Exploring real-time dynamic trip-vehicle assignment for ride-sharing simulation

Andrew Park
EE227BT Convex Optimization
University of California, Berkeley
jungho.park@berkeley.edu

Abstract

Shared autonomous vehicles (SAVs) are among the most important future mobility use cases due to their potential to reduce overall vehicle miles traveled (VMT), alleviate metropolitan congestion, and improve the efficiency of road networks. This report explores strategies for assigning vehicles to travel requests within a road network. A computer program was implemented to test random trip requests, and its real-time performance was evaluated. Finally, the report discusses how these strategies can be applied in a real-time simulation, paving the way for future work.

1 Introduction

1.1 Problem Statement

In a real-world scenario, ride requests are generated spontaneously over a large network. In the case of ride-sharing, which accommodates multiple passengers with different trip requests simultaneously, it is essential to match passengers with similar trips and incorporate those trips with passengers already on board. This enables vehicle agents to add passengers whose trips align with existing routes, while satisfying various constraints associated with individual trips. In most studies, these constraints are primarily imposed in terms of time [Tian et al., 2024].

To leverage this characteristic of ride-sharing in a simulation environment, it is necessary to generate trip scenarios based on existing trip demands. Trip demands are often inferred from prior knowledge. However, serving these demands depends on the policies implemented by stakeholders. Such policies possess degrees of freedom that can be adjusted to achieve specific objectives.

This report serves as a foundation for building a simulation environment where planning algorithms can be validated. The planning method utilized in this work focuses on optimizing the time efficiency of ride-sharing services. Alternative methods include optimization for energy efficiency, congestion mitigation, and revenue maximization. To minimize cost in terms of time, it is necessary to define the state of the network. This work builds upon Alonso-Mora et al. [2017], which introduced algorithms for dynamic vehicle-trip assignment and provided mathematical proof for convergence to a solution.

1.2 Definitions

To formulate the problem, the state of the network is defined as consisting of requests and vehicles. For the implementation of this algorithm, a graph representation of the road network was selected, using OpenStreetMap as the source for road specifications [OpenStreetMap contributors, 2024]. The road network is represented as a graph $G = (v, e)$, where vertices represent intersections and edges represent road segments. Vehicle and request agents are defined on this network. For simplification, these agents are assumed to be located at intersections for matching purposes. In a real-world scenario,

each new request would correspond to a distinct node that can be located anywhere on the road network. First, a request is defined as request $r = \{o_r, d_r, t_r^r, t_r^{pl}, t_r^p, t_r^d, t_r^*\}$, where

- o_r : passenger pick-up point
- d_r : passenger point
- t_r^r : time the request is made
- t_r^{pl} : latest acceptable pick-up time ($t_r^{pl} = t_r^r + \Omega$)
- Ω : maximum waiting time
- t_r^p : pick-up time
- t_r^d : estimated arrival time
- t_r^* : fastest drop-off time

Note that the fastest drop off time is calculated by having the current position of the vehicle at the origin of the request, hence the request is instantaneously picked up and proceeds to be dropped off. A ride-sharing vehicle can also be defined as a tuple $v = \{q_v, t_v, P_v\}$, where

- q_v : vehicle position
- t_v : drop-off point
- P_v : Current passengers

Each vehicle can be assigned a set of requests that it can serve in each time step. The set of requests can be expressed as an array of requests, which need to be constructed based on the time constraints. A trip can be expressed as $\mathrm{T} = \{r_1, r_2, ..., r_{n_T}\}$. For this study, the size of the fleet was fixed, m, and each vehicle had a capacity limit, v. The maximum travel delay is defined as the difference between the earliest possible drop-off time and the actual expected drop-off time after assignment.

- m : fixed fleet size
- v : vehicle capacity
- Δ : max travel delay (from earliest drop-off time)

2 Formulation of Optimization Problem

2.1 Initial Problem Formulation

The minimization problem was formulated as follows.

$$\begin{aligned}
\text{minimize} \quad & C(\Sigma) = \sum_{v \in V} \sum_{p \in P_v} \delta_p + \sum_{r \in R_{ok}} \delta_r + \sum_{r \in R_{ko}} c_{ko} \\
\text{subject to} \quad & t_r^p \leq t_r^{pl} \leq t_r^r + \Omega \quad \text{(maximum waiting time)} \\
& t_r^d \leq t_r^* + \Delta \quad \text{(maximum travel delay)} \\
& n_v^{pass} \leq v \quad \text{(maximum passenger limit)}
\end{aligned}$$

This problem, unfortunately, is an NP hard problem, since comparing these time variables requires factorial comparison among all possible trip configurations.

