Optimal On-the-fly Route Planning with Rich Transportation Requests

Cristian-Ioan Vasile*, *Member, IEEE,* Jana Tumova*, *Member, IEEE,* Sertac Karaman, *Member, IEEE,* Calin Belta, *Fellow, IEEE,* and Daniela Rus, *Fellow, IEEE*

***Abstract*—The paper considers the route planning problem for a vehicle with limited capacity operating in a road network. The vehicle is assigned a set of transportation requests that are more complex than traveling between two locations, may involve dependencies between their sub-tasks, and include deadlines and priorities. The requests arrive gradually over the deployment time-horizon, and thus replanning is needed for new requests. We address cases when not all requests can be serviced by their deadlines despite car sharing. We introduce multiple quality measures for plans that account for requests' delays with respect to deadlines and priorities. We formalize the problem as planning in a weighted transition system under syntactically co-safe LTL formulas. We develop an online planning and replanning algorithm based on the automata-based approach to least-violating plan synthesis and on translation to a Mixed Integer Linear Program (MILP). Furthermore, we show that the MILP reduces to graph search for a subclass of quality measures that satisfy a monotonicity property. We show the approach in simulations, including a case study on the mid-Manhattan road network over the span of 24 hours.**

***Index Terms*—Route Planning, Mobility on Demand, Autonomous Agents, Temporal Logic, MILP**

I. Introduction

THIS work is motivated by mobility-on-demand scenarios, where a single vehicle receives transportation requests from multiple customers over time and must service all of them. For example, a customer in a Manhattan road network illustrated in Fig. 1 would like to be picked up at the intersection of 7th Ave and 43rd St and brought to the intersection of Park Ave and 53rd St. Another customer is interested in being picked up at the Madame Tussauds, taken to the Rockefeller Center, while passing by a flower shop, where they can pick up flowers. A third customer would like to visit any shopping center within Midtown. Customers' requests need to be completed by their deadlines and according to associated priorities (e.g., membership level of customers in the mobility-on-demand system). Customers may share the vehicle, but never exceed its capacity. The planning goal is to compute a trip for the vehicle that services all requests before their given deadlines. Similar planning goals occur in package and service delivery using ground vehicles [1], aerial drones [2], or mixed fleets [3], [4] that may be required to satisfy complex transportation demands, too. For example, during the covid-19 pandemic, taxi drivers from Stockholm, Sweden were delivering home tests to people, who ordered them, and picking them up approximately 15 minutes later, which made their task more involved than the standard pick-up and drop-off. They had to decide how to route optimally in the road network to satisfy all demands. However, in many cases the problem does not have a solution that meets all the deadlines. What should the vehicle do then? Which transportation requests should be delayed and how much? What is the optimal route of the vehicle in the road network?

Fig. 1: The road network corresponding to part of mid-Manhattan is shown. Travel duration estimates are inferred from real taxi travel data in hourly increments.

In this work, we aim to design a route planning approach that can (i) accommodate rich transportation demands arriving sequentially in time and involving temporal or conditional dependencies, deadlines, priorities, and constraints on capacity of the vehicle, and (ii) resolve situations, when all demands cannot be met by the desired deadlines.

A. Related Work

We discuss the related work from two perspectives: First, we position the problem that we tackle in the context of route planning in road networks. Second, we position the proposed temporal logic-based solution in the context of literature that focuses on solving similar temporal logic planning problems in general terms, or for various autonomous planning applications.

1) Route planning: Route planning, and re-planning in the context of a autonomous mobility-on-demand system was recently considered in a number of related works. Real-time rebalancing policy for a shared fleet of autonomous vehicles

Cristian-Ioan Vasile is with Lehigh University, Bethlehem, PA, USA, email: cvasile@lehigh.edu. J. Tumova is with KTH Royal Institute of Technology, Stockholm, Sweden, e-mail: tumova@kth.se. Sertac Karaman is with Massachusetts Institute of Technology, Cambridge, MA, USA, email: sertac@mit.edu. Calin Belta is with the University of Maryland, College Park, MD, USA, email: calin@umd.edu. Daniela Rus is with Massachusetts Institute of Technology, Cambridge, MA 02139, USA, email: rus@csail.mit.edu.

*Authors contributed equally.

Manuscript received Month Day, Year; revised Month Day, Year.

was developed [5], [6] and the effects of the rebalancing on congestion was studied [7], [8]. Different route planning algorithms were proposed to improve the performance of the traffic network taking the mesoscopic perspective and using dynamic estimations of congestion or transportation demands [9], [10]. From a microscopic perspective, task assignment and (predictive) route planning for individual vehicles, or different variants of dial-a-ride problem with cost optimization were addressed by e.g., [11] or [12].

Related work also includes various versions of the vehicle routing problem in general [13], and especially dynamic vehicle routing with time-dependent travel times, typically captured through a stochastic model. Here, the goal becomes to find a path with the least expected travel time. For instance, [14] proposed to find a set of non-dominated shortest paths in a stochastic road network. The closest route planning literature to our work focuses on handling dynamically changing demands. For example, [15] tackled real-time demands and sought optimal departure times using Mixed Integer Linear Programming (MILP). [16] handled disruptions occurring during the execution of a vehicle route. Recently, [17] considers the dynamic vehicle routing problem with stochastic requests and proposes an approximate solution with knapsack-based linear models to scale to larger instances. This paper also includes a comprehensive literature review on the topic.

This paper differs from the state-of-the-art literature in its focus on handling a richer class of transportation planning problems than simple A-to-B travels, coverage, or multi-depot vehicle routing. To that end, we propose to use temporal logic formulas as a request specification. Temporal logics are expressive enough to capture all three above mentioned transportation request types, and much more than them. It should be noted that temporal logic formulas cannot be generally decomposed into a conjunction of simple tasks due to interdependencies introduced by nesting temporal operators [18]. In recent years, temporal logics have been widely adopted in robotics to specify robotic tasks due to their richness, rigorousness, resemblance to natural language, and the existence of algorithmic solutions to synthesize plans that provably satisfy a given temporal logic task [18]. Related work has focused on a variety of challenges, including consideration of system dynamics [19], [20].

