Distributed Fair Assignment and Rebalancing for Mobility-on-Demand Systems via an Auction-based Method

Kaier Liang and Cristian-Ioan Vasile

Abstract— In this paper, we consider fair assignment of complex requests for Mobility-On-Demand systems. We model the transportation requests as temporal logic formulas that must be satisfied by a fleet of vehicles. We require that the assignment of requests to vehicles is performed in a distributed manner based only on communication between vehicles while ensuring fair allocation. Our approach to the vehicle-request assignment problem is based on a distributed auction scheme with no centralized bidding that leverages utility history correction of bids to improve fairness. Complementarily, we propose a rebalancing scheme that employs rerouting vehicles to more rewarding areas to increase the potential future utility and ensure a fairer utility distribution. We adopt the max-min and deviation of utility as the two criteria for fairness. We demonstrate the methods in the mid-Manhattan map with a large number of requests generated in different probability settings. We show that we increase the fairness between vehicles based on the fairness criteria without degenerating the servicing quality.

I. INTRODUCTION

Mobility-On-Demand systems have been recognized as a promising solution to reduce travel costs, traffic congestion, and emissions [1], [2]. Passengers can specify their demands and share vehicles with others, and it can greatly improve transportation for people and goods. However, most research in this area has been focused on the passenger's perspective, and less attention has been paid to the problem from the driver's perspective. The assignment objectives are usually centered on minimizing the travel cost, which may not be in accord with the driver's preferences [3], [4]. Moreover, within the vehicle fleet, due to competition, unfairness may arise due to the uneven distribution of requests, resulting in some vehicles being underutilized.

Furthermore, the vehicle assignment problem is usually done via a centralized method, such as optimization [5]–[7]. However, this method requires drivers to share a lot of information with all other vehicles and adhere to the assignment provided by the centralized solver. Although fleets belonging to the same company may be willing to follow the centralized assignment, it may not be suitable for situations with numerous competitors or a large number of independent drivers. As a result, using distributed methods that require vehicles to share limited information with only limited groups can be more favorable [8], [9].

Rebalancing policy is also studied to improve efficiency and alleviate congestion problems [10]–[12]. Rebalancing works by moving idle vehicles to another location based on different criteria and purposes, e.g., directly serving

Kaier Liang and Cristian-Ioan Vasile are with the Mechanical Engineering and Mechanics Department at Lehigh University, PA, USA: {kal221, cvasile}@lehigh.edu

other unassigned requests, avoiding congestion, increasing the likelihood of picking up requests, and thus improving performance. However, rebalancing can be also effective in terms of fairness for the vehicles. As idle vehicles being mobilized by rebalancing can also receive more utilities in the future. In this paper, we use the rebalancing approach to improve the fairness for drivers, specifically to balance their collected utilities over a period of time.

Another aspect that has received increasing attention, is the idea of moving from simple pick-up and drop-off requests to more complex demands that do not require customers to plan out trips for their tasks. This is especially important for unmanned transportation. Moreover, requests may need to share the same vehicle or use more than one. To accommodate these two problems, we use Linear Temporal Logic (LTL) to model requests in the vehicle routing problem [13]. Temporal logics have been successful in specifying and automating the synthesis of control and motion policies for robots [14]–[17] and dynamical systems [18]–[20]. In this work, we leverage automata-based techniques [21] to compute small routing problems with LTL requests, and employ a distributed auction algorithm to assign the requests to vehicles.

The contributions of this work are the following: 1) We define a distributed auction assignment algorithm with temporal logic demands where all computation is performed based on inter-vehicle communication and no vehicle has a special role (e.g., centralized bidding), 2) We propose a rebalancing scheme to move idle vehicles to more rewarding locations that takes into account fair distribution of vehicles' cumulated utility, 3) We show via case studies in a large environment in mid-Manhattan with a large fleet of vehicles and a number of requests that our distributed assignment method does not degenerate the performance of the Mobility-on-Demand system compared to a centralized approach. Moreover, our algorithms significantly reduce the deviation of utility and increase the minimum utility, which leads to fairer distribution for vehicles.

II. PRELIMINARIES

In this section, we introduce the notation used in the paper and review concepts in formal language and automata theory.

We denote the set of real and integer numbers as $\mathbb{R}$ and $\mathbb{Z}$, respectively. The real and integer numbers greater than a are denoted by $\mathbb{R}_{>a}$ and $\mathbb{Z}_{>a}$. Similarly, we have $\mathbb{R}_{\geq a}$ and $\mathbb{Z}_{\geq a}$ for real and integer numbers greater or equal to a. For a finite set S, we denote its cardinality and the power set as $|S|$ and 2^S.

