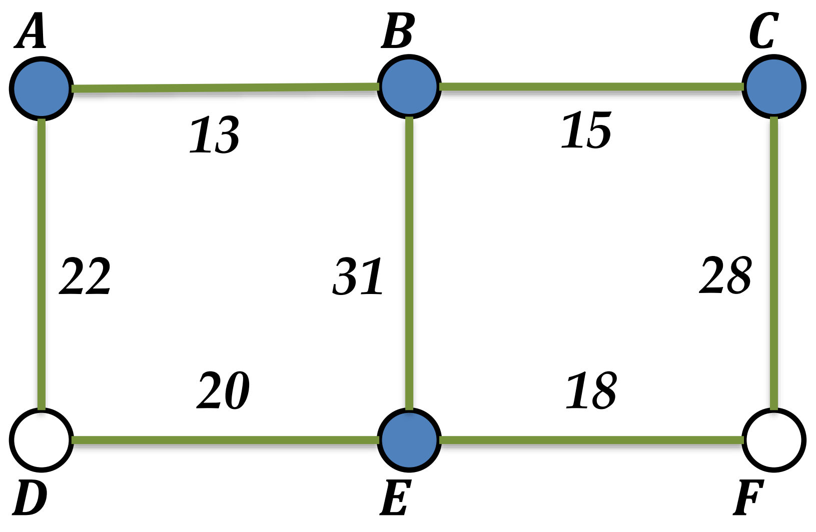
LEMMA 6.2. *Removing all vertices selected by the DVS algorithm from G_c with their edges leads to no detour cost.*

LEMMA 6.3. *$\forall u \in V$ is accessible after removing vertices selected by the DVS algorithm from G_c with MPs.*





(a) Graph for car

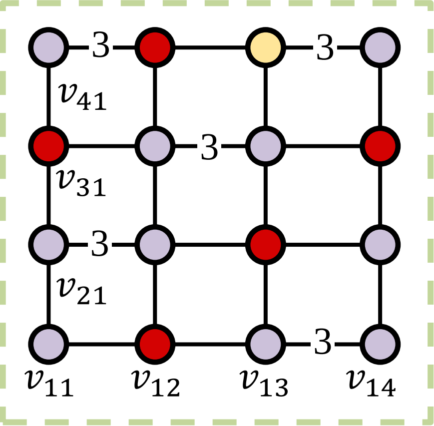


(b) Graph for passenger

Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.

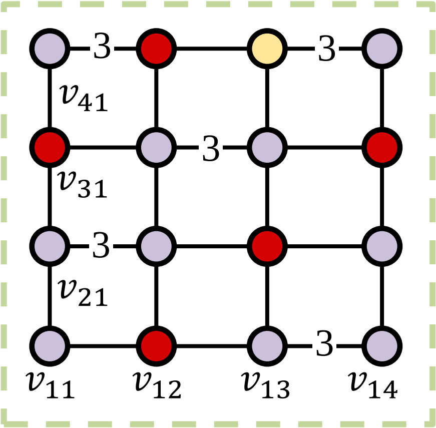


A graph and its **2-skip cover V^***
(Any shortest path longer than **2**
contains at least one of its vertex)

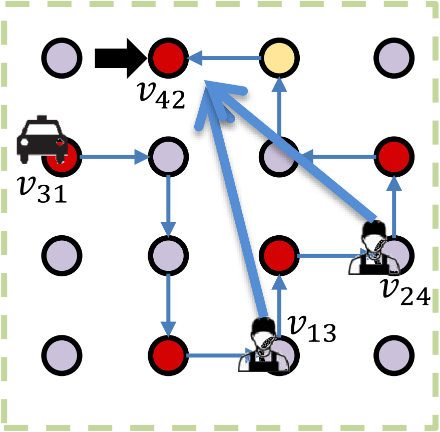
Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.



A graph and its **2-skip cover V^***

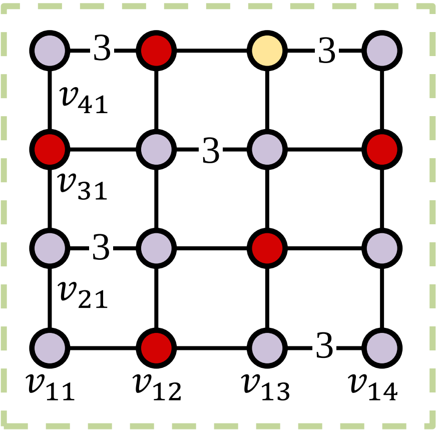


A car picks up 2 requests
in the **traditional** mode

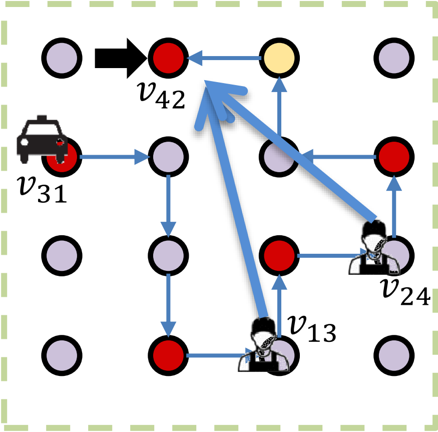
Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

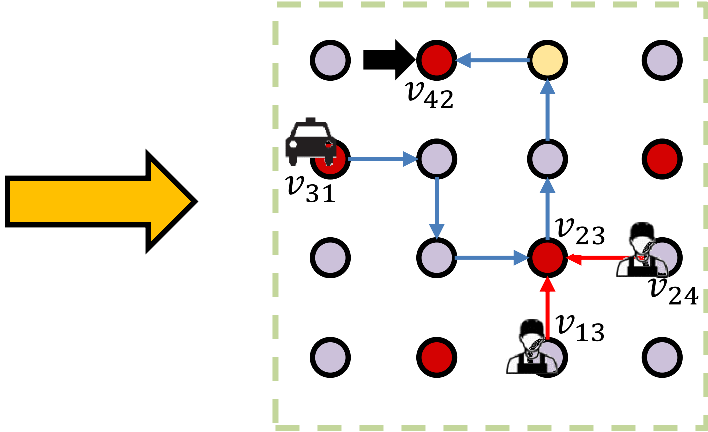
- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.