2) Planning with temporal logic specifications: Since we aim to specifically focus on situations, when all demands expressed in temporal logic cannot be met simultaneously, the closest related work within temporal logic-based planning literature is the work handling unsatisfiable specifications that includes, e.g., finite least-violating planning [21], [22], [23]. The authors focus on finding the maximal part of the specification that can be satisfied by the system model and computing the corresponding plan for this part of specification only. In [24] and [25], a given temporal logic formula is systematically revised such that system satisfies it, and the modified formula is close to the original one. In contrast to our work, none of these consider deadlines or explicit time bounds associated with the temporal operators in the specification and hence neither their impact on solution quality and computation.

Quantitative models and specifications have been proposed in temporal logic-based planning, where minimization of the durations between revisits of specified locations is required in addition to satisfaction of temporal logic missions [26], [27]. Timed temporal logics have been chosen as a task and motion specification language for autonomous robots in several studies, tackling the problem for instance via automata-based formal synthesis in coupling with system abstractions [28], [29] or sampling-based planning [30], via reduction to MILP [31], [32], [33], or via timed Petri nets [34]. However, the problem these work focus on correct-by-design synthesis and not on least-violating planning with respect to a set of specifications. Related work also includes work on periodic replanning under knowledge updates [35], [36], [23]. The updates originate due to changes in the model, and not due to changes in the specifications, which we focus on in this paper.

Vasile et al. proposed a timed logic called Time Window Temporal Logic (TWTL), and the temporal relaxation of deadlines is considered under a measure resembling the bottleneck delay cost [37]. This logic was used to specify a surveillance task that needs to be satisfied infinitely many times in sequence by a multi-robot system [38]. However, that work does not consider task priorities, replanning based on task arrival, and multiple measures of violation. The authors considered only cumulative delay and bottleneck delay without priorities. While TWTL may be used to specify tasks with more complex timing constraints, such as tasks with time constrained subtasks and Boolean compositions of time constrained tasks aside from conjunction, the demands considered in this paper are more general.

B. Contribution

The contributions of this work are four-fold.

1) We formalize the problem of route planning under complex, gradually arriving, prioritized, and possibly mutually unsatisfiable transportation requests via temporal logic-based planning framework and propose three different cost functions to determine the quality of a routing plan based on delays in servicing the requests.
2) We design a general MILP-based solution to compute the optimal path with respect to a selected cost function and we instantiate the MILP formulation for the three different cost functions we have proposed.
3) We design a more efficient, linear-time graph search-based solution for a particular subclass of cost functions.
4) We evaluate the proposed formulation and solution, and compare the performance of the three proposed cost functions on several case studies, including a case study on the mid-Manhattan road network.

This paper builds on its preliminary conference version [39], but adds two new technical aspects, that make the problem more realistic and yield changes to the problem formalization and solution. First, we introduce the feature of car sharing with a vehicle capacity. Second, the transportation request is required to hold from the time of customer pick-up, not from the time of its arrival. Furthermore, we provide more elaborate experimental evaluation by adding a realistic case study of

route planning in the mid-Manhattan road network. To improve runtime performance, we have employed an abstraction method for the road network based on minimal paths between locations. We have added theoretical complexity and empirical scalability analysis. Lastly, we extended the description of the solution approach and made the mathematical programming encodings explicit.

To our best knowledge, this work is the first one that systematically integrates planning for an infinite sequence of gradually arriving demands in the form of temporal logic specifications with planning under infeasible deadlines. The route planning algorithms presented in this paper were used by [40] for combined motion and route planning for a vehicle tasked with servicing a stream of demands similar to the setup in this paper. The authors used the present work as a black box to ensure long-term satisfaction of transportation requests while ensuring minimum violation of rules of the road induces by local short-horizon motion plans.

II. Preliminaries

We denote the positive and nonnegative real numbers by $\mathbb{R}_+$ and $\mathbb{R}_0$. The power set and cardinality of a set S are 2^{S} and $|\mathsf{S}|$. Given an infinite sequence $w = w_1 w_2 \ldots$, we use w_j and $w_{n:m}$ to denote the $j-th$ element, and the segment $w_n \ldots w_m$. Specifically, $w_{1:m}$ is the prefix ending at the m-th position of w and $w_{n:\infty}$ is the suffix starting at the n-th position. If $n > m$ then $w_{n:m}$ is empty. The *concatenation* of a finite sequence w and a finite or an infinite sequence w' is denoted by $w \cdot w'$. For simplicity, we use $s \in w_1 w_2 w_3 \ldots$ to denote the membership of the element s in the set $\{w_1, w_2, w_3, \ldots\}$. The *i-th projection* proj_i of a tuple $(s_1, \ldots, s_n)$ is $\mathrm{proj}_i(s_1, \ldots, s_n) = s_i$. We will use proj_i to denote also the i-th projection proj_i of a sequence of tuples $(s_{1,1}, \ldots, s_{n,1}) \ldots (s_{1,m}, \ldots, s_{n,m})$, which is $\mathrm{proj}_i(s_{1,1}, \ldots, s_{n,1}) \ldots (s_{1,m}, \ldots, s_{n,m}) = s_{i,1} \ldots s_{i,m}$. The canonical basis of $\mathbb{R}^m$ is denoted by $\{\zeta_1, \ldots, \zeta_m\}$. For positive integer m, we denote $\langle m \rangle = \{1, \ldots, m\}$.

