GeoPrune: Efficiently Matching Trips in Ride-sharing Through Geometric Properties

Yixin Xu, Jianzhong Qi, Renata Borovica-Gajic, Lars Kulik
yixinx3@student.unimelb.edu.au,{jianzhong.qi,renata.borovica,lkulik}@unimelb.edu.au
The University of Melbourne, Melbourne, Australia

GeoPrune Algorithms

Jae Won Lee

October 17, 2020

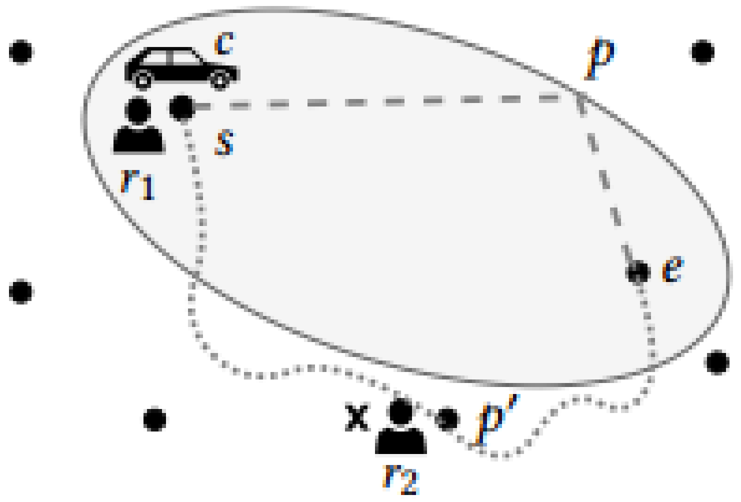


Figure 1: Illustration of our key idea

Table 1: Frequently Used Symbols

Notation	Description
$G = \langle N, E \rangle$	a road network with vertex (edge) set N (E)
$t(n_i, n_j)$	the estimated shortest travel time between vertices n_i and n_j
$R = \{r_i\}$	a set of trip requests
$C = \{c_j\}$	a set of vehicles
$r_i = \langle t, s, e, w, \epsilon, \eta \rangle$	a trip request issued at time t with source s, destination e, maximum waiting time w, maximum detour ratio ϵ and η passengers
$r_i.lp, r_i.ld$	the latest pickup and drop-off times of r_i
$r_i.wc, r_i.rd$	the waiting circle and the detour ellipse of r_i
$c_j = \langle l, S, u, v \rangle$	a vehicle at l with planned trip schedule S, capacity u and traveling speed v
(p^{k-1}, p^k)	the segment between p^{k-1} and p^k
$vd[k]$	the detour ellipse of (p^{k-1}, p^k)

Example 2.1. *Assume two trip requests* $r_1 = \langle 9{:}00\,am, s_1, e_1, 5\,min, 0.2, 1\rangle$ *and* $r_2 = \langle 9{:}07\,am, s_2, e_2, 5\,min, 0.2, 1\rangle$ *in Figure 2. The shortest travel times from* s_1 *to* e_1 *and from* s_2 *to* e_2, *i.e.,* $t(s_1, e_1)$ *and* $t(s_2, e_2)$, *are both 15 min. Then, the time constraints of* r_1 *and* r_2 *are:* $r_1.lp$=9:00 am+5 min=9:05 am, $r_2.lp$=9:07 am+5 min=9:12 am, $r_1.ld$=9:05 am+15 min×1.2=9:23 am, $r_2.ld$=9:12 am+15 min×1.2=9:30 am.