2.2 Request-Vehicle (RV) Graph

To resolve this, a sub-graph of the original network is first created, pairing each vehicle with all of the requests it is able to serve. This process also accounts for the current passenger occupancy of the vehicles. To construct the request-vehicle (RV) graph, each request-request pair is evaluated by solving Dijkstra's shortest path algorithm, assuming the trip begins at the origin of one of the requests. If the pair satisfies the maximum waiting time delay requirement for both requests, the two requests, r_1 and r_2, are connected with an edge, where the cost is the sum of the delays associated with each

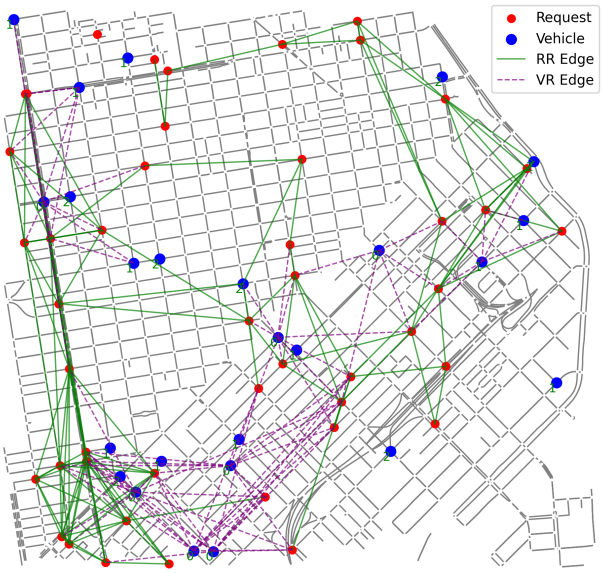

Figure 1: RV Graph

edge. Similarly, a trip and a request are paired if they can be combined under the same conditions. An implementation result of the RV graph for requests on the east side of downtown San Francisco is shown in Figure 1.

2.3 Request-Trip-Vehicle (RTV) Graph

Secondly, from the request-vehicle sub-graph, another graph is formed, consisting of trips (or groups of requests) that can be combined and served by a connected vehicle. Each edge connection from a vehicle to a trip combination represents a possible assignment. This abstraction of the RV sub-graph allows the search algorithm to be implemented efficiently across the network. The algorithm for RTV graph formulation is shown in Figure 2. The computation time for RTV graph formation is improved by preconditioning the process, such as verifying that each request's pick-up and drop-off sequence is physically feasible. The ultimate goal is to achieve trip-vehicle pairing, $e(T, v)$, for all valid trips.

Algorithm 1 Generation of RTV-graph

$\mathcal{T} = \emptyset$
for each vehicle $v \in \mathcal{V}$ **do**
  $\mathcal{T}_k = \emptyset \ \forall k \in \{1, \ldots, \nu\}$
  [Add trips of size one]
  for $e(r, v)$ edge of RV-graph **do**
    $\mathcal{T}_1 \leftarrow T = \{r\}$; Add $e(r, T)$ and $e(T, v)$
  [Add trips of size two]
  for all $\{r_1\}, \{r_2\} \in \mathcal{T}_1$ and $e(r_1, r_2) \in$ RV-graph **do**
    if $travel(v, \{r_1, r_2\}) =$ valid **then**
      $\mathcal{T}_2 \leftarrow T = \{r_1, r_2\}$; Add $e(r_i, T)$ and $e(T, v)$
  [Add trips of size k]
  for $k \in \{3, \ldots, \nu\}$ **do**
    for all $T_1, T_2 \in \mathcal{T}_{k-1}$ with $|T_1 \cup T_2| = k$ **do**
      Denote $T_1 \cup T_2 = \{r_1, ..., r_k\}$
      if $\forall i \in \{1, ..., k\}$, $\{r_1, ..., r_k\} \setminus r_i \in \mathcal{T}_{k-1}$ **then**
        if $travel(v, T_1 \cup T_2) =$ valid **then**
          $\mathcal{T}_k \leftarrow T = T_1 \cup T_2$
          Add $e(r_i, T)$, $\forall r_i \in T$, and $e(T, v)$
  $\mathcal{T} \leftarrow \bigcup_{i \in \{1, ..., \nu\}} \mathcal{T}_i$

Figure 2: RTV Algorithm

2.4 Optimal Assignment

Thirdly, the optimal assignment is performed based on the possible vehicle-trip combinations. The initial solution is obtained using a greedy assignment approach. In the greedy search algorithm, the time efficiency per passenger is computed for each trip-vehicle edge, and the edges with maximum values are prioritized and assigned using the pop operation of a max-priority queue implemented with a heap structure. The time complexity of the initial search is relatively low at $O(log(n))$.