A graph and its **2-skip cover V^***



A car picks up 2 requests in the traditional mode



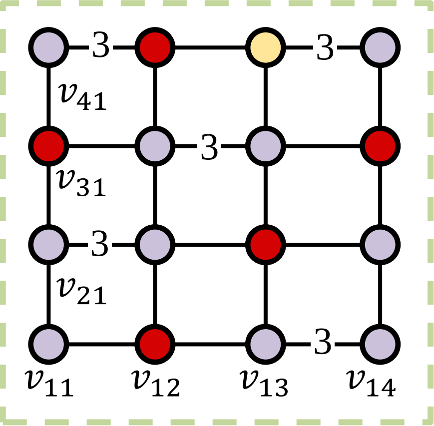
A car picks up 2 requests in the **meeting point** mode (meet at v_{23})

Both of ***k*-skip cover** and **meeting points** expect "convenient" vertices

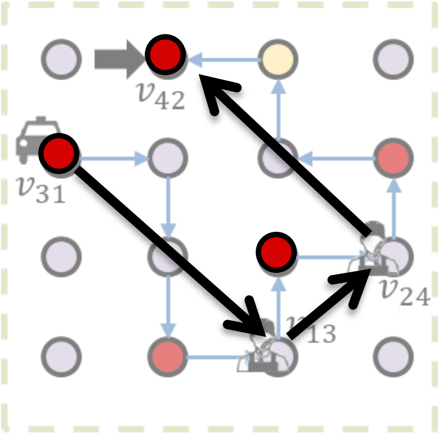
Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

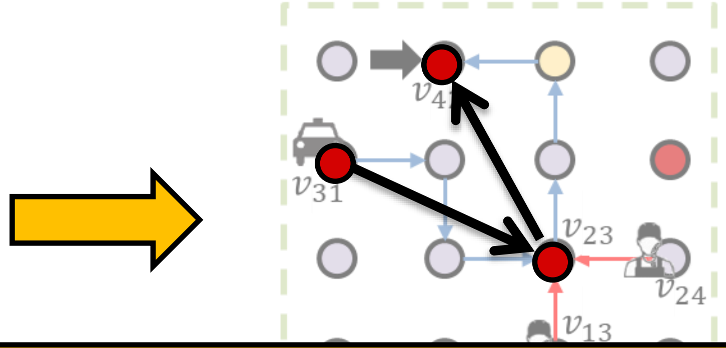
- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.



A graph and its **2-skip cover V^***



A car picks up 2 requests in the traditional mode



Meeting point -> more queries between k-skip cover

A car picks up 2 requests in the **meeting point** mode (meet at v_{23})

Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.
  - **V_{co} is a k-skip cover** on the updated graph without V_{de}
  - Proportion factor ϵ of vertices have at least one vertex $u \in V_{co}$ as its MP candidate

Meeting point -> more queries between k-skip cover

***k*-skip cover** -> faster inner queries to improve efficiency

Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.
  - **V_{co} is a k-skip cover** on the updated graph without V_{de}
  - Proportion factor ϵ of vertices have at least one vertex $u \in V_{co}$ as its MP candidate

Formulate as an integer linear program

$$\begin{aligned} \min \quad & \sum_{u \in V - V_{de}} \delta_u \\ \text{s.t.} \quad & \phi_v + \sum_{u: v \in MS(u)} \delta_u \geq 1 && \forall v \in V_p \\ & \sum_{v \in V_p} \phi_v \leq (1-\epsilon) |V_p| \\ & \delta_u \in \{0,1\} && \forall u \in V - V_{de}, \\ & \phi_v \in \{0,1\} && \forall v \in V_p, \end{aligned}$$

Propose algorithm with **theoretical bound** on # of V_{co}

Algorithm 5: Core Vertices Selection Algorithm

Input: All the vertices V and defective vertices V_{de}, list ECO and ECI from Algo.3, MP candidates MC

Output: The set of core vertices V_{co}

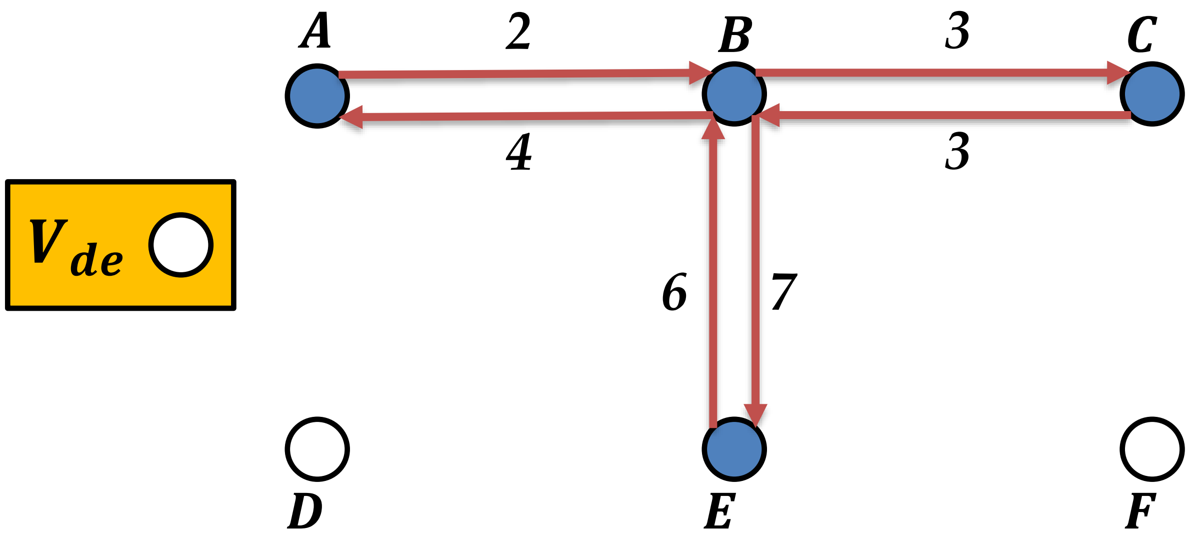
1. Initialize candidate serving set for each $u \in V - V_{de}$ as $\emptyset$.
2. **foreach** $v \in V_p$ **do**
3. **foreach** $u \in MC(v)$ **do**
4. Add v to $MS(u)$;
5. Solve the partial set cover problem using the algorithm in [19], where the weights of sets are substituted with $ECO + ECI$ in its sorting step. Finally, record its cost $cost_u$ of choosing each $MS(u)$ as the highest cost set. Denote its output as V'_{co}.
6. Initialize the core vertices $V_{co} = V - V_{de}$.
7. Sort the vertices $u \in V - V'_{co} - V_{de}$ in decreasing order of $cost_u$. Check them one-by-one and remove a vertex from V_{co} if the V_{co} is still a k-skip cover.
8. Sort and check the vertices $u \in V'_{co}$ in decreasing order of $cost_u$. Remove a vertex v from V_{co} if $V_{co} - \{v\}$ is still a k-skip cover and the MSs of $V_{co} - \{v\}$ cover $\epsilon \cdot |V_p|$ vertices.
9. **return** V_{co}

LEMMA 6.5. *Assume that we have N vertices in total, with M set as optimal solution for the attribute* (ii)*, the upper bound of the size of core vertex set is* $\sigma(k) = \max(\frac{N}{k} log \frac{N}{k}, nc_m \cdot M)$.

Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.
  - **V_{co} is a k-skip cover** on the updated graph without V_{de}
  - Proportion factor ϵ of vertices have at least one vertex $u \in V_{co}$ as its MP candidate

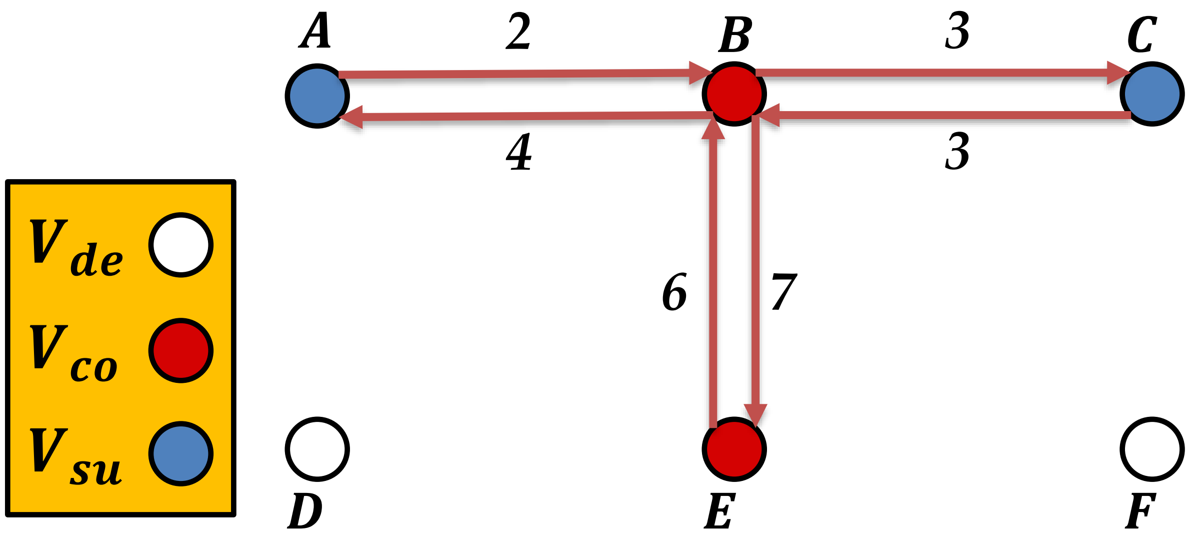


(a) Graph for car

Hierarchical Meeting-Point Oriented graph

Core vertices V_{co}

- Select "convenient" vertices as the skeleton of the graph
- k-skip cover has good compatibility with the meeting point: use it as backbone.
  - **V_{co} is a k-skip cover** on the updated graph without V_{de}
  - Proportion factor ϵ of vertices have at least one vertex $u \in V_{co}$ as its MP candidate

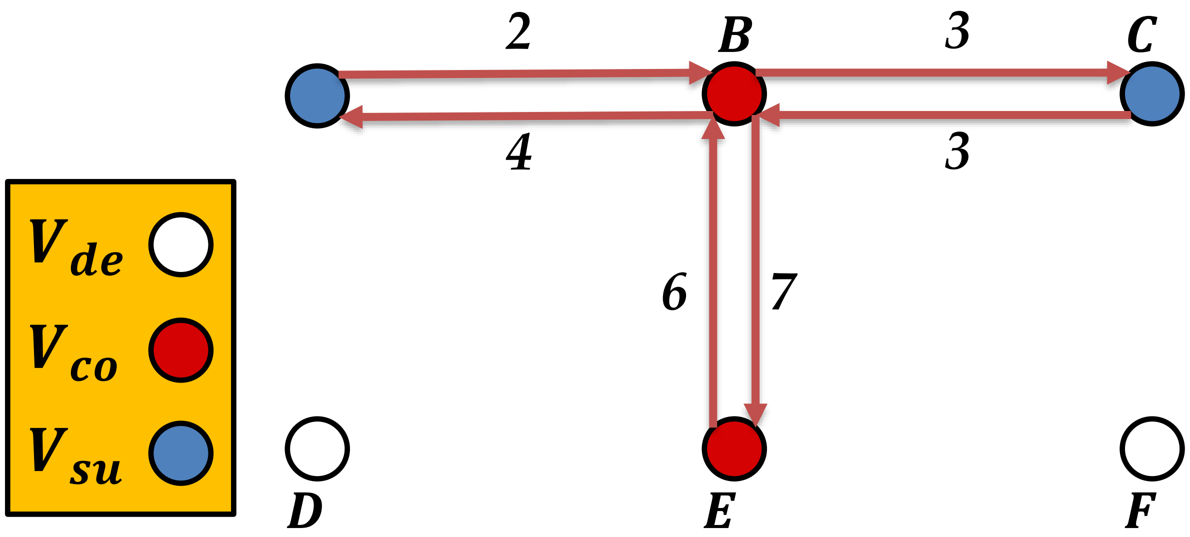


(a) Graph for car

Hierarchical Meeting-Point Oriented graph

Graph construction

- Vertices: previously obtained V_{de}, V_{co}, V_{su}
- Edges: since V_{co} forms a k-skip cover, we can build super edges among $V_{co} \cup V_{su}$ following existing theory [1]:
  - E_{cc}: super edges between core vertices
  - E_{cc}: super edges from core to sub-level vertices
  - E_{cc}: super edges from sub-level to core vertices