Definition 1 (Finite Automaton). *A deterministic finite state automaton (DFA) is a tuple $\mathcal{A} = \left(Q_\mathcal{A}, q_{init}^\mathcal{A}, 2^\Pi, \delta_\mathcal{A}, F_\mathcal{A}\right)$, where $Q_\mathcal{A}$ is a finite set of states; $q_{init}^\mathcal{A} \in Q$ is the initial state; 2^Π is the input alphabet; $\delta_\mathcal{A} : Q_\mathcal{A} \times 2^\Pi \to Q_\mathcal{A}$ is a transition function; $F_\mathcal{A} \subseteq Q_\mathcal{A}$ is the set of accepting states.*

An input word $\boldsymbol{\sigma} = \sigma_0 \sigma_1 \ldots \sigma_n$ over alphabet 2^Π generates the *trajectory* of the DFA $\mathbf{q} = q_0 q_1 \ldots q_n$ with $q_{init} = q_0$ and $q_{k+1} = \delta_\mathcal{A}(q_k, \sigma_k)$, for all $k \in \{0, \ldots, n-1\}$. The trajectory $\mathbf{q}$ is called *accepting* if $q_n \in F_\mathcal{A}$.

Definition 2 (scLTL). *A co-safe Linear Temporal Logic (scLTL) formula over a set of atomic propositions Π is defined recursively as:*

$$\phi ::= \pi \mid \neg\pi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2 \mid \Diamond\phi,$$

where ϕ_1, ϕ_2 are scLTL formulae, $\pi \in \Pi$ is an atomic proposition, $\neg$ (negation), $\wedge$ (disjunction), and $\vee$ (conjunction) are Boolean operators, and $\mathcal{U}$ (until), $\bigcirc$ (next), and $\Diamond$ (eventually) are temporal operators.

The semantics of scLTL formulae are defined over infinite words with symbols from 2^Π. Intuitively, $\bigcirc\phi$ holds if ϕ is true at the next position in the word; $\phi_1 \mathcal{U} \phi_2$ expresses that ϕ_1 is true until ϕ_2 becomes true; and $\Diamond\phi$ expresses that ϕ becomes true at some future position in the word. The formal definition of the semantics can be found in [22]. Given a word $\boldsymbol{\sigma}$ over the alphabet 2^Π that satisfies the scLTL formula ϕ, we denote the satisfaction as $\boldsymbol{\sigma} \vDash \phi$. A finite word $\boldsymbol{\sigma}$ satisfies scLTL formula ϕ if for all infinite $\boldsymbol{\sigma}'$ the concatenated (infinite) word $\boldsymbol{\sigma}\boldsymbol{\sigma}' \vDash \phi$. The finite word $\boldsymbol{\sigma}$ is *minimal* if none of its prefixes satisfies ϕ.

scLTL formulae can be translated to DFAs using off-the-shelf tools such as scheck [23] and spot [24].

Definition 3 (Weighted Transition System). *A weighted transition system (WTS) is a tuple $\mathcal{T} = (S, s_{init}, D, W, \Pi, L)$, where S is a finite set of states, $s_{init} \in S$ is the initial state, $D \subseteq S \times S$ is a transition relation, $W : D \to \mathbb{R}_{\geq 0}$ is a weight function, Π is a set of atomic propositions and $L : D \to 2^\Pi$ is a labeling function.*

The transition from the current state s at time t to the next state s' is reached at time $t' = t + W((s, s'))$ if $(s, s') \in D$. A *trajectory* of $\mathcal{T}$ is a finite sequence $\mathbf{s} = s_0 s_1 \ldots s_n$, such that $s_0 = s_{init}$, and $(s_k, s_{k+1}) \in D$ for all $k \in \{0, \ldots, n-1\}$. The length of the trajectory $\mathbf{s}$ is n, and its total duration is $W(\mathbf{s}) = \sum_{i=0}^{n-1} W((s_i, s_{i+1}))$. The *output trajectory* induced by $\mathbf{s}$ is $\mathbf{o} = L(s_0)L(s_1) \ldots L(s_n)$. A finite trajectory $\mathbf{s}$ satisfies a scLTL formula ϕ, denoted $\mathbf{s} \vDash \phi$, if the induced output trajectory $\mathbf{o} = L(\mathbf{s})$ satisfies ϕ.

III. Problem Formulation

In this section, we formulate the fair mobility-on-demand problem with requests expressed as scLTL specifications. The objective is to sequentially generate assignments for incoming scLTL requests to a fleet of vehicles, with the goal of minimizing the total travel time and ensuring fairness among the fleet of vehicles.

A. Vehicle, Environment, and Request Models

The fleet of vehicles $\mathcal{V} = \{v_1, v_2, \ldots, v_p\}$ is deployed in a road network with intersections S and roads $D \subseteq S \times S$. The transition $(s, s') \in D$ represents a road from intersection

s to s'. Each vehicle $v \in \mathcal{V}$ is initially located at $s_{v,init} \in S$. All vehicles' motion evolves in discrete time $t \in \mathbb{Z}_{\geq 0}$ synchronized via a global clock. The traversal duration of road (s, s') is $W((s, s')) \in \mathbb{Z}_{>0}$.