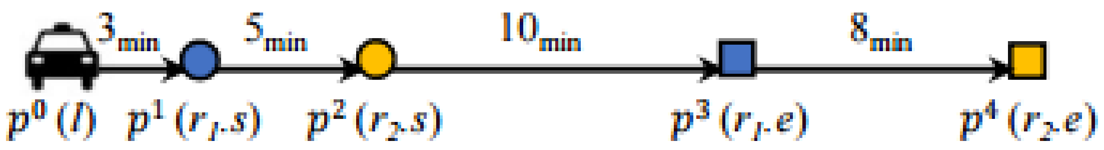
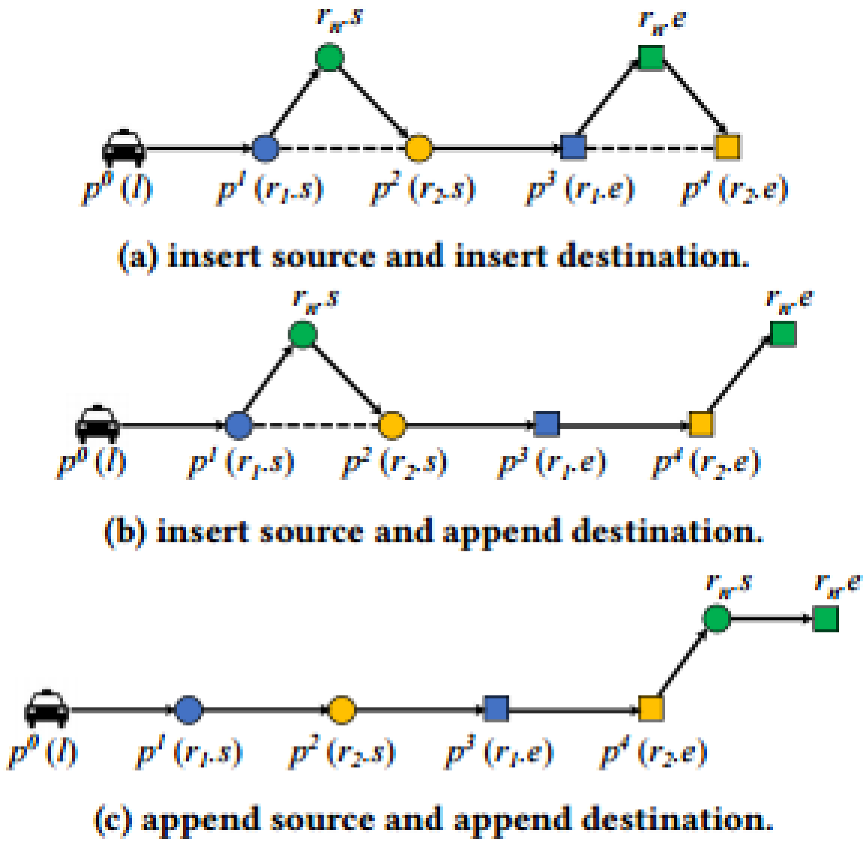


Figure 2: A vehicle schedule example at 9:00 am.

Table 2: Recorded Data for the Trip Schedule in Figure 2.

p^k	$arr[k]$	$ddl[k]$	$ddl[k] - arr[k]$	$slk[k]$
p^1	9:03 am	9:05 am	2 min	2 min
p^2	9:08 am	9:12 am	4 min	4 min
p^3	9:18 am	9:23 am	5 min	4 min
p^4	9:26 am	9:30 am	4 min	4 min

Pruning Rules



(a) insert source and insert destination.

(b) insert source and append destination.

(c) append source and append destination.

Figure 5: Cases to add a new trip request to a trip schedule

R-tree based pruning

- T_{seg} : One R-tree store the detour ellipses of all segments for all vehicle trip schedules
- T_{end}: The other R-tree stores the location of the ending stops of all non-empty vehicles

GeoPrune runs four queries.

- 1) $Q_1 = T_{seg}.pointQuery(r_n.s)$
  - Returns all segments whose detour ellipses cover $r_n.s$
- 2) $Q_2 = T_{seg}.pointQuery(r_n.e)$
  - Returns all segments whose detour ellipses cover $r_n.e$
- 3) $Q_3 = T_{end}.rangeQuery(r_n.wc)$
  - Returns all ending stops covered by $r_n.wc$
- 4) $Q_4 = T_{end}.rangeQuery(r_n.rd)$
  - Returns all ending stops covered by $r_n.rd$

Purning process

Algorithm 1: Prune non-empty vehicles

Input: A new trip request r_n

Output: a set of possible vehicles to serve r_n

// Pruning stage

1 $r_n.wc \leftarrow$ waiting circle of r_n; $r_n.rd \leftarrow$ detour ellipse of r_n