Definition 1 (WTS)**.** *A weighted deterministic transition system (WTS) is a tuple $\mathcal{T} = (S, s_{init}, R, W, \Pi, L)$, where S is a finite set of states; $s_{init} \in S$ is the initial state; $R \subseteq S \times S$ is a transition relation; $W : R \to \mathbb{R}_+$ is a weight function; Π is a set of atomic propositions; and $L : S \to 2^{\Pi}$ is a labeling function.*

Even though the WTS defined above does not have inputs, we call it *deterministic* to emphasize that transitions can be chosen deterministically. Given that the current state of the system is $s \in S$ at time t, by taking a transition $(s, s') \in R$, the system reaches the state s' at time $t' = t + W\big((s, s')\big)$. A *trace* $\tau = s_1 s_2 s_3 \ldots$ is an infinite sequence of states of $\mathcal{T}$, such that $s_1 = s_{init}$, and $(s_j, s_{j+1}) \in R$, for all $j \geq 1$. Each trace $\tau = s_1 s_2 s_3 \ldots$ is associated with the *time sequence* $\mathbb{T}(\tau) = t_1 t_2 t_3 \ldots$, where $t_1 = 0$, and $t_j = t_{j-1} + W\big((s_{j-1}, s_j)\big)$, for all $j \geq 2$. The time t_j denotes the time elapsed till reaching the j-th state s_i on the trace τ. The *word* produced by τ is the sequence $w(\tau) = L(s_1)L(s_2)L(s_3) \ldots$.

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

- *π and $\neg\pi$ are scLTL formulas*
- *if φ_1 and φ_2 are scLTL formulas, then also $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\mathcal{N}\varphi_1$, $\mathcal{F}\varphi_1$, and $\varphi_1 \,\mathcal{U}\, \varphi_2$ are scLTL formulas,*

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

An scLTL formula is interpreted over infinite sequences over 2^{Π}, such as the words produced by a WTS. Intuitively, π is satisfied by a word w if it holds true in its first position w_1 and $\neg\pi$ is satisfied if π is not true in w_1. $\mathcal{N}\varphi$ is satisfied if φ holds true in the next position, w_2, $\mathcal{F}\varphi$ if φ holds true eventually, in some w_i, $i \geq 1$, and $\varphi_1 \,\mathcal{U}\, \varphi_2$ holds true if φ_1 holds true until a position where φ_2 holds true.

A *good prefix* w.r.t. scLTL formula φ is defined as a finite prefix $w_{1:n}$, with the property that $w_{1:n} \cdot w'$ satisfies φ, for all w' over 2^{Π}.

Definition 3 (Minimal good prefix)**.** *Given a word w over 2^{Π} and ϕ over Π, a good prefix $w_{1:n}$ of w is* minimal*, if $w_{1:n-1}$ is not a good prefix.*

A trace τ of WTS satisfies φ if and only if it produces a word $w(\tau)$ that satisfies φ. A (minimal) *good trace prefix* is the one that produces a (minimal) good prefix.

Definition 4 (Finite automaton)**.** *A deterministic finite automaton is a tuple $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, where Q is a set of states; $q_{init} \in Q$ is the initial state; Σ is a finite alphabet; $\delta \subseteq Q \times \Sigma \to Q$ is a transition function; $F \subseteq Q$ is a set of finite states.*

A run of a finite automaton $\mathcal{A}$ over a finite word $w = \sigma_{1:n}$ is a sequence of states $\rho = q_{1:n+1}$, such that $q_1 = q_{init}$ and $(q_i, \sigma_i, q_{i+1}) \in \delta$, for all $1 \leq i < n$ and it is accepting if $q_{n+1} \in F$. In a nonblocking automaton, $\delta(q, \sigma)$ is defined for all $q \in Q$ and $\sigma \in \Sigma$.

For any scLTL formula φ over Π there exists a nonblocking deterministic finite automaton $\mathcal{A} = (Q, q_{init}, \Sigma, \delta, F)$, such that $\Sigma = 2^{\Pi}$, and the sets of all words accepted by $\mathcal{A}$ and all good prefixes of all words that satisfy φ coincide [41].

III. Problem Formulation

A. Model

We consider a vehicle in a road network that is modeled as a WTS $\mathcal{T} = (S, s_{init}, R, W, \Pi, L)$ with the following properties. The set of states S represents locations of interest of the road network, i.e. intersections, pick-up and drop-off points, etc. The vehicle's initial location is the initial state state s_{init}. We capture the motion of the vehicle in the road network with the transition relation R. The vehicle can move between locations s and s' if $(s, s') \in R$. The time duration of a transition is given by the weight function W. The vehicle can be stationary at any location which we model using self-transitions, i.e., $(s, s) \in R$ and $W(s, s) = \epsilon$, for all $s \in S$ and some small duration $\epsilon > 0$. We capture properties of interest such as

"Main road and 1st street intersection" or "a shopping mall" via the set of atomic propositions Π. All states are labeled via the labeling function $L : S \to 2^{\Pi}$ that defines what atomic propositions are true at each location. Furthermore, the vehicle is associated with a capacity $c \in \mathbb{N}$, and it can be shared among customers (demands) as long as the capacity is not exceeded.

Example 1. *We consider a small road network modeled as a WTS shown in Fig. 2. Its states are labeled with atomic propositions from the set* $\Pi = \{\mathsf{A}, \ldots, \mathsf{H}, \mathsf{shopping\ mall}\}$. *The capacity of the vehicle is* $c = 4$.