Vehicles are tasked with satisfying a finite set of request $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$ that arrive sequentially over the horizon time $H \in \mathbb{Z}_{>0}$. A request $r \in \mathcal{R}$ is defined as a tuple $r = (\pi_{pick,r}, \phi_r, t_{req,r}, \rho_r, \Omega_{\max,r}, \Delta_{\max,r})$, where

- $\pi_{pick,r}$ is a proposition marking the pick-up location;
- ϕ_r is the scLTL formula specifying the request;
- $t_{req,r} \in \{0, \ldots, H\}$ is the request's arrival time;
- $\rho_r \in \mathbb{Z}_{>0}$ is the number of required seats;
- $\Omega_{\max,r} \in \mathbb{Z}_{>0}$ is the maximum waiting time, i.e., the latest accepted pick-up time is $t_{req,r} + \Omega_{\max,r}$;
- $\Delta_{max,r} \in \mathbb{Z}_{>0}$ is the maximum allowed delay.

The maximum transportation capacity of vehicle $v \in \mathcal{V}$ is $Cap_v \in \mathbb{Z}_{>0}$, while the *available capacity* at time t is $c_v(t) \in \{0, \ldots, Cap_v\}$. Vehicle v is *available* at time t if $c_v(t) > 0$, it is *occupied* if $c_v(t) = 0$, and *vacant* if $c_v(t) = Cap_v$. The sets of available and vacant vehicles at time t are $\mathcal{V}_t^a$ and $\mathcal{V}_t^{vac}$, respectively.

The delay Δ_r is the difference between the actual and optimal satisfaction duration. Formally, $\Delta_r = \max_{v \in V} t_{drop,r,v} - t_{req,r} - t_r^\star$, where $t_{drop,r,v}$ is the drop-off time of request r by vehicle v, and $t_r^\star$ is the optimal satisfaction time, i.e., the minimum duration to fulfill the request if a vehicle picks up the request at $t = t_{req,r}$ and does not share with other requests. We require that $\Delta_r \leq \Delta_{\max,r}$.

At current time $t \in \mathbb{Z}_{\geq 0}$, a request is *active* if $t_{req} \leq t$ and it has not been picked-up yet; a request is *in progress* if it has been picked up and not completed. The sets of active and in progress requests at time t are $\mathcal{R}_t^a$ and $\mathcal{R}_t^p$, respectively.

An assignment $Asg_t : \mathcal{R}_t^a \to \mathcal{V}_t^a$ at time $t = t_{req,r}$ allocates active requests to vehicles. If the assignment $Asg_t(r) = \varnothing$, then r is *unassigned* at time t. In case this holds for all $t \in \{t_{req,t}, \ldots, t_{req,r} + \Omega_{max,r}\}$, r is unassigned. Requests that are *in progress* cannot be reassigned, and vehicles need to be *available* before picking up new requests. Between request arrivals times, i.e., $t \neq t_{req,r}$, assignments do not change. The travel duration for the vehicle v fulfilling request r starts from the time $t_{asgmt,r,v}$ when r is assigned to v until v is dropped off at time $t_{drop,r,v}$. Formally, we have

$$\sigma_v(r) = t_{drop,r,v} - t_{asgmt,r,v}, \tag{1}$$

Our objective is to minimize the total traveling duration for all requests defined as

$$J = \sum_{r_i \in \mathcal{R}} \sigma_{v_i}(r_i), \tag{2}$$

where v_i is the vehicle satisfying request r_i.

This problem can be solved using centralized methods such as optimization techniques [25], [26]. However, centralized approaches may not be able to handle disruptions well in real-time, e.g., vehicles entering and leaving the system and changes in the environment and requests. These issues are compounded by the need to collect information into a central node for decision making which may lead to delays. Moreover, for vehicle-request problems, each vehicle usually makes individual choices, and vehicles may not be willing to disclose the information to others. Therefore, in this paper,

we seek distributed assignment methods that avoid the need for centralized data collection.

We assume that all agents can communicate with each other, e.g., via broadcasting to the entire fleet or a subgroup of vehicles. In this paper, a *distributed assignment* Asg_t at time t is defined as a assignment function computed by each vehicle based on messages exchanged with other vehicles, and no vehicle takes a special role in decision-making and coordination.

Problem 1 (Distributed Assignment). *Given the set of vehicles $\mathcal{V}$ deployed in environment $\mathcal{T}$, and the set of requests $\mathcal{R} = \{r_1, \ldots, r_m\}$ arriving sequentially over time horizon H, compute distributed assignments Asg_t at each sample time $t \in \{0, \ldots, H\}$ and routes $\mathbf{s}_v$ for all vehicles $v \in \mathcal{V}$ such that the total servicing time J is minimized.*

B. Fairness

For vehicle assignment problems, the serving rate or customer satisfaction is a crucial factor. However, it is equally important to consider drivers' viewpoints in terms of the fairness of allocating requests. The utility for a vehicle v for a given time period h is the sum of the onboard passengers:

$$U_v = \sum_{t=0}^{h} (Cap_v - c_v(t)). \tag{3}$$

Vehicles' utilities may vary greatly over the service horizon H. Thus, it is important to ensure fair assignment of requests while maintaining good overall performance of the fleet in terms of the total travel time for requests satisfaction J.

There are different criteria to quantify fairness, such as envy-free fairness, max-min fairness, and proportionality fairness [27]. In this paper, we use the *max-min utility* and *deviation of utility* as the two quantities to measure the fairness of the vehicles.