Following this, an integer linear program (ILP) is formulated to refine the initial assignment and bring it closer to optimality. To solve this, the branch-and-bound method with an optimality gap specification is applied. Given binary variables, $\epsilon_{i,j} \in \{0, 1\}$ for each edge $e(T_i, v_j)$ between trip $T_i \in T,\ v_j \in V$ in the RTV graph, where,

- $\epsilon_{i,j} = \begin{cases} 1 & \text{if } v_j \text{ assigned to } T_i \\ 0 & \text{o.w.} \end{cases}$
- $x_k \in \{0, 1\}$ for each request $r_k \in R$
- $x_k = \begin{cases} 1 & \text{if } r_k \text{ cannot be served by any vehicle} \\ 0 & \text{o.w.} \end{cases}$
- $X = \{\epsilon_{i,j},\ x_k; \forall e(T_i, v_j) \text{ and } \forall r_k \in R\}$

An ILP can be formulated based on the above integer variables to minimize the total cost of the assignment. The cost of a trip is defined as $c_{i,j} = \sum_{r \in T_i} (t_r^d - t_r^*)$, representing the sum of delays for a given trip-vehicle pair. Additionally, a penalty for unassigned trips is denoted as c_{ko}. Indices are defined as follows:

- I_j^V indexes i for edge $e(T_i, v_j)$ exists in the RTV graph
- I_k^R indexes i for edge $e(r_k, T_i)$ exists in the RTV graph
- I_i^T indexes j for edge $e(T_i, v_j)$ exists in the RTV graph

The minimization problem then becomes:

$$\text{argmin}_X\, C(X) := \sum_{i,j\ \in \varepsilon_{TV}} c_{i,j} \cdot \epsilon_{i,j} + \sum_{k \in \{0,\ldots,n\}} c_{ko} X_k$$

$$\text{subject to} \sum_{i \in I_j^V} \epsilon_{i,j} \leq 1 \quad \forall v_j \in V$$

$$\sum_{i \in I_k^R} \sum_{j \in I_i^T} \epsilon_{i,j}\ + x_k = 1 \quad \forall r_k \in R$$

The ILP is solved efficiently because trip validity is pre-computed during RTV graph formation. It has been verified that the solution converges to an optimal result with a 0.1% optimality gap in less than 2 seconds. In this iteration, an improvement of approximately 7% was achieved compared to the greedy assignment in terms of time efficiency. The result of the ILP convergence is shown in Figure 3.

Nodes		Current Node			Objective Bounds			Work	
Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
0	0	11707.8136	0	6	12584.0838	11707.8136	6.96%	-	0s
H 0	0				11766.094833	11707.8136	0.50%	-	0s
H 0	0				11760.210405	11707.8136	0.45%	-	0s
0	0	cutoff	0		11760.2104	11760.2104	0.00%	-	0s

Figure 3: Optimization result

2.5 Demand Re-balancing

Finally, unserved demands are addressed. Due to an imbalance between available vehicles and the spatial distribution of excess trip requests, some requests may remain unfulfilled in a single iteration

of the optimal assignment. These requests are subsequently served by idle vehicles that were not previously assigned in the trip-vehicle matching process, with penalties imposed on unserved vehicles. For this adjustment, additional variables and penalty terms are defined.

- $\mathcal{V}_{\text{idle}}$: the idle (empty, stopped, and unassigned) vehicles
- $\mathcal{R}_{k_o}$: the unassigned requests
- $\tau_{v,r}$: the shortest travel time for vehicle $v \in \mathcal{V}_{\text{idle}}$ to pick request $r \in \mathcal{R}_{k_o}$