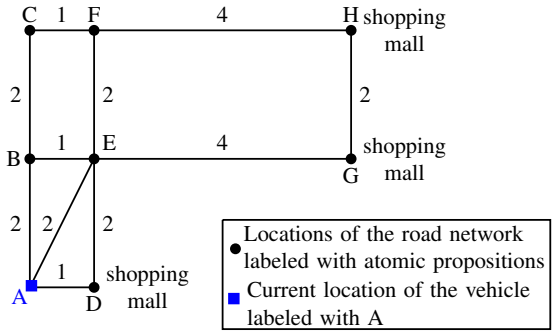


Fig. 2: The figure shows an example of a WTS. The black nodes represent location of interest, i.e., the state of the WTS, and are labeled with atomic propositions. For example, the propositions $\{\mathsf{H}, \mathsf{shopping\ mall}\}$ hold at the upper right node. The motion of the vehicle is captured by the edges (the transitions of the WTS) with weights denoting travel times. In this example, all roads are bi-directional with the same weight, i.e., $(s, s') \in R \Rightarrow (s', s) \in R$ and $W(s, s') = W(s', s)$. For simplicity, we do not show edges' orientations, and self-loop transitions $(s, s) \in R$.

B. Specification

The rich transportation requests are given as a (possibly infinite) set of demands $Demands = \{D_1, D_2, \ldots\}$ that appear gradually, at times $t_{D_1} \leq t_{D_2} \leq \ldots$, where $t_{D_i} \in \mathbb{R}_0$, for all $i \geq 1$. Each demand has five components: a pick-up location, a task, a deadline, a capacity needed, and a priority. All tasks define finite time behaviours. Thus, they can be expressed using syntactically co-safe LTL (scLTL), a fragment of LTL. Since each task specifies desired behavior only within a finite time window, the syntactically co-safe fragment of LTL is expressive enough, yet easier to handle in comparison to full LTL. Formally, for all $i \geq 1$, the demand $D_i = (s_{pick,i}, \varphi_i, T_i, c_i, p_i)$, where

- $s_{pick,i} \in S$ is the pick-up location
- φ_i is an scLTL formula specifying the task translated into a finite deterministic nonblocking automaton $\mathcal{A}_i$;
- $T_i \in \mathbb{R}_0$ is the deadline;
- $c_i \in \mathbb{N}$ is the needed capacity and
- $p_i \in \mathbb{N}$ is the priority

Without loss of generality, we assume that the demands are not satisfied trivially, and are not satisfied by only visiting the pick-up location. The higher the p_i is, the more important the D_i is.

C. Problem statement

Our goal is to find a plan for the vehicle that consists of the vehicle's route in the road network and a schedule of pick-ups and drop-offs. The capacity of the vehicle should never be exceeded, all tasks should be satisfied, and even though it might not be possible to satisfy all of them within the desired deadlines, the delays in servicing the tasks should be minimized while taking into account the demand priorities. In this section, we formalize this goal after introducing some necessary definitions.

Consider a WTS $\mathcal{T}$, a trace $\tau = s_{1:\infty}$ of $\mathcal{T}$ with a time sequence $\mathbb{T}(\tau) = t_{1:\infty}$, an infinite set of demands $Demands$, and a *pick-up decision function* $Pick : Demands \to \mathbb{N} \cup \{\infty\}$, where $Pick(D_i)$ indicates that demand D_i is picked up at time $t_{Pick(D_i)}$, and satisfies $s_{Pick(D_i)} = s_{pick,i}$ (the demand is picked up at the right location) and $t_{Pick(D_i)} \geq t_{D_i}$ (the demand is picked up after it arrived). $Pick(D_i) = \infty$ indicates that the demand D_i is never picked up.

Definition 5 (Active demand and demand in progress)**.** *A demand is active if it has arrived, but the desired task has not been achieved, yet. Formally, demand* D_i *is* active *on the trace* τ *at time* $t \in \mathbb{R}$ *if and only if* $t_{D_i} \leq t$ *and for all* $t_k \leq t$ *it holds that* $\tau_{Pick(D_i):k}$ *is not a good prefix w.r.t.* φ_i *according to Def. 3.* [1]

An active demand may or may have not been picked up. If it has been picked up, i.e. if $t_{Pick(D_i)} < t$, *we call it* in progress.

On the other hand, the demand is completed *if* $t_{D_i} \leq t_{Pick(D_i)} \leq t$ *and there exists* $t_k \leq t$ *such that* $\tau_{Pick(D_i):k}$ *is a good prefix w.r.t.* φ_i.

Definition 5 implies that we evaluate the satisfaction of ϕ_i by the trace of the vehicle only from the moment that D_i is picked up. Let $\mathcal{D}activ_{\tau,Pick}(t)$ and $\mathcal{D}inprog_{\tau,Pick}(t)$ denote the set of all demands that are active and in progress at time t, respectively. For simplicity, we use $\mathcal{D}activ(t)$ ($\mathcal{D}inprog(t)$) to denote $\mathcal{D}activ_{\tau,Pick}(t)$ ($\mathcal{D}inprog_{\tau,Pick}(t)$) whenever τ and $Pick$ are clear from the context.

Definition 6 (Plan)**.** *A plan is a tuple* $(\tau, Pick)$ *given by a trace* τ *of* $\mathcal{T}$ *and pick-up decision function* $Pick$ *that satisfy the following properties:*

- *Each demand eventually becomes completed. Formally, for all* $i \in \mathbb{N}$, $\tau_{Pick(D_i):\infty} \models \varphi_i$.
- *The capacity of the vehicle is never exceeded. Formally, for all* $j \in \mathbb{N}$, *it holds that* $\sum_{D_i \in \mathcal{D}inprog(t_j)} c_i \leq c$.