The max-min fairness criterion emphasizes the maximization of the least utility that a vehicle obtains, i.e., it captures the lower bound or the worst case of utility. This criterion is widely used in many applications [28]. The deviation of the utility fairness criterion, on the other hand, captures the utility distribution from the perspective of the entire group, as it directly reflects the utility spread among all vehicles.

In the vehicle assignment scenario, multiple factors can contribute to significantly uneven utility results. Vehicles' location in the road network impacts their chances of picking up requests due to spatial and temporal variations of requests' arrival. Secondly, differences in utility between requests and their limited number can lead to some vehicles servicing high utility requests while others are assigned lower utility ones or not at all. This may happen even in the case of a uniform probability distribution of requests over space and time.

The first case, due to spatial and temporal variation, rebalancing strategies can be used to mitigate the effects of request arrival differences over the road network. Rebalancing works by moving idle vehicles to another location to increase their chances of being assigned requests.

Problem 2 (Fair Rebalancing). *Given the set of vacant vehicles $\mathcal{V}^{vac}$ deployed in environment $\mathcal{T}$, compute the rebalancing scheme such that the chances of idle vehicles picking up requests in the future increase.*

For the second case, due to requests' utility differences, we impose that assignments are distributed in a fair way in the sense of max-min and deviation of utility criteria.

Problem 3 (Distributed Fair Assignment). *Given the set of vehicles $\mathcal{V}$ deployed in environment $\mathcal{T}$, and the set of requests $\mathcal{R} = \{r_1, \ldots, r_m\}$ arriving sequentially over time horizon H, compute distributed assignments Asg_t at each sample time $t \in \{0, \ldots, H\}$ and routes $\mathbf{s}_v$ for all vehicles $v \in \mathcal{V}$ in such that J is minimized and vehicles' utilities are allocated fairly.*

Summary of the approach.

For a fixed time sample interval, we conduct an auction for each active request to available vehicles. First, we construct product automata between the motion model (road network) of a vehicle and the DFAs corresponding to the requests. The route is then computed via the shortest path method (e.g., Dijkstra algorithm) applied on the product automaton graph and projection onto the motion model. If the maximum waiting and delay time is permissible, we allow the vehicles to generate the bid for the requests. After assigning the requests to the vehicles based on the auction results, we conduct a rebalancing for each idle vehicle to move vehicles to more ideal locations.

IV. SOLUTION

A. Fair Auction Based Assignment Scheme

The auction algorithm is a widely used approach for solving assignment problems in a distributed manner. The algorithm consists of two phases: the bidding phase and the assignment phase. During the bidding phase, each agent (in our case, each vehicle) makes a bid for each item (i.e., request). Then, during the assignment phase, the item is assigned to the agent with the highest bid. This process is repeated iteratively until there is no change in the assignment. The auction algorithm is known to be optimal and has a polynomial runtime for assignment problems [29].

We modify the standard algorithm to account for fair allocation in addition to optimizing an objective function. In our specific setting, the objective is to minimize the total traveling time, as defined by equation (2), with the requests as the items for auction and the vehicles as the bidders. To consider fairness, we add an intermediate *Weight Correction Phase* between bidding and assignment. The auction algorithm we use for our vehicle assignment problem is outlined in Alg. 1. In the algorithm, we use two communication primitives: (a) broadcasting function $broadcast(msg, V)$ that sends message msg to all vehicles in V, and (b) receive function $recv(v')$ that returns the message sent by agent v'. We assume that no packages are lost, and they are received in the same order they are sent. Thus, the receive function $recv()$ is used in blocking mode.

To find the minimum of the objective function, the auction algorithm is used in reverse. We use the travel time with opposite sign to compute the first and second most rewarding requests in lines 5-6 based on the utility value defined in equation (3). Specifically, the vehicles prefer requests that induce lower travel times. During the bidding phase, each available vehicle places a bid for the most desirable request. This utility value takes into account the constraints of maximum waiting time and the delay time for the request.

The bid amount is calculated in line 7 and is the sum of the request's price, the difference between the first and second most desirable request's utility difference, and a slack constant variable ϵ. This constant is typically set as $\frac{1}{N}$, where N is the number of bidders. The price of a request is initialized with the negative of the smallest travel time of any request for the vehicle at line 3. Agents broadcast their preferred request (line 8) to the fleet, and construct the *bidding group* G of other agents interested in the same request (line 9).

After the bidding phase, a weight correction phase is added to promote fairness. The weight correction is computed using equation (4), which adjusts the original travel utility based on the difference between the vehicle utility and the average utility of all vehicles in the same bidding group G. This allows vehicles with low history utility to increase their bids beyond their actual bidding capability, giving them a greater chance of winning the auction. The weight correction phase aims to balance the auction and prevent vehicles from continuously dominating the auction process.

$$\text{WeightCorrection}(v, U_{v_{avg},G}) = \alpha \cdot (U_v - U_{v_{avg},G}), \quad (4)$$

where α is a constant tuning parameter and G is a bidding group of vehicles. $U_{v_{avg},G}$ is the average history utility for all $v \in G$. We employed the weight correction in our previous integer linear programming (ILP) approach [26], which requires all vehicles to send their history utility to a central node. However, since our goal is to have a distributed implementation, we restricted the weight correction to be performed only within the same bidding group. This means that vehicles that bid on the same request adjust their bids only locally inside the group. This modification enables us to maintain the distributed nature of our approach. The communication between agents in the bidding group G is captured in lines 10-11 of Alg. 1. Vehicles within G exchange their utility histories computed using equation 3 to compute the mean utility value of the group (line 12).