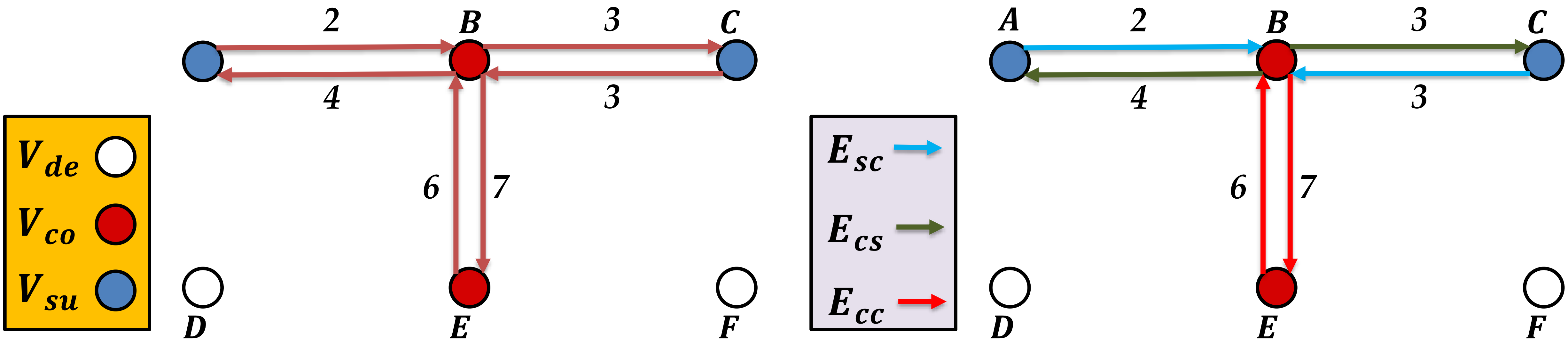


[1] Yufei Tao, et al. 2011

Hierarchical Meeting-Point Oriented graph

Graph construction

- Vertices: previously obtained V_{de}, V_{co}, V_{su}
- Edges: since V_{co} forms a k-skip cover, we can build super edges among $V_{co} \cup V_{su}$ following existing theory [1]:
  - E_{cc}: super edges between core vertices
  - E_{cc}: super edges from core to sub-level vertices
  - E_{cc}: super edges from sub-level to core vertices



[1] Yufei Tao, et al. 2011

HMPO Graph-Based Insertion

Recall that we select MP Candidates ($\boldsymbol{MC(u)}$) for each vertex $\boldsymbol{u}$

- Vertices$\in MC(u)$ are reachable via short walking ⇒ they are close to each other
- One interesting problem is, if inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?

HMPO Graph-Based Insertion

Recall that we select MP Candidates ($\boldsymbol{MC(u)}$) for each vertex $\boldsymbol{u}$

- Vertices$\in MC(u)$ are reachable via short walking $\Rightarrow$ they are close to each other
- One interesting problem is, if inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?

- $\Rightarrow$ We define a new distance correlation, which **bounds the time saving** of switching from v to any other vertices.
- If deducting the saving still cannot meet the time limitation:
  - $\Rightarrow$ prune the whole set $MC(u)$!

HMPO Graph-Based Insertion

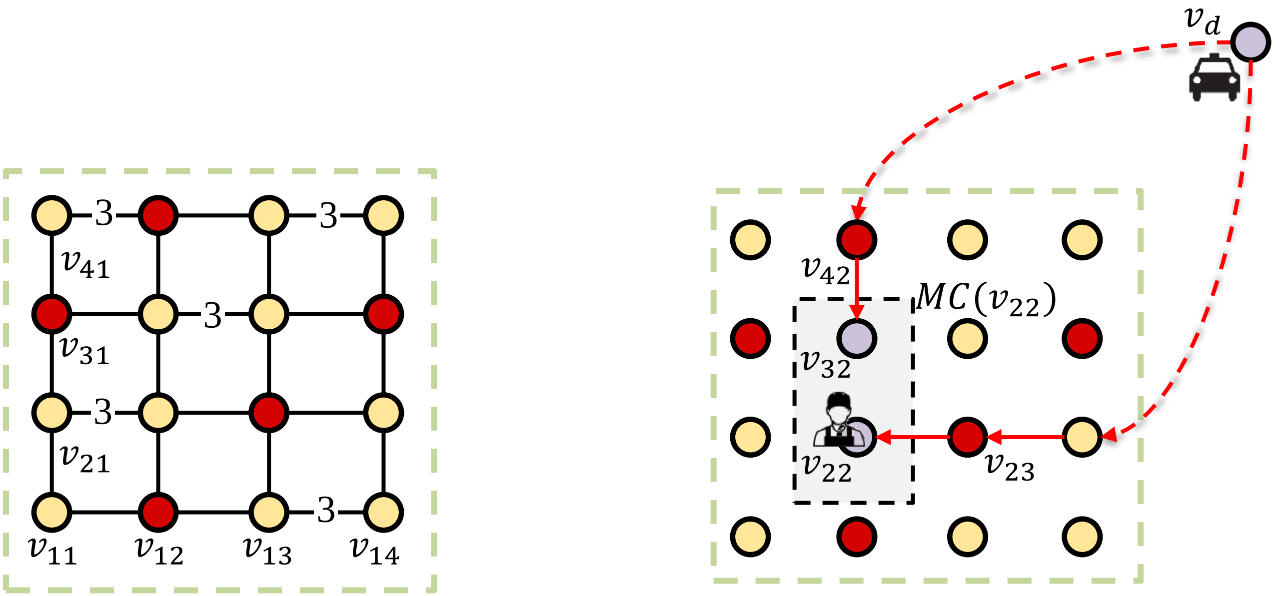
Recall that we select MP Candidates ($MC(u)$) for each vertex u

- Vertices$\in MC(u)$ are reachable via short walking ⇒ they are close to each other
- One interesting problem is, if inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?

- ⇒ We define a new distance correlation, which **bounds the time saving** of switching from v to any other vertices.
- If deducting the saving still cannot meet the time limitation:
  - ⇒ prune the whole set $MC(u)$!