Remark 1. *It might seem that a plan should include also a drop-off decision. However, since we assume that the drop-off happens when the demand gets completed, the drop-offs can be fully reconstructed from a trace and a pick-up function.*

We have now formalized the notion of a plan that yields completion of all demands while never exceeding the vehicle's capacity. Our goal is, however, to find an optimal plan in situations where all tasks cannot be serviced within their respective deadlines and a decision has to be made on which

[1] If $t_{Pick(D_i)} > t_k$, then $\tau_{Pick(D_i):k}$ is empty and cannot be a good prefix of demand φ_i.

demands will be delayed and how much. We define the degree to which a plan meets a given set of demands based on the demand priorities and the delays in servicing the tasks. The degree's value is time-varying, because demands arrive sequentially over time are unknown prior to deployment. At time t, it depends only on the task execution durations, deadlines and priorities of the active demands, i.e., the demands that have arrived, but have not been completed, yet. Our aim is then to find a plan that maximizes this degree of satisfaction at all times. To summarize, let us assume that we have an appropriately defined *cost function* $J(\tau, Pick, j)$, which expresses whether, and when the demands are met. We will show several examples of such cost function below. The problem we tackle has a receding horizon nature and can be summarized as:

Problem 1 (Optimal continuation planning problem)**.** Given *a road network modeled as a WTS* $\mathcal{T}$ *and an infinite set of gradually arriving demands, repeatedly* find *an optimal continuation of the already executed plan. Formally,* given *a trace prefix* $\tau_{1:j} = \tau_{1:(j-1)} \cdot s_j$*, a pick-up decision function* $Pick$*, and a set of active demands* $\mathcal{D}activ(t_j)$ *at time* $t_j \in \mathbb{R}_0$*,* find *an optimal continuation, i.e. a trace suffix* $\tau^\star_{j:\infty} = s_j \cdot \tau^\star_{j+1:\infty}$ *and a consistent pick-up decision function* $Pick^\star : \mathcal{D}activ(t_j) \to \mathbb{N}$ *with the property* $Pick^\star(D_i) < j \implies Pick^\star(D_i) = Pick(D_i)$ *that minimize the cost* $J(\tau_{1:(j-1)} \cdot \tau^\star_{j:\infty}, Pick^\star, j)$*.*

To complete the problem formulation, let us define the task delay and three different instances of cost function J.

Definition 7 (Task delay)**.** *The* time of service *of task* φ_i *is* $t_{\varphi_i} = t_k$*, such that* $\tau_{Pick(D_i):k}$ *is the minimal good prefix of* $\tau_{Pick(D_i):\infty}$ *w.r.t.* φ_i *according to Def. 3. The* task execution duration *is* $d_i = (t_{\varphi_i} - t_{D_i})$*; and the* task delay *is then* $\Delta_i = d_i - T_i = (t_{\varphi_i} - t_{D_i}) - T_i$*.*

A negative task delay indicates that φ_i has been serviced within the deadline T_i after its arrival. In contrast, a positive task delay indicates that φ_i was serviced, but delayed by Δ_i.

1) Highest-priority-first cost function: An optimal plan must service the largest subset $\mathcal{D}_{\max} \subseteq \mathcal{D}activ(t_j)$ of active demands with highest priority by their deadlines at each time $t_j \in \mathbb{T}(\tau)$. The *highest-priority-first cost* function J_h is:

$$J_h(\tau, Pick, j) = \sum_{D_i \in \mathcal{D}activ(t_j)} \big(\Delta_i + M|\mathcal{D}activ(t_j)|^{p_i} \cdot I(i)\big), \tag{1}$$

where $M > \max_{D_i \in \mathcal{D}activ(t_j)} \Delta_i$ and $I(i) = \begin{cases} 0 & \text{if } \Delta_i \le 0 \\ 1 & \text{if } \Delta_i > 0 \end{cases}$.

This way, for each demand $D_i \in \mathcal{D}activ(t_j)$ it holds that

$$\sum_{\substack{D_{i'} \in \mathcal{D}activ(t_j) \\ \text{s.t. } p_{i'} < p_i}} |\mathcal{D}activ(t_j)|^{p_{i'}} < |\mathcal{D}activ(t_j)|^{p_i}.$$

since there are at most $|\mathcal{D}activ(t_j)| - 1$ active demands $D_{i'}$ with priorities $p_{i'} \le p_i - 1$. Hence, the high-priority demands are strictly prioritized. Among two traces and pick-up decision functions, the one that services the highest-priority demand yields lower cost. In other words $\forall (\tau_1, Pick_1), (\tau_2, Pick_2) \in \{(\tau, Pick) \mid \tau_{1:j} = \tau^\star_{1:j}, Pick^\star(D_i) < j \implies Pick(D_i) = Pick^\star(D_i)\}$ it holds that $J_h(\tau_1, Pick_1, j) < J_h(\tau_2, Pick_2, j) \iff \exists D_i \in \mathcal{D}activ_{\tau_1}(t_j)\ s.t.\ \Delta_i \le 0 \wedge \nexists D_{i'} \in \mathcal{D}activ_{\tau_2}(t_j)\ s.t.\ \Delta'_i \le 0 \wedge p' \ge p$.

2) Bottleneck-delay cost function: The optimal plan must "fairly" service the active demands. For all times $t_j \in \mathbb{T}(\tau)$, we want to minimize the largest task delay weighted by its priority. The *bottleneck-delay cost* function J_b is:

$$J_b(\tau, Pick, j) = \max_{D_i \in \mathcal{D}activ(t_j)} \Delta_i \cdot p_i \tag{2}$$

3) Cumulative-delay cost function: The optimal plan must minimize the weighted sum of delays for all of active demands with their priorities as weights. The *cumulative-delay cost* function J_c is

$$J_c(\tau, Pick, j) = \sum_{D_i \in \mathcal{D}activ(t_j)} \Delta_i \cdot p_i \tag{3}$$

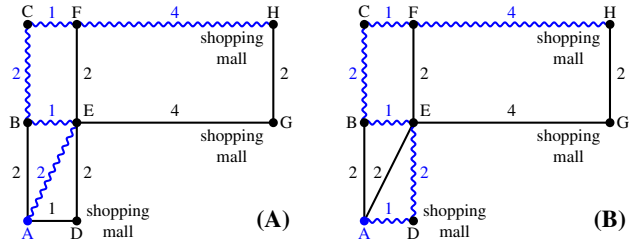
Planning is non-preemptive using the highest-priority-first cost function. Demand with high priorities will be preferred and not interrupted irrespective of the delays incurred by lower priority requests. Bottleneck- and cumulative-delay cost functions enable a compromise. No demand is delayed indefinitely using the bottleneck-delay cost while cumulative delay aims to optimize overall service efficiency over all demands.