Finally, during the assignment phase, the request is allocated to the vehicle that offers the highest bid (lines 14-17), and the auction is executed iteratively. In the subsequent rounds, other vehicles can increase their bids until the highest bid and bidder remain the same. Note that the price of the request is also updated at the end of each round at line 18.

It is important to note that even though the auction algorithm restricts a vehicle to bid for only one request at each round, we can still enable vehicle sharing by allowing vehicles with $c_v > 0$ to participate in the next auction, as long as the total capacity does not exceed the maximum Cap_v [9].

B. Automata-based Route Planning

To conduct an auction in the bidding phase, we need to determine which vehicles are eligible to bid for which requests and what the utility (essentially the route) is for each request. We obtain this information through the construction of product automata.

The requests are represented as scLTL formula and vehicles are represented as a WTS. Formally, we have the $\mathcal{T}_v = (S, s_{\text{init}}, D, W, \Pi, L)$ that captures vehicle v's motion in the environment. The set of propositions Π includes the active requests' pick-up propositions $\pi_{pick,r}$.

Algorithm 1: Fair Auction Algorithm

Input: $\mathcal{T}$ – the road map, $\mathcal{V}^a$ – available vehicles, $\mathcal{R}$ - active requests
Output: Asg_t – Assignment Function

1 **foreach** $v \in \mathcal{V}^a$ **do**
　　// Initialization
2 　　Compute $\sigma_v(r)$ for each $r \in \mathcal{R}$ using $\mathcal{P}_r = \mathcal{T} \times \mathcal{A}_r$
3 　　$p_j \leftarrow -\min_{r \in \mathcal{R}} \sigma_v(r)$ // Set initial price
4 　　**while** Asg_t *changed* **do**
　　　　// I. Bidding Phase
5 　　　　$U_{v,j} \leftarrow \max_{r_j \in \mathcal{R}}(-\sigma_v(r_j))$ // Find the most rewarding Request j
6 　　　　$U_{v,k} \leftarrow \max_{r_k \in \mathcal{R} \setminus \{r_j\}}(-\sigma_v(r_k))$ // Find the second most rewarding Request k
7 　　　　$B_{v,j} \leftarrow p_j + U_{v,j} - U_{v,k} + \epsilon$ // Initial bid
8 　　　　broadcast$(r_j, \mathcal{V}^a)$
　　　　// Group of vehicles bids for r_j
9 　　　　$G \leftarrow \{v\} \cup \{v' \in \mathcal{V}^a \setminus \{v\} \mid recv(v') = r_j\}$
　　　　// II. Weight Correction Phase
10 　　　　broadcast(U_v, G) // Broadcast to G
11 　　　　$U_{v'}^G \leftarrow recv(v'), \forall v' \in G \setminus \{v\}$
　　　　// Compute mean utility history for G
12 　　　　$U_{v_{avg},G} \leftarrow \frac{1}{|G|} \sum_{v' \in G} U_{v'}^G$
　　　　// Update bid
13 　　　　$B_{v,j} \leftarrow B_{v,j} + \text{WeightCorrection}(v, U_{v_{avg},G})$
　　　　// III. Assignment Phase
14 　　　　broadcast$(B_{v,j}, G)$ // Broadcast bid to G
15 　　　　$B_{v',j} \leftarrow recv(v'), \forall v' \in G \setminus \{v\}$
16 　　　　$v^* \leftarrow \arg\max_{v' \in G} B_{v',j}$
17 　　　　$Asg_t(r_j) \leftarrow v^*$ // Assign request to the largest bidder
18 　　　　$p_j \leftarrow B_{v^*,j}$ // update price

19 **return** Asg_t

For every available vehicle v and active request r, we construct a weighted product automaton $\mathcal{P}_{rv} = \mathcal{T}_v \otimes \mathcal{A}_r$. $\otimes$ is a product operation. $\mathcal{T}_v$ is the transition system for the vehicle v with initial position s_{init} set as the vehicle' current position. If the vehicle already has an onboard passenger r', we construct the weighted product automaton $\mathcal{P}_{v,r,r'} = \mathcal{T}_v \otimes \mathcal{A}_r \otimes \mathcal{A}_{r'}$ to validate if the r can be served together without violating the constraints for r and r'. After obtaining the product automata, we use graph search methods such as Dijkstra's algorithm to compute an admissible path [30].

The formal definition of the product automaton is the following.