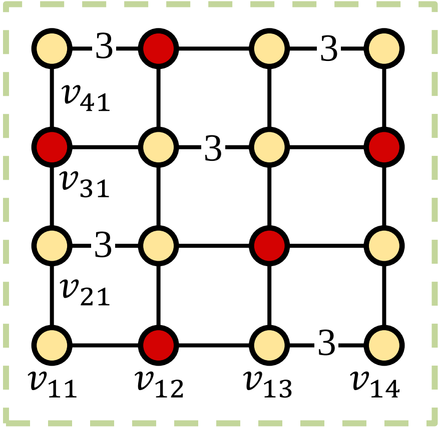
HMPO Graph-Based Insertion

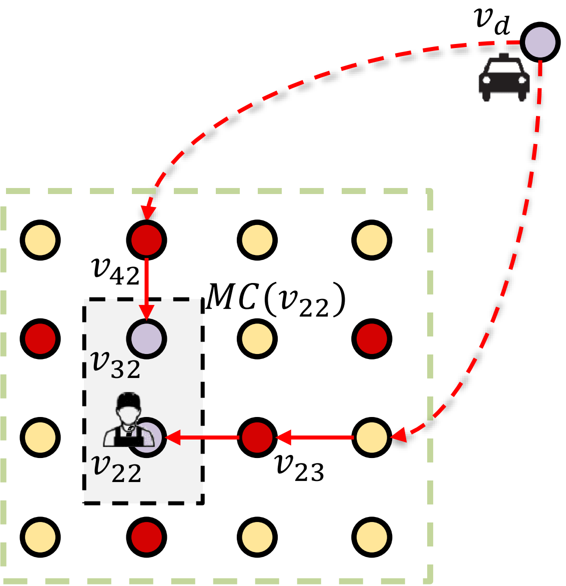
- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{32}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?



HMPO Graph-Based Insertion

- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?

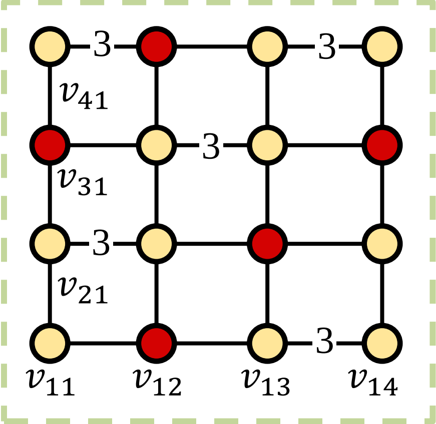




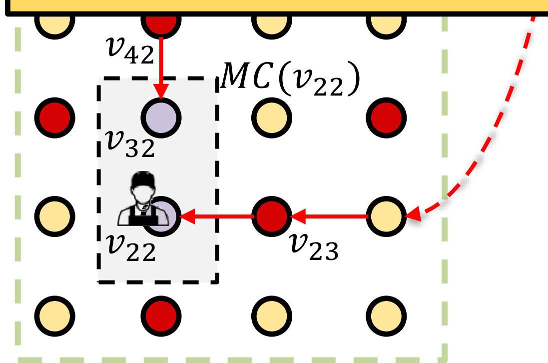
Traditionally, we need to derive **all** the time costs from graph to v_{22} and v_{23} though they are **close** to each other

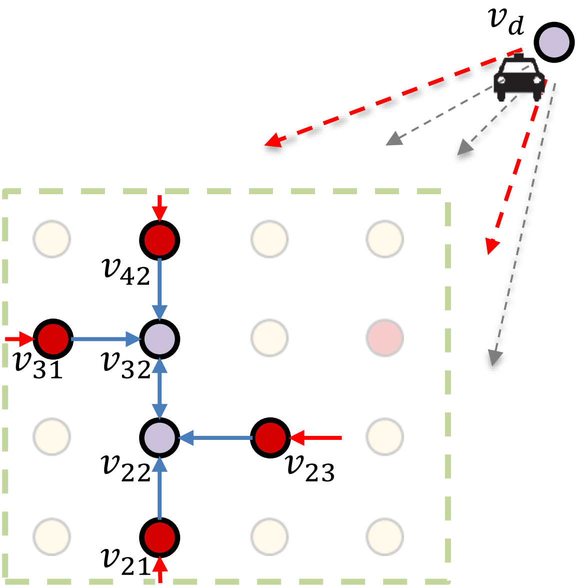
HMPO Graph-Based Insertion

- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?



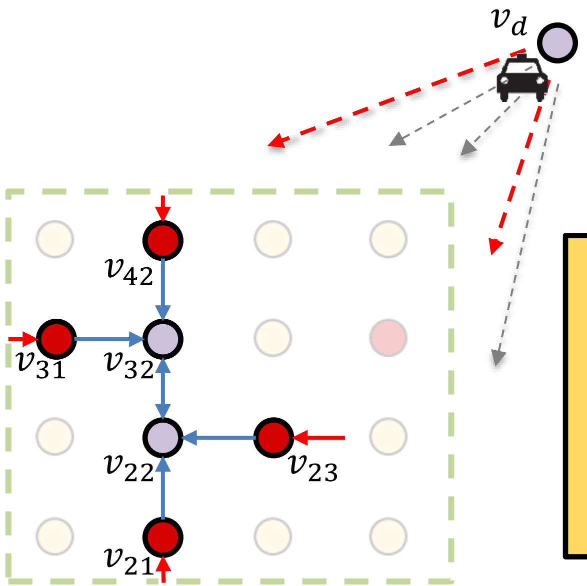
Luckily, the k-skip cover "**cut off**" the shortest paths using the **core vertices**, which enables us to bound the time-saving.





HMPO Graph-Based Insertion

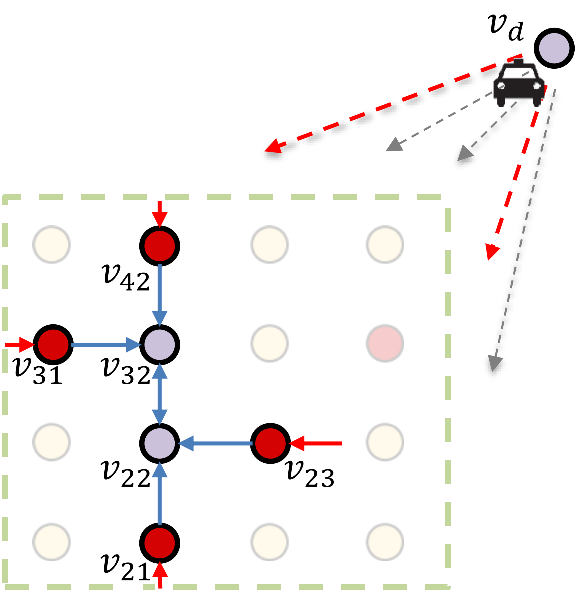
- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?