The goal is to prioritize the assignment of unserved requests to vehicles that can reach these requests as quickly as possible. This problem can be formulated as a linear program that minimizes the total travel time for the fleet of vehicles to serve the unserved requests. The linear program is formulated using the variable $y_{v,r}$, hich indicates the assignment of a vehicle to a request. This is a relaxed version of the ILP for the assignment problem. The original ILP employs a decision variable $y_{v,r} \in \{0, 1\}$. However, this decision variable can be relaxed to a continuous variable under the assumption that there are no ties in the shortest travel time to an unserved request. Therefore, a relaxed decision variable is used, where $y = \bigcup_{v \in \mathcal{V}_{\text{idle}}, r \in \mathcal{R}_{k_o}} y_{v,r}$, taking values between 0 and 1. The resulting linear program is as follows:

$$
\begin{aligned}
&\arg\min_{y} \sum_{v \in \mathcal{V}_{\text{idle}}} \sum_{r \in \mathcal{R}_{k_o}} \tau_{v,r} \cdot y_{v,r} \\
&\text{subject to} \sum_{v \in \mathcal{V}_{\text{idle}}} y_{v,r} = \min(|\mathcal{V}_{\text{idle}}|, |\mathcal{R}_{k_o}|), \quad \forall r \in \mathcal{R}_{k_o} \\
&\qquad\qquad 0 \leq y_{v,r} \leq 1, \quad \forall y_{v,r} \in y
\end{aligned}
$$

This linear program can be solved over the remaining RV graph of unserved requests, which is created by removing the requests that have already been served. In the future, this operation can be integrated into the trip assignment step to improve execution efficiency.

3 Result and Discussion of Future Works

3.1 Python Implementation Result

A Python program was developed using map data from OpenStreetMap and demand data with randomized origin-destination pairs. The data was generated to be randomly distributed across the road network at a given time. The software implementation demonstrates that optimal assignments can be computed at specified time steps to continually optimize vehicle-trip assignments over the network.

The computation can greatly benefit from parallelization, as the majority of the operations follow a single-instruction multiple-data (SIMD) structure. However, scalability to larger networks is anticipated to be a challenge, as optimization over larger networks without any physical constraints requires combinatorial time-delay comparisons. During implementation, it was discovered that constraining the comparison operation—by hard-limiting the radius of each vehicle-request-request combination—significantly improves computation time and enhances the scalability of the algorithm.

The pseudocode illustrating the workflow of the algorithm is shown below.

```
function solve_batch_assignment(requests, vehicles, constraints):
    # Step 1: Build the RV-Graph
    rv_graph = build_rv_graph(requests, vehicles, constraints)

    # Step 2: Generate the RTV-Graph
    rtv_graph = build_rtv_graph(rv_graph)

    # Step 3: Initialize variables for ILP
    initialize_ilp_variables(rtv_graph)
```

```
    # Step 4: Define objective function
    objective = minimize_cost(rtv_graph)

    # Step 5: Define constraints
    add_constraints(rtv_graph, constraints)

    # Step 6: Solve the ILP
    optimal_assignment = solve_ilp(objective, constraints)

    return optimal_assignment

function build_rv_graph(requests, vehicles, constraints):
    graph = {}
    for request in requests:
        for vehicle in vehicles:
            if is_feasible(request, vehicle, constraints):
                graph.add_edge(request, vehicle)
    return graph

function build_rtv_graph(rv_graph):
    graph = {}
    for trip in feasible_combinations(rv_graph.requests):
        for vehicle in rv_graph.vehicles:
            if can_vehicle_serve_trip(vehicle, trip):
                graph.add_edge(trip, vehicle)
    return graph

function is_feasible(request, vehicle, constraints):
    return (
        vehicle.capacity >= request.passengers
        and travel_time(vehicle.position, request.origin) <= constraints.max_waiting_time
    )
```

Figure 4 shows the greedy assignment and the improved assignment side by side. This optimization has improved overall time efficiency by 7%. Continuous trip optimization over an entire day's operation is expected to significantly reduce energy consumption.

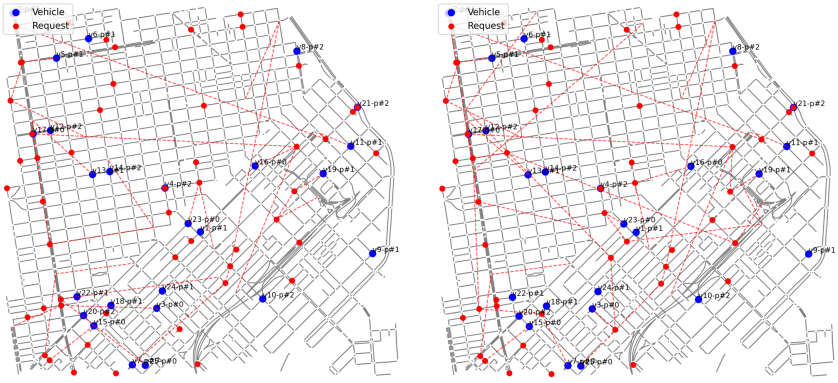

Figure 4: Greedy Assignment (Left), ILP Result (Right)

3.2 Discussion of Future Works

In the future, other planning and control algorithms are intended to be verified within this simulation environment. The current environment runs on a real-time traffic simulator accelerated by GPU. Further work is required to integrate it with a Python framework, as some AI algorithms trained in Python environments still need to be validated in CUDA or other GPU programming languages [Jiang et al., 2024].