Definition 4 (Weighted product automaton at time t)**.** *The weighted product automaton $\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_m$ of vehicle v at time t is a tuple $(Q_\mathcal{P}, Q_{init,\mathcal{P}}, \delta_\mathcal{P}, F_\mathcal{P}, W_\mathcal{P})$, where*

- $Q_\mathcal{P} \subseteq S \times Q_{\mathcal{A}_1} \times \ldots Q_{\mathcal{A}_m}$;
- $Q_{init} = \{s_j, q_{1,k}, \cdots, q_{m,k}\}$, *where s_j is the current state of vehicle v in the map;*

$$q_{i,k} = \begin{cases} \delta_i(\pi_{pick,i}, L(s_j)) & \text{if } t_{pick,r_i} = t \\ \delta_i(q_{i,k-1}, L(s_j)) & \text{if } t_{pick,r_i} < t \\ q_{init}^{\mathcal{A}_i} & \text{else,} \end{cases}$$

where t_{pick,r_i} is the pick-up time for r_i, k is the current (event) step associated with time t, and $q_{i,k-1}$ are the states of the request at the previous step;

- $\delta_\mathcal{P} \subseteq Q_\mathcal{P} \times Q_\mathcal{P}$ *is a transition function:*
  $((s, q_1, \ldots, q_m), (s', q'_1, \ldots, q'_m)) \in \delta_\mathcal{P}$ *if and only if*

$(s, s') \in R$ and $(q_i, L(s'), q_i') \in \delta_i$;

- $F_{\mathcal{P}} = \{(s, q_{1,k}, \ldots, q_{m,k}) \mid q_{i,k} \in F_i, \forall i \in \{1, \ldots, m\}\}$;
- $W_{\mathcal{P}}$: $\delta_{\mathcal{P}} \rightarrow \mathbb{R}_+$ *is the weight function given by* $W_{\mathcal{P}}(((s, q_1, \ldots, q_m), (s', q_1', \ldots, q_m'))) = W(s, s')$.

A satisfying path **q** *in* $\mathcal{P}$ *connects the initial state* Q_{init} *with a reachable final state* $F_{\mathcal{P}}$. *If such a path exists, we project it onto* $\mathcal{T}$ *by taking the first component of each state in the state path* **q**.

C. Fair Rebalancing

In real-life scenarios, the road map for request generation is often non-uniformly distributed. For instance, certain areas like the city center or airport have a higher probability of generating requests than rural areas where requests are infrequent. As a result, due to the maximum waiting time and maximum allowed delay, there can be a significant difference in utility among vehicles, leading to an unfair distribution of utility for the vehicles. To address this problem, we propose a rebalancing scheme that reduces these unfair effects. For each node $s \in S$ in the road map $\mathcal{T}$ at time t, we calculate the potential utility $P_t(s)$ as:

$$P_t(s) = \frac{Pr_t(s)}{1 + N_{v,t}} * U_{r_{avg}}, \tag{5}$$

where $Pr_t(s)$ is the probability of a request arriving at node s at a given time t; $N_{v,t}$ is the number of nearby idle vehicles at time t for a fixed distance range, and $U_{r_{avg}}$ is the average utility for requests arriving at s which can be obtained from the history data. The term $N_{v,t} + 1$ is added in the denominator to avoid division by zero. This formula considers both the probability of a new request arriving and the number of competing vehicles nearby, reflecting the potential utility for a vehicle at location s and time t.

We simplify the problem by assuming $Pr_t(s)$ is independent of time and only related to locations. However, for a large road map and a significant number of vehicles, simply calculating the highest utility $P_{v,t}(s)$ for every vehicle v and rebalancing the vehicle to the corresponding location s can be expensive and inadvisable for several reasons:
(1) The highest location can be the same for all vehicles, which can be seen from the independence with respect to a specific vehicle in equation (5).
(2) Rebalancing itself will require some cost as it will require idle vehicles to move to another location. Therefore the highest potential location that is far away may be less attractive than a location with a smaller value but close.

To deal with these problems, we use a slack parameter and a distance search window. The rebalancing target location is calculated in Alg. 2:

To implement the rebalancing scheme, we first sort all potential rebalancing locations based on their degree or distance from the initial location, as specified in line 2 of Alg. 2. The distance search window $k_w(v)$ is used for vehicle v to ensure that the rebalancing search is not performed to a location that is too far away preventing making unnecessary searches. Then in line 5, starting from the first-degree nodes, or the nearest nodes, we then find the maximum potential utility node using equation (5). To ensure that vehicles take account of both the distance and utility, we use a constant slack variable $k_a > 1$ in line 7 to increase the perceived cost

---

Algorithm 2: Fair Rebalancing Algorithm

Input: $\mathcal{T}$ – the road map, $\mathcal{V}^{vac}$ – available vehicles, k_w – Distance Search Window