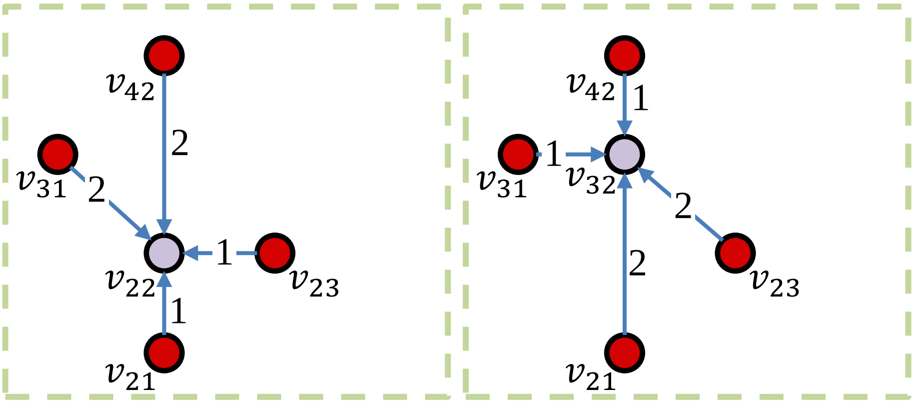
Cut off from core vertices:
If v_d shares no super edges with $\{v_{21}, v_{23}, v_{31}, v_{42}\}$, any shortest paths from v_d to v_{32} and v_{22} need to pass the 4 core vertices

HMPO Graph-Based Insertion

- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?

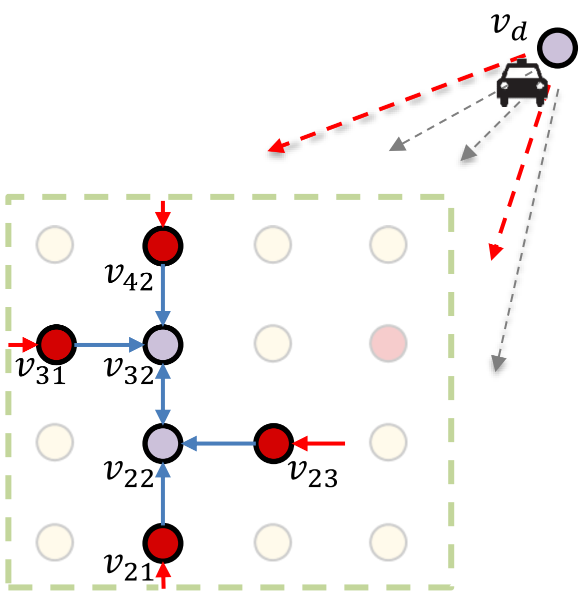


The distance from the 4 core vertices $\{v_{21}, v_{23}, v_{31}, v_{42}\}$ to v_{32} and v_{22}

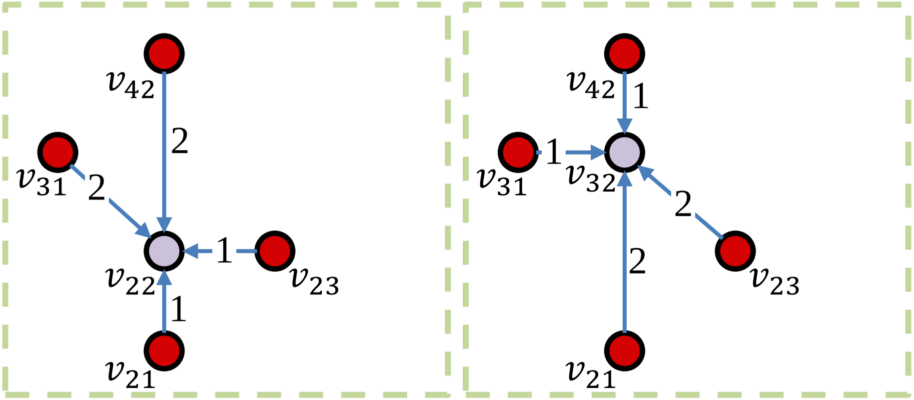


HMPO Graph-Based Insertion

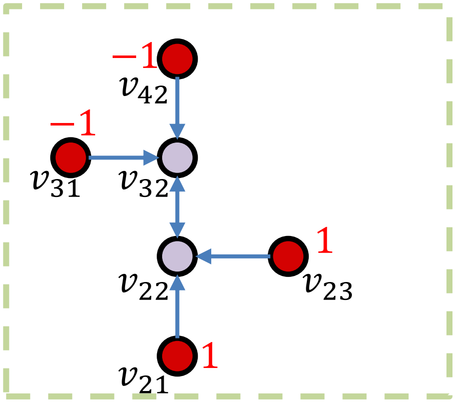
- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?



The distance from the 4 core vertices $\{v_{21}, v_{23}, v_{31}, v_{42}\}$ to v_{32} and v_{22}

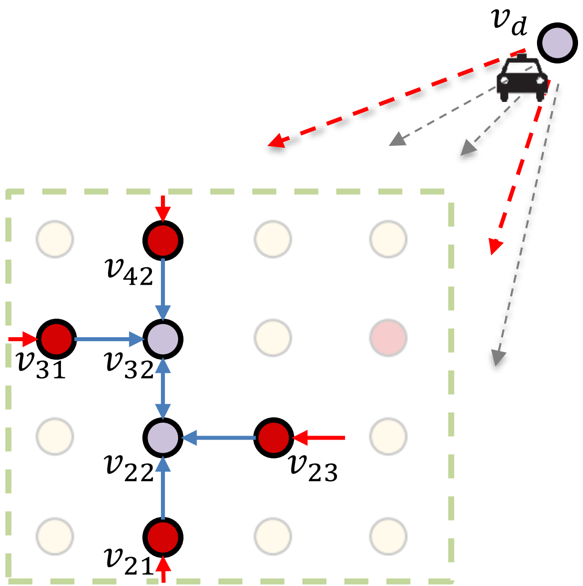


Calculate the difference for each core vertices, the time saving is bounded by 1 here.