2 $Q_1 \leftarrow T_{seg}.pointQuery(r_n.s)$

3 $Q_2 \leftarrow T_{seg}.pointQuery(r_n.e)$

4 $Q_3 \leftarrow T_{end}.rangeQuery(r_n.wc)$

5 $Q_4 \leftarrow T_{end}.rangeQuery(r_n.rd)$

6 **for** *an element in* $Q_1, Q_2, Q_3,$ *and* Q_4 **do**

7   **if** *the time or capacity constraint is violated* **then**

8   remove the element

9 Record the corresponding vehicles of the elements in Q_1, Q_2, Q_3, Q_4 in O_1, O_2, O_3, O_4.

10 $F, F_1, F_2, F_3 \leftarrow \emptyset$

11 $F_1 \leftarrow O_1 \cap O_2$ // insert-insert case

12 $F_2 \leftarrow O_1 \cap O_4$ // insert-append case

13 $F_3 \leftarrow O_3$ // append-append case

14 $F \leftarrow F_1 \cup F_2 \cup F_3$

15 **return** F

1: Compute waiting cycle and detour ellipse of the new trip request.

2: Returns all segments whose detour ellipses cover $r_n.s$.

3: Returns all segments whose detour ellipses cover $r_n.e$.

4: Returns all ending stops covered by $r_n.wc$.

5: Returns all ending stops covered by $r_n.rd$.

6 – 8: Each returned segments and ending stops are checked against the capacity and time constraints.

10-15: Vehicles of the remaining segments and ending stops are candidates.

If a new trip request is matched with a vehicle

Algorithm 2: Update index - match

Input: A new trip request r_n and the matched vehicle c_i

```
1  if c_i empty then
2      T_ev.remove(c_i)
3  else
4      for a segment in the trip schedule of c_i do
5          remove the ellipse of the segment from T_seg
6      T_end.remove(ending stop of c_i)
7  add r_n.s and r_n.e to the trip schedule of c_i
8  for a segment in the trip schedule of c_i do
9      compute the detour ellipse of the segment
10     insert the ellipse of the segment into T_seg
11 T_end.insert(the end stop of c_i)
```

1-2: If c_i is empty, it is available to be occupied. Remove from the T_{ev} that stores empty vehicles for fast nearest empty vehicle computation.

4-6: If not, remove the segments and the ending stops from T_{seg} and T_{end}.

7: Add the new trip request.

8-10: Recompute the detour ellipses based on the updated schedule.

11: New ending stop is inserted into T_{end}.

Update the data structures when the vehicles move

Algorithm 3: Update index - move

Input: A moving vehicle c_i

1 $P \leftarrow$ obsolete segments of c_i

2 **for** $p \in P$ **do**

3 $\quad T_{seg}.remove(p)$

4 **if** c_i *reaches the ending stop* **then**

5 $\quad T_{end}.remove($ending stop of $c_i)$

6 $\quad T_{ev}.insert(c_i)$

1-3: At every time point, check if a vehicle has reached a stop in its trip schedule. If yes, the segments before the reached stop become obsolete and their detour ellipses are removed from T_{seq}.

4-6: When the vehicle reaches its ending stop, the vehicle becomes empty. Remove it from T_{end} and insert it into T_{ev}.

Experimental set-up

Table 3: Datasets

Name	# vertices	# edges	# requests
NYC	166,296	405,460	448,128
CD	254,423	467,773	259,343

Table 4: Experiment parameters

Parameters	Values	Default
Number of vehicles	2^{10} to 2^{17}	2^{13}
Waiting time (min)	2, 4, 6, 8, 10	4
Detour ratio	0.2, 0.4, 0.6, 0.8	0.2
Number of requests	20k to 100k	60k
Frequency of requests (# requests/second)	1 to 10	refer to table 3
Transforming speed (km/h)	20 to 140	48

- Datasets: OpenStreetMap, New York City (April 09, 2016) and Chengdu (Nov 18, 2016)
- Transform the coordinates to *Universal Transverse Mercator* (UTM) coordinates to support pruning based on Euclidean distance
- Each request consists of *a source, a destination, and an issue time.*
- Assume the number of passengers to be *one* per request.
- Use a constant travel speed for all edges 48km/h.

Baselines

- GreedyGrids
- Tshare
- Xhare

- *Need to study them further.

Metrics

- Number of remaining vehicles: number of remaining candidates after pruning
  - Note that GeoPrune prunes empty vehicles and non-empty vehicles separately with different criteria.
- Match time: total running time of the matching process, including both pruning and selection time
- Overall update time: overall match update and move update time
- Memory consumption