Ridesharing: Simulator, Benchmark, and Evaluation

2019年10月19日 17:22

Abstract
Ridesharing: gather several customers into one vehicle and save cost.
Work: simulator and benchmark to evaluate existing algorithms.

Introduction
DARP (offline) and RSP (online) situation.

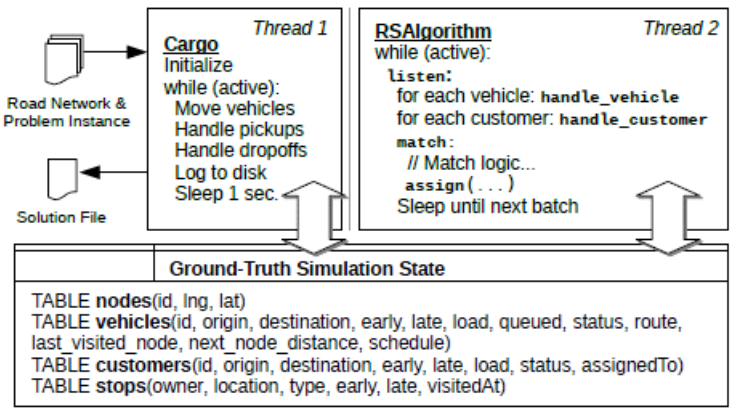
Background
Define: road network, vehicles and customers, time window, capacity, Euclidean distance, shortest-path cost, precedence.
Is it NP-hard? Why without loss of the generality?
Online assignment, optimization goals, vehicle schedule:

$$C(S,k) = c_{k_o,u_1} + \sum_{i}^{s-1} c_{u_i,u_{i+1}} + c_{u_s,k_d}$$

Related: DARP benchmark, comparison study, experimental platform.

Ridesharing algorithms
Branch-and-Bound (BB), Nearest Neighbor (NN), Greedy Insertion (GR), Greedy Kinetic Tree (KT), Bilateral Arrangement (BA), Simulated Annealing (SA), GRASP (GP), Trip-Vehicle Grouping (TG).
Offline: BB; others are online.
Search: NN, GR, KT, BA; Join: SA, GP, TG.

Cargo architecture
Cargo: multi-threaded simulation system. Simplify the implementation of ridesharing algorithms.
Main component: problem instance and road network.
Simulation progress in disk.
Output a small uniform text file.



Vehicle motion: SQL notes location of vehicles, speed is constant.
Question: each change leads to one SQL statement? Too slow?
Seems not, one SQL bulk-update all vehicles, so may be a buffer.
RS algorithm: listen, handle customer, handle vehicle, match.
Assign mode: synchronization (match latency), strict mode / non-strict mode (better).

Spatial indexes: G-tree, grid index.

Benchmark

No edges longer than 100 meters.
Problem instances: data from 6:00 - 6:30 pm, 9.70 per sec in Beijing, and 4.96, 2.80 for Chengdu and Manhattan.
    Scale factors: 0.5x, 2.0x, and 4.0x. Put extra trips into 30 minutes.
        But where the extra data from?
Time window of customer: plus a delay of 6, 12, or 24 minutes.

Experiment

Grid dimensions, LRU cache size, simulation duration, road network, vehicle speed, vehicle capacity, number of vehicles, cust scale factors, delay tolerance.
Runtime: static mode (30 sec) and dynamic mode (online).
Timeout: 30 sec timeout.
Metrics: average customer handling time, service rate, distance saving ($1-z/z_0$):

$$z = \sum_{k \in M} (\text{distance traveled by } k) + \sum_{r \in N_{ko}} c_{r_o r_d},$$

$$\text{and } z_0 = \sum_{r \in N} c_{r_o r_d}.$$

Static runtime evaluations: search / join algorithms. Filtering by grid size and Euclidean distance, insertion by schedule lengths (x^2).
    Search algorithms: NN, GR, KT, BA, BA+.
    Join algorithms: SA100, SA50, GP16, GP4, TG.
    Effects of number of vehicles, vehicle capacity, time window.
    Conclusions in takeaway part.
Real-time dynamic evaluations: NN, GR, KT, BA+, SA50, GP4, TG.
    Mostly could not achieve near static mode performance.
    Competitive ratios analysis: cost of online algorithm / best case cost.

Conclusion

Practitioners:
    Maximizing matches: NN, SA.
    Maximizing distance savings: KT.
    3-capacity vehicles is best.
Algorithm design:
    Indexing vehicles to improve performance, or cheaper heuristics (instead of schedule cost).
    Aggressively rather than randomly.
    Don't list customer-vehicle combinations (TG) until computational burden overcome.