HMPO Graph-Based Insertion

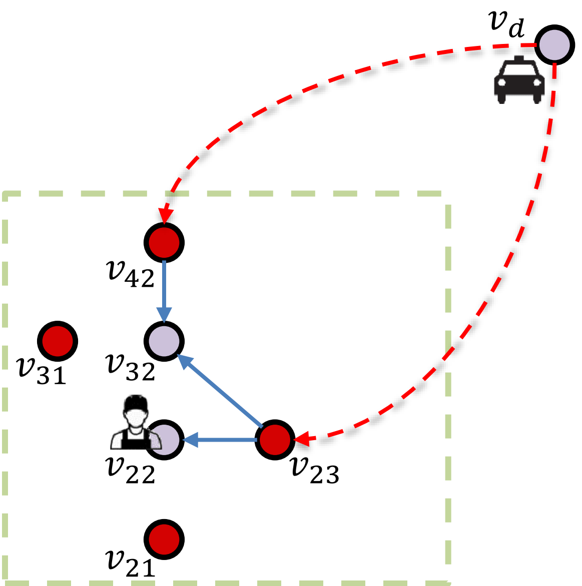
- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.
  - A driver want to serve a request at v_{22}, which has MP candidates $\{v_{22}, v_{23}\}$
  - If v_{22} exceeds the time limitation for 3 minutes, do we still need to test v_{32}?



If the shortest path from v_d to v_{32} is passed through v_{42}, the time saving is

$$(v_d \to v_{23} + v_{23} \to v_{22}) - (v_d \to v_{42} + v_{42} \to v_{32}) \leq (v_d \to v_{23} + v_{23} \to v_{22}) - (v_d \to v_{23} + v_{23} \to v_{32}) = v_{23} \to v_{22} - v_{23} \to v_{32} \leq 1$$

Detailed theory and proof are given in the paper



HMPO Graph-Based Insertion

- If inserting a candidate $v \in MC(u)$ fails to meet the time limitation, do the rest candidates help?
- **Bounds the time saving** of switching from v to any other vertices.

- Design SMDBoost algorithm.
- For each pair of driver and request, we test one vertex for insertion and prune the rest if the time limitation cannot meet with the bounded saving.

Algorithm 2: SMDBoost

Input: a driver w_i with route S_{w_i}, request r_j, MP candidate set MC, set maximum difference SMD, checker set Ch, dead vertices DV

Output: a route S^*_w for the driver w and updated DV

if *Driver's location* $l_i \in DV$ **then**
    Return S_{w_i} and DV without insertion
Generate arriving time $arv[\cdot]$ for S_{w_i}
Collect all sub-level vertices which have super-edges to vertices in $MC(s_j)$ into set Ne
The largest index to insert pick-up: $id^* = |S_{w_i}|$
foreach $v \in S_{w_i}$ **do**
    if $v \in Ne$ **then**
        Continue
    if $arv[v] + SP_h(v, Ch(s_j)) - SMD(Ch(s_j)) \geq tp_j$ **then**
        if $v = l_i$ **then**
            Add l_i to DV. Insertion fails and returns Null
        Record $id^* = idx(v) - 1$
        Break
Insert r_j with adapted insertion algorithm where insertion indexes of pick-ups larger than id^* are pruned.
return S^*_w, DV

Outline

- Background and Motivation
- The Meeting-Point-based Online Ridesharing Problem
- Framework Overview
- Methods
- Experimental Evaluation
- Summary

Experimental Settings

- Road Network
  - NYC ($|V|$=57,030, $|E|$=122,337)
- Real-World Dataset
  - Taxi Trips (yellow and green) in NYC (277,410 trip records)
- Synthetic Dataset
  - Generated according to the distribution of NYC (100k to 1m trip records)

Experimental Settings

- Compared parameters
  - e_r: the deadline coefficient.
  - a_w: the capacity of workers.
  - α: the weight for driving cost.
  - β: the weight for walking cost.
  - p_o: the ratio of penalty cost
  - $|W|$: number of workers
  - $|R|$: number of requests

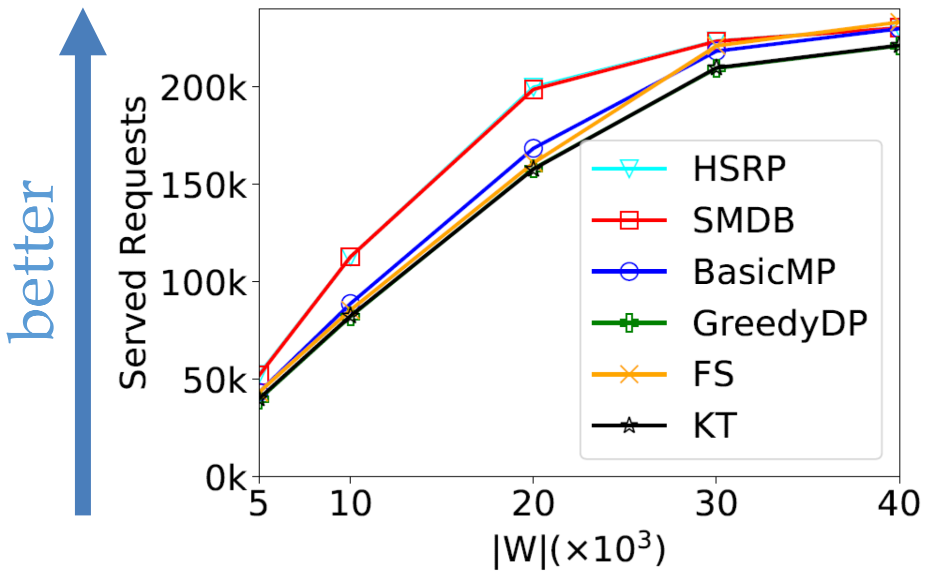
Parameters	Settings
Deadline Coefficient e_r	0.1, 0.2, **0.3**, 0.4, 0.5
Capacity a_w	2, **3**, 4, 7, 10
Driving Distance Weight α	**1**
Walking Distance Weight β	0.5, **1**, 1.5, 2
Penalty p_o	3, 5, 10, 15, **30**
Number of drivers $\|W\|$	5k, 10k, **20k**, 30k, 40k
Number of requests $\|R\|$	100k, 200k, 400k, 800k, 1000K