One critical area requiring further exploration is the integration of energy dynamics. Proper validation of algorithms that optimize the energy efficiency of the fleet requires incorporating basic driving dynamics, including various resistances. In another project, integrating energy considerations into a GPU simulation environment has been studied. The energy dynamics of a single vehicle for a specific trip design are shown in Figure 5. It primarily illustrates the power consumption of two vehicles driving in series, with one following the other. The two vehicles follow intelligent driver model (IDM) addressed in Albeaik et al. [2022].

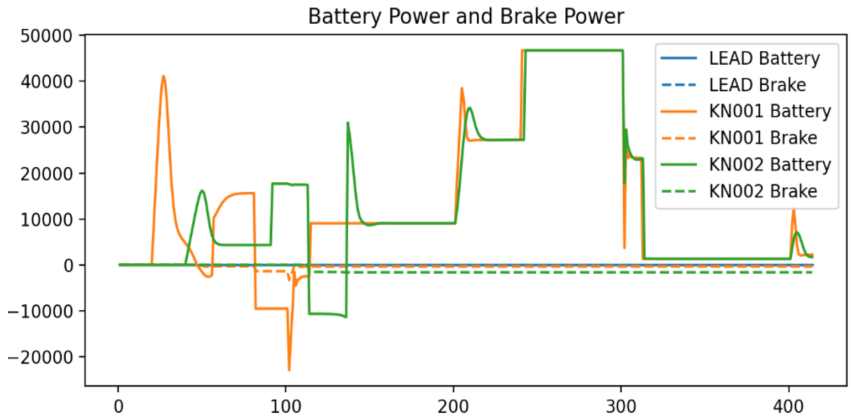

Figure 5: Energy dynamics over a driving cycle

Several optimization strategies incorporating energy considerations have been proposed. The equation below presents a basic formulation of energy consumption, which can be converted into a convex optimization problem if the expected energy profile is represented by a convex function.

- Objective: Minimize total energy consumption:

$$\text{Minimize} \quad \sum_{i=1}^{N} \int_0^T P_i(v_i(t))\, dt$$

  where: $P_i(v_i(t))$: Power consumption of vehicle i as a function of speed $v_i(t)$

- Constraints:
  1. Speed limits: $0 \leq v_i(t) \leq v_{\max}, \quad \forall i, t$
  2. Travel time: $\int_0^T v_i(t)\, dt = d_i, \quad \forall i$ where d_i is the trip distance.

Finally, the vehicle miles traveled (VMT) for fleet vehicles with optimal planning will be analyzed to examine its implications for vehicle development. Vehicle mileage accumulation is a key indicator of how vehicles are utilized over their life cycle. It is a crucial development factor for designing robust and reliable products. Given that the use case of shared autonomous vehicles (SAVs) is particularly demanding, it provides valuable insights into future usage scenarios for SAVs optimized in a similar manner.

References

Saleh Albeaik, Alexandre Bayen, Maria Teresa Chiri, Xiaoqian Gong, Amaury Hayat, Nicolas Kardous, Alexander Keimer, Sean T. McQuade, Benedetto Piccoli, and Yiling You. Limitations and improvements of the intelligent driver model (idm). *SIAM Journal on Applied Dynamical Systems*, 21(3):1862–1892, 2022. doi: 10.1137/21M1406477.

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017. doi: 10.1073/pnas.1611675114. URL `https://www.pnas.org/doi/abs/10.1073/pnas.1611675114`.

Xuan Jiang, Raja Sengupta, James Demmel, and Samuel Williams. Large scale multi-gpu based parallel traffic simulation for accelerated traffic assignment and propagation. *Transportation*

Research Part C, 169:104873, 2024. doi: 10.1016/j.trc.2024.104873. URL https://github.com/Xuan-1998/LPSim.

OpenStreetMap contributors. OpenStreetMap: The free wiki world map, 2024. URL https://www.openstreetmap.org. Accessed: 2024-06-17.

Hangqi Tian, Ning Wang, and Yuntao Guo. Multi-stage dynamic scheduling optimization for autonomous mobility-on-demand systems. *2024 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1562–1567, 2024. doi: 10.1109/ICMA61710.2024.10632943.