Example 2. *We revisit Example 1 and consider a simple scenario with two demands from Table I, and a vehicle with capacity* $c = 4$ *deployed in the road network shown in Fig. 2. Both demands arrive at the same time* $t_1 = t_{D_1} = t_{D_2} = 0$*, and at the same pick-up location* A*. Thus, demands* D_1 *and* D_2 *are active at time* t_1*. The first demand* φ_1 *requires visiting locations* E*,* B*, and* H *in order (possibly concurrently) within the deadline* $T_1 = 10$*. It's priority is* $p_1 = 7$*, and requires* $c_1 = 1$ *capacity. The second demand* φ_2 *requires visiting a* shopping mall*, i.e., one of* D, G, H*. It's priority is* $p_2 = 1$*, and requires* $c_2 = 2$ *capacity. Two solutions paths, good trace prefixes, are shown in Fig. 3. Both demands are picked up immediately,* $Pick(D_1) = Pick(D_2) = 1$*, since* $c_1 + c_2 = 3 < c = 4$*. The solution in Fig. 3.(A) visits* E, B, H *in order and services both demands by* $t_6 = 10$*. On the other hand, the solution in Fig. 3.(B) visits* D, E, B, H *in order. At time* $t_2 = 1$*, demand* D_2 *is done and becomes inactive. Demand* D_1 *is completed by* $t_7 = 11$*. We summarize the durations, delays, and cost functions for the two cases (A) and (B) in Table II. For the highest-priority-first cost function* J_h *in (1), solution in (A) is optimal as long as* M *is chosen to be larger than the longest delay, i.e. 7, which is expected because the priority* p_1 *of* D_1 *is higher than priority* p_2 *of* D_2*. For the cumulative-delay cost function* J_c *in (3), solution in (B) is optimal. The cost strikes a balance between short delays for higher-priority demands and avoiding the excessive delays of lower-priority ones. Both solutions are optimal under the bottleneck-delay cost* J_b *in (2). Increasing the priority of demand* D_1 *to* $p_1 = 10$ *would shift the optimal solution to (A) for all three cost functions. A lower priority of* $p_1 < 7$ *would make (B) optimal for the bottleneck-delay cost as well, but not highest-priority-first.*

The notation used throughout this paper is summarized in Table III.

	Pick-up	scLTL formula	Arrival	Deadline	Priority	Capacity
D_1	$s_{pick,1} = A$	$\varphi_1 = \mathcal{F}(\mathsf{E} \wedge \mathcal{F}(\mathsf{B} \wedge \mathcal{F}\mathsf{H}))$	$t_{D_1} = 0$	$T_1 = 10$	$p_1 = 7$	$c_1 = 1$
D_2	$s_{pick,2} = A$	$\varphi_2 = \mathcal{F}$ shopping mall	$t_{D_2} = 0$	$T_2 = 3$	$p_2 = 1$	$c_2 = 2$

TABLE I: Example from Fig. 2 with two demands.



● Locations of the road network labeled with atomic propositions
Trace prefix from A to H

Fig. 3: The figure shows two example solution paths (good trace prefixes) for the WTS in Fig. 2 and demands in Table I. Both start at the current location A and end at location H.

	(A)	(B)
$t_{\varphi_1} = d_1$	10	11
$t_{\varphi_2} = d_2$	10	1
Δ_1	0	1
Δ_2	7	-2
$J_h(\tau, 1, 1)$	$(0+0)+(7+2^1M)$ $= 2M+7$	$(1+2^7M)+(-2+0)$ $= 128M-2$
$J_b(\tau, 1, 1)$	$\max(0 \cdot 7, 7 \cdot 1) = 7$	$\max(1 \cdot 7, (-2) \cdot 1) = 7$
$J_c(\tau, 1, 1)$	$0 \cdot 7 + 7 \cdot 1 = 7$	$1 \cdot 7 - 2 \cdot 1 = 5$

TABLE II: Duration, delay and cost values for the solution paths (good trace prefixes) in Fig. 3.(A) and 3.(B).

TABLE III: Symbols table.

$\mathcal{T}$	transition system representing road network
Π	set of atomic propositions and associated labeling map
L	labeling map
$Demands$	set of all demands
D_i	$i-$th demand
$\mathcal{D}activ(t)$, $\mathcal{D}inprog(t)$	sets of active and in-progress demands at time t
$s_{pick,i}$	pick-up location of demand D_i
φ_i	sc-LTL task specification of demand D_i
T_i, c_i, p_i	deadline, capacity, and priority of demand D_i
$t_{D_i}, t_{Pick(D_i)}$	time of arrival and pickup of demand D_i
$t_{\varphi_i}, d_i, \Delta_i$	time of completion, task execution duration, and delay of D_i
τ,	trace in $\mathcal{T}$
$Pick$	$Pick(D_i)$ indicates D_i is picked up at time $t_{Pick(D_i)}$
$\mathfrak{s}, \mathfrak{q}$	current state in $\mathcal{T}$ and $\mathcal{A}$

IV. MILP-Based Solution

A. Approach overview

To solve Problem 1, we initialize $j := 1$, $\tau_{1:j} := s_{init}$, $Pick(D_i) = \infty$ for all $D_i \in Demands$, and $t_1 := 0$ and we iteratively

(i) find a solution $(\tau^\star_{j:\infty}, Pick^\star)$ to Problem 1;

(ii) perform the pick-up ${Pick^\star}^{-1}(j)$ if any, and execute the first transition (s_j, s_{j+1}) of $\tau^\star_{j:\infty}$ and;

(iii) repeat the procedure starting with step (i) with new trace prefix $\tau_{1:j} := \tau_{1:j} \cdot s_{j+1}$, updated $Pick := Pick^\star$, time $t_j := t_j + W(s_j, s_{j+1})$ and index $j := j+1$.