Experimental Settings

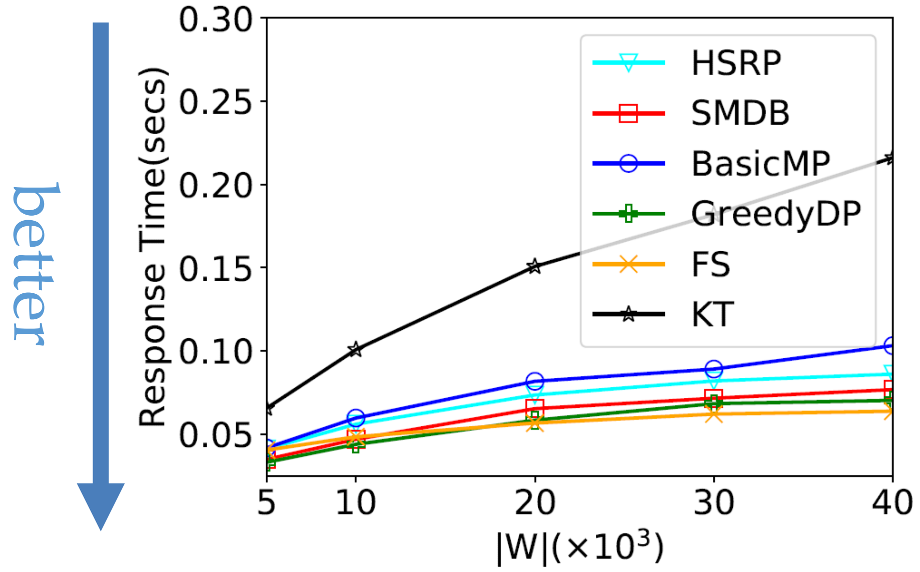
- Tested Algorithms
  - Traditional
    - **GreedyDP [1]**: the state-of-art route planning algorithm using insertion. No demand-related information is used.
    - **Kinetic Tree [2]**: it saves all the possible routes for the assigned request using a structure called Kinetic and inserts requests by traversing and updating the tree.
  - Meeting-Point-Based
    - **BasicMP**: It is an extension from GreedyDP by adapting MPs to solve the MORP problem.
    - **First Serve.** A variant of BasicMP, where each request is directly assigned to the first driver who can serve it.
    - **HSRP**. It uses the HMPO Graph to improve the effectiveness of BasicMP without pruning.

[1] Yongxin Tong, et al. 2018
[2] Shuo Ma, et al. 2013

Experimental Settings



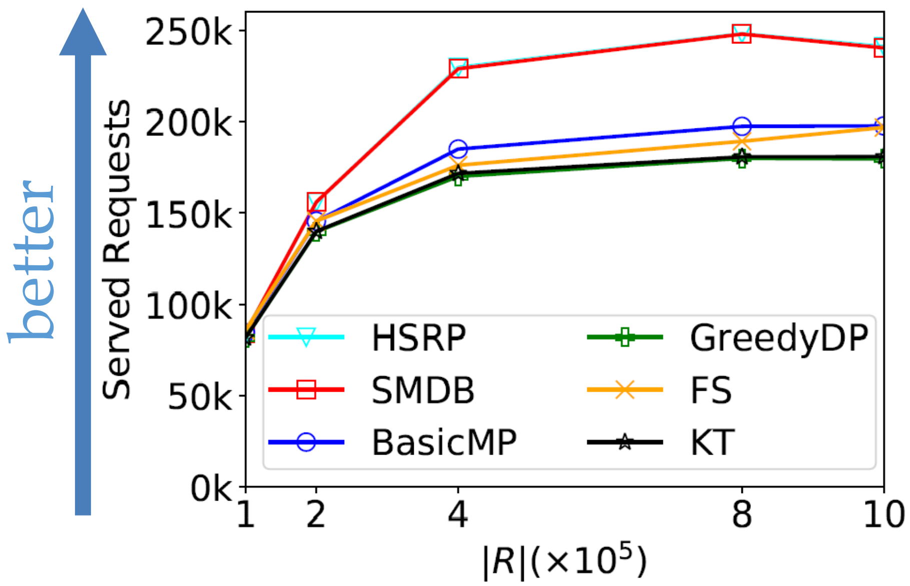
Serve ***21.4%*** to ***29.9%*** more requests



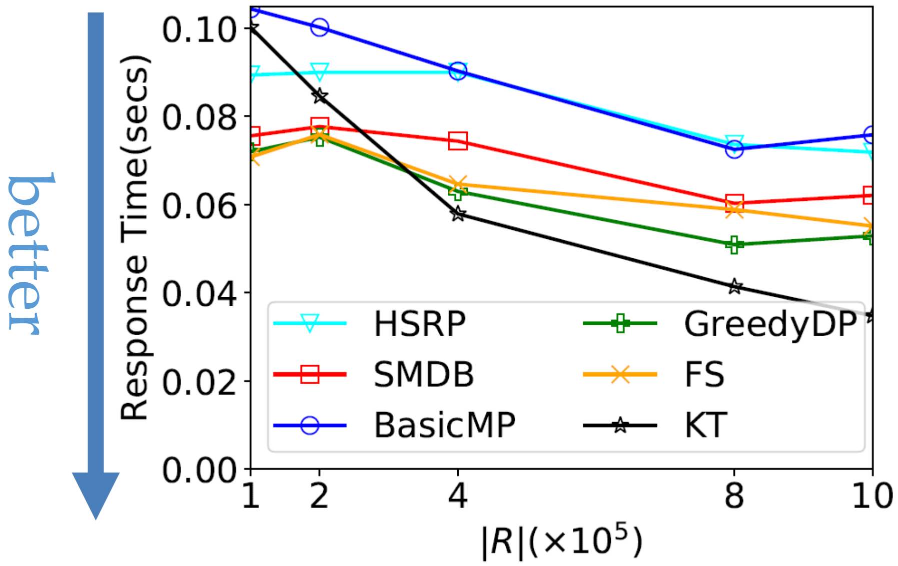
Response time < ***0.08s***

Performance of varying number of workers $|W|$

Experimental Settings



Serve *7.3%* to *28.4%* more requests



Response time < ***0.08s***

Performance of varying number of requests $|R|$

Outline

- Background and Motivation
- Framework Overview
- The Cache Replacement Problem
- Theoretical Guarantees
- Experimental Evaluation
- Summary

Summary

- We formulate the online route planning problem with MPs mathematically, namely MORP. We prove that it is NP-hard and has no algorithm with a constant competitive ratio.
- We propose an algorithm to select MP candidates for riders, which is based on a unified cost function considering the travel cost from additional walking.
- We propose a novel hierarchical structure of the road network, namely hierarchical meeting-point oriented (HMPO) graph, to fasten the solution for MORP.
- Based on the HMPO graph, we propose an effective and efficient insertor, namely SMDB, to handle the requests in MORP.

Thank You!

The code and datasets
https://github.com/dominatorX/open.