Output: Reb_t – Rebalance Function

1 **foreach** $v \in \mathcal{V}^{vac}$ **do**
2    $\mathcal{G}_v \leftarrow Neighbor(s_{v,init}, \mathcal{T}, k_w(v))$
3    $P_{v,t}(s_{v,tar}) \leftarrow P_{v,t}(s_{v,init})$
4    $s_{v,tar} \leftarrow s_{init}$
5   **foreach** $N_{deg,i} \in \mathcal{G}_v$ **do**
6      $P_{v,t}(s_{v,max}) \leftarrow \max_{s_j \in N_{deg,i}} P_{v,t}(s_j)$
7   **if** $P_{v,t}(s_{v,max}) \geq P_{v,t}(s_{v,tar}) * k_a$ **then**
       $s_{v,tar} \leftarrow s_{v,max}$
8      $P_{v,t}(s_{v,tar}) \leftarrow P_{v,t}(s_{v,max})$
9    $Reb_t(v) \leftarrow s_{v,tar}$
10 **return** Reb_t

---

of rebalancing to outer degree nodes or farther nodes. The vehicle will only choose to rebalance to a high degree node if the potential utility is significantly greater than the current target rebalancing node.

The auction and rebalancing are implemented sequentially. At a given time sample frequency, an auction is conducted to assign available vehicles to every unassigned request. Then the rebalancing is conducted to move idle vehicles to move to better locations. Therefore, vehicles are either in progress to serve requests or in rebalancing to move to another location.

V. SIMULATION

In this section, we present the simulation results to demonstrate the performance of distributed fair assignment and the rebalancing scheme.

A. Simulation Specifications

The road map for the simulation is used as the Mid-Manhattan map, which contains 184 nodes, and the weights for every edge are acquired by real travel duration from taxi driving data [25]. We varied the request generation probabilities and the number of requests to evaluate the fairness performance of the system. Three different maps were used for the simulations, namely the center map, corner map, and two peaks map, with request generation probabilities as shown in Fig. 2. These maps are characterized by high probability areas where requests are more likely to be generated. So that it can reflect the uneven distribution of requests in real-life scenarios.

The simulation duration is set to 1000 seconds with varying the number of vehicles and requests. The initial positions of all vehicles are generated in a uniform distribution. The scLTL formulas for the requests are generated from the following scLTL pattern stochastically.

scLTL pattern:
$$\tilde{\phi}_1(s_{pick}, s_1, s_2) = \Diamond(s_{pick} \land \Diamond(s_1 \land \Diamond(s_2))),$$
$$\tilde{\phi}_2(s_{pick}, s_1, s_2) = \Diamond(s_{pick} \land \Diamond((s_1 \lor s_2) \land s_3)),$$
$$\tilde{\phi}_3(s_{pick}, s_1, s_2, s_3) = \Diamond(s_{pick} \land \Diamond(s_1 \land (s_2 \lor s_3)))$$

where s_i are locations in the road map. The arrival time is generated according to a uniform Poisson process. The locations are chosen based on the corresponding probability of request generation in the road map.

Throughout the simulation, we perform the auction and rebalancing every 10 seconds. Additionally, we set the maximum waiting time and delay time to 40 and 100 seconds.

Fig. 1: Different settings for the Mid-Manhattan Map: the colors in nodes represent the probability of a new request arrival.

Fig. 2: rebalancing performance in different map settings

Fig. 3: Performance comparison for rebalancing and weight correction (center map: 20 vehicles, 300 requests) from the minimum utility, utility deviation and average utility

B. Simulation Results

In the simulation results shown in Fig. 2, we consider 20 vehicles and a varying number of requests from 200 to 400 to demonstrate the effect of the rebalancing strategy. Each data point in the figure is the average result of 20 simulations. The fairness is compared using the minimum and deviation utility. Fig. 2 shows we can increase the minimum utility and decrease the deviation utility consistently without degenerating the serving rate in all three map settings.

Fig. 3 shows the comparison between the planning with and without weight correction and rebalancing settings. We can see the improvement of introducing rebalancing or the weight correction from the two fairness criteria; the settings that adopt the rebalancing or weight correction can increase the minimum utility and decrease the deviation utility. And the setting performs best when it uses the rebalancing and the weight correction together.

In Fig. 3, we also notice that using balancing or weight correction does not affect the average utility. This suggests that although we cannot increase the total utility for the entire system, we can adjust the utility distribution in a fair way by increasing the minimum utility and decreasing the deviation.

Fig. 4: Comparison between ILP and auction without using rebalancing and weight correction

Comparison with the centralized approach: here we present the performance comparison between the auction algorithm and centralized algorithm using ILP [26]. Although both algorithms can obtain the optimal solution, the algorithms' implementations are different. First, the ILP setting allows more than one request to be assigned together at one step due to the optimization nature, whereas the auction algorithm can only assign one request to one vehicle at one-time. Furthermore, since both methods are run continuously throughout the simulation, it is not possible to obtain the global optimal solution. Therefore, the current optimal solution does not imply the global property, as future events cannot be predicted at the current time step.

For the comparison shown in Fig. 4, we compare the auction and ILP methods for the setting with 20 vehicles and a varying number of requests. While both approaches are to minimize the traveling time, this is not easy to quantify and compare directly. Thus, we evaluate the average utility and the number of unassigned requests. The average utility and the number of unassigned requests capture the running quality from the requests and vehicles' perspectives. For the comparison, we used both the auction and ILP settings without rebalancing and weight correction. We see that the two approaches perform very similarly which is expected.