With $j \to \infty$, we obtain that $(\tau_{1:j}, Pick)$ are the solution to Problem 1.

Remark 2. *In our framework, re-computation can take place only at states. In other words, transitions once started cannot be preempted, and future plans can be modified only after they are completed.*

B. Product automaton

We propose an automata- and optimization-based solution to Problem 1. At time t_j when a new demand arrives or one is completed, we construct a finite weighted product automaton $\mathcal{P}(j)$. The product model combines the motion model of the vehicle WTS $\mathcal{T}$ with the finite state automata obtained from all currently active demands $\mathcal{D}activ(t_j)$. The initial state of the product captures the current location of the vehicle in the road network, and the progress towards satisfaction of the active demands. The weights of $\mathcal{P}(j)$ are computed based on durations (weights) of $\mathcal{T}$ and the priorities of active demands $\mathcal{D}activ(t_j)$. Thus, we translate the problem of finding an optimal path for the vehicle (trace of $\mathcal{T}$) with respect to cost J into the problem of finding an optimal run in $\mathcal{P}(j)$. The general idea of handling infeasibility via weighed product construction has been introduced earlier, e.g., in [22]. The key elements of our approach are the definition of states, design of the weight function and evaluation of the initial state that enables us to do so.

Consider a WTS $\mathcal{T}$, its trace prefix $\tau_{1:j}$ executed up to time t_j, a pick-up decision function $Pick$, a set of active demands $\mathcal{D}activ(t_j) = \{D_{i_1}, \ldots, D_{i_{m_j}}\} \neq \emptyset$, and a cost function J. Note that given $Pick$, we know the demands in progress $\mathcal{D}inprog(t_k)$ for all $1 \leq k < j$. Let us use $\mathcal{D}active(t_0) = \emptyset$ to denote that before the start of the vehicles run in the road network, there are no active demands.

We translate each formula φ_i of demand $D_i \in \mathcal{D}(t_j)$ into a nonblocking deterministic finite automaton $\mathcal{A}_i = (Q_i, q_{init,i}, 2^\Pi, \delta_i, F_i)$. Since φ_i is an scLTL formula, we assume that accepting states are terminal, i.e., $\delta_i(q, \sigma) \in F_i$ for all $q \in F_i$ and $\sigma \in 2^\Pi$.

Let $\mathfrak{s}_j$ denote the current state of $\mathcal{T}$ at time t_j, i.e. the last state of $\tau_{1:j} = \tau_{1:(j-1)} \cdot \mathfrak{s}_j$ and let $\mathfrak{q}_{i,j}$ denote the current state of the automaton $\mathcal{A}_i$ at t_j, for all $D_i \in \mathcal{D}(t_j)$. Note that at time $t_j = t_1 = 0$, $\mathfrak{s}_1 = s_{init}$. However, $\mathfrak{q}_{i,1}$ can be either $q_{init,i}$ if demand $D_i \in \mathcal{D}(t_1)$ is not picked up at t_1, or $\delta_i(q_{init,i}, L(\mathfrak{s}_1))$ if it is. Similarly, the set of possible current states for a demand $D_i \in \mathcal{D}active(t_j) \setminus \mathcal{D}inprog(t_{j-1})$ is

$$\mathfrak{Q}_{i,j} = \begin{cases} \{q_{init,i}, \delta_i(q_{init,i}, L(\mathfrak{s}_j))\} & \text{if } \mathfrak{s}_j = s_{pick,i} \\ \{q_{init,i}\} & \text{otherwise.} \end{cases} \quad (4)$$

Note that for all $D_i \in \mathcal{D}active(t_j) \cap \mathcal{D}inprog(t_{j-1})$, the current state is $\mathfrak{q}_{i,j} = \delta_i(\mathfrak{q}_{i,j-1}, L(\mathfrak{s}_j))$.

Definition 8 (Weighted product automaton at t_j)**.** *The weighted product automaton $\mathcal{P}(j) = \mathcal{T} \otimes \mathcal{A}_{i_1} \otimes \ldots \otimes \mathcal{A}_{i_{m_j}}$ at time t_j is a tuple $(Q_\mathcal{P}, Q_{init,\mathcal{P}}, \delta_\mathcal{P}, F_\mathcal{P}, E_{init,\mathcal{P}}, W_\mathcal{P})$, where*

- $Q_\mathcal{P} = \{(s, q_1, \ldots, q_{m_j}, \mathcal{D}) \in S \times (\prod_{\ell=1}^{m_j} Q_{i_\ell}) \times \{2^{\mathcal{D}activ(t_j)} \mid \sum_{i \in \mathcal{D}} c_i < c\}$;

- $Q_{init,\mathcal{P}} = \{(\mathfrak{s}_j, \mathfrak{q}_{i_1,j}, \dots, \mathfrak{q}_{i_{m_j},j}, (\mathcal{D}activ(t_j) \cap \mathcal{D}inprog(t_{j-1})) \cup \mathcal{D}picked)\}$, *where* $\mathcal{D}picked \subseteq \{D_i \in \mathcal{D}activ(t_j) \mid \mathfrak{s}_j = s_{pick,i} \wedge q_i \notin F_i\}$, *and*

$$\mathfrak{q}_{i,j} = \begin{cases} \delta_i(q_{init,i}, L(\mathfrak{s}_j)) & \text{if } D_i \in \mathcal{D}picked \\ \delta_i(\mathfrak{q}_{i,j-1}, L(\mathfrak{s}_j)) & \text{if } D_i \in \mathcal{D}(t_j) \cap \mathcal{D}inprog(t_{j-1}) \\ q_{init,i} & \text{otherwise;} \end{cases}$$