VI. CONCLUSIONS

In conclusion, this paper presents a novel approach to the problem of fair assignment and rebalancing in Mobility-On-Demand systems. Our proposed distributed assignment method reduces the need for a central authority for coordination. The introduction of the rebalancing scheme leads to a fairer distribution of requests for vehicles, as demonstrated by an increase in the minimum utility and a decrease in the utility deviation compared to the baseline. By modeling requests using temporal logic formulas, our approach accommodates complex demand patterns. The results of our study demonstrate the efficacy of the proposed method in achieving fairer vehicle assignment in Mobility-On-Demand systems.

REFERENCES

[1] T. Teubner and C. M. Flath, "The economics of multi-hop ride sharing," *Business & Information Systems Engineering*, vol. 57, no. 5, pp. 311–324, 2015.

[2] S. Liyanage, H. Dia, R. Abduljabbar, and S. A. Bagloee, "Flexible mobility on-demand: An environmental scan," *Sustainability*, vol. 11, no. 5, p. 1262, 2019.

[3] Y. Cao, S. Wang, and J. Li, "The optimization model of ride-sharing route for ride hailing considering both system optimization and user fairness," *Sustainability*, vol. 13, no. 2, p. 902, 2021.

[4] M. D. Aleksandrov, "Fair division meets vehicle routing: Fairness for drivers with monotone profits," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, pp. 915–920, IEEE, 2022.

[5] L. Foti, J. Lin, O. Wolfson, and N. D. Rishe, "The nash equilibrium among taxi ridesharing partners," in *ACM SIGSPATIAL Intl Conf on Advances in Geographic Information Systems*, pp. 1–4, 2017.

[6] Y. Lin, W. Li, F. Qiu, and H. Xu, "Research on optimization of vehicle routing problem for ride-sharing taxi," *Procedia-Social and Behavioral Sciences*, vol. 43, pp. 494–502, 2012.

[7] M. W. Levin, "Congestion-aware system optimal route choice for shared autonomous vehicles," *Transportation Research Part C: Emerging Technologies*, vol. 82, pp. 229–247, 2017.

[8] V. Pandey, J. Monteil, C. Gambella, and A. Simonetto, "On the needs for maas platforms to handle competition in ridesharing mobility," *Transportation Research Part C: Emerging Technologies*, vol. 108, pp. 269–288, 2019.

[9] A. Simonetto, J. Monteil, and C. Gambella, "Real-time city-scale ridesharing via linear assignment problems," *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 208–232, 2019.

[10] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *Intl Conf on Intelligent Transportation Systems*, pp. 220–225, IEEE, 2017.

[11] S. L. Smith, M. Pavone, M. Schwager, E. Frazzoli, and D. Rus, "Rebalancing the rebalancers: Optimally routing vehicles and drivers in mobility-on-demand systems," in *2013 American Control Conference*, pp. 2362–2367, IEEE, 2013.

[12] K. Spieser, S. Samaranayake, W. Gruel, and E. Frazzoli, "Shared-vehicle mobility-on-demand systems: a fleet operator's guide to rebalancing empty vehicles," in *Transportation Research Board 95th Annual Meeting*, no. 16-5987, Transportation Research Board, 2016.

[13] J. Tumova, S. Karaman, C. Belta, and D. Rus, "Least-violating planning in road networks from temporal logic specifications," in *Intl Conf on Cyber-Physical Systems*, pp. 1–9, IEEE, 2016.

[14] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4817–4822, IEEE, 2013.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[16] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI communications*, vol. 29, no. 1, pp. 151–162, 2016.

[17] D. Kamale, E. Karyofylli, and C.-I. Vasile, "Automata-based optimal planning with relaxed specifications," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6525–6530, 2021.

[18] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[19] G. A. Cardona, D. Saldaña, and C.-I. Vasile, "Planning for modular aerial robotic tools with temporal logic constraints," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 2878–2883, 2022.

[20] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5319–5325, IEEE, 2014.

[21] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*, vol. 15. Springer, 2017.

[22] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[23] T. Latvala, "Efficient Model Checking of Safety Properties," in *10th International SPIN Workshop*, Model Checking Software, pp. 74–88, Springer, 2003.

[24] A. Duret-Lutz, "Manipulating LTL formulas using Spot 1.0," in *Intl Symposium on Automated Technology for Verification and Analysis*, vol. 8172 of *LNCS*, (Hanoi, Vietnam), pp. 442–445, Springer, 2013.

[25] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[26] K. Liang and C.-I. Vasile, "Fair planning for mobility-on-demand with temporal logic requests," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1283–1289, IEEE, 2022.

[27] S. J. Brams, S. J. Brams, and A. D. Taylor, *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

[28] H. P. Young, *Equity: in theory and practice*. Princeton University Press, 1995.

[29] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of operations research*, vol. 14, no. 1, pp. 105–123, 1988.

[30] M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion," *Control and cybernetics*, vol. 35, no. 3, pp. 599–620, 2006.