- $\delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}}$ *is a transition relation;* $\big((s, q_1, \dots, q_{m_j}, \mathcal{D}), (s', q'_1, \dots, q'_{m_j}, \mathcal{D}')\big) \in \delta_{\mathcal{P}}$ *if*
  - $(s, s') \in R$;
  - $(q_i, L(s'), q'_i) \in \delta_i, \forall D_i \in \mathcal{D} \cap \mathcal{D}'$ *or* $q_i = q'_i$;
  - $\mathcal{D}' = (\mathcal{D} \setminus \mathcal{D}dropped) \cup \mathcal{D}picked$, *where* $\mathcal{D}dropped = \{D_i \in \mathcal{D}activ(t_j) \mid q_i \in F_i\}$, $\mathcal{D}picked \subseteq \{D_i \in \mathcal{D}activ(t_j) \mid s' = s_{pick,i} \wedge q_i \notin F_i\}$; *and*
  - $\sum_{D_i \in \mathcal{D}'} c_i \leq c$
- $F_{\mathcal{P}} = \{(s, q_{i_1}, \dots, q_{i_{m_j}}, \emptyset) \mid q_i \in F_i, \forall i \in \{i_1, \dots, i_{m_j}\}\}$;
- $E_{init,\mathcal{P}} = (\nu_1, \dots, \nu_{m_j})$, *where* $\forall \ell \in \{1, \dots, m_j\}$, $\nu_\ell = t_j - t_{D_{i_\ell}}$ *is the evaluation associated with the initial state; and*
- $W_{\mathcal{P}} : \delta_{\mathcal{P}} \to \mathbb{R}_0^{m_j}$ *is the weight function, where* $W_{\mathcal{P}}\big((s, q_1, \dots, q_{m_j}, \mathcal{D}), (s', q'_1, \dots, q'_{m_j}, \mathcal{D}')\big) = (\nu_1, \dots, \nu_{m_j})$, *such that* $\forall \ell \in \langle m_j \rangle$,

$$\nu_i = \begin{cases} W(s, s') & \text{if } q_{i_\ell} \notin F_{i_\ell} \\ 0 & \text{otherwise.} \end{cases}$$

The product $\mathcal{P}(j)$ is a finite state automaton, whose states are augmented with information about picked-up demands, evaluation of the initial state based on changes in the set of active demands, and transition weights that capture delays in completion of demands. We omit the alphabet of $\mathcal{P}(j)$ since it bears no significance.

A finite run of $\mathcal{P}(j)$ is a sequence $\rho = v_j v_{j+1} \dots v_n$, where $\forall j \leq k \leq n$, $v_k = (s_k, q_{1,k}, \dots, q_{m_j,k}, \mathcal{D}_k)$, $v_j \in Q_{init,\mathcal{P}}$, and $\forall j \leq k < n$ it holds that $(v_k, v_{k+1}) \in \delta_{\mathcal{P}}$. It is accepting if $v_n \in F_{\mathcal{P}}$. Note that in the accepting states, the set of demands in progress $\mathcal{D}_k$ is empty. From the construction of $\mathcal{P}(j)$, it follows that ρ projects onto a trace fragment $\text{proj}_1(\rho) = \tau_{j:n} = \mathfrak{s}_j \mathfrak{s}_{j+1} \dots s_n$ of $\mathcal{T}$ and a finite run fragment $\text{proj}_{\ell+1}(\rho) = \varrho_{i_\ell,j:n} = \mathfrak{q}_{i_\ell,j} q_{i_\ell,j+1} \dots q_{i_\ell,n}$ of $\mathcal{A}_{i_\ell}$, for all $\ell \in \langle m_j \rangle$. If ρ is accepting then $\tau_{1:(j-1)} \cdot \tau_{j:n}$ is a good trace prefix with respect to φ_i, i.e. each $q_{i_\ell,n}$ is an accepting state of $\mathcal{A}_{i_\ell}$ for all $D_{i_\ell} \in \mathcal{D}activ(t_j)$. Vice versa, for each good trace prefix $\tau_{1:n}$ of $\mathcal{T}$ that produces a word accepted by $\mathcal{A}_i$ via an accepting run ϱ_i for each demand $D_i \in \mathcal{D}activ(t_j)$, there exists a finite accepting run of $\mathcal{P}$ that projects onto a suffix $\tau_{j:n}$ of $\tau_{1:n}$ that starts in $\mathfrak{s}(t_j)$ and onto a suffix $\varrho_{i,j:n}$ of each ϱ_i that starts in $\mathfrak{q}_{i,j}$ and ends in an accepting state. The construction of the product automaton also implies the following: assuming that demand D_i is in progress, a conflicting demand will not be picked up unless D_i is completed.

Remark 3. *Similar to [27] or [42], we construct $\mathcal{P}$ such that all its states are reachable from the initial one, which reduces the memory footprint considerably. Furthermore, before constructing $\mathcal{P}$, we reduce the transition system $\mathcal{T}$ by algorithmically removing states that are not significant with respect to the satisfaction of the active demands analogously to [43]. For instance, consider scLTL formulas $\varphi_1 = \mathcal{F}(\mathsf{E} \wedge \mathcal{F}(\mathsf{B} \wedge \mathcal{F}\mathsf{H}))$ and $\varphi_2 = \mathcal{F}\,\mathsf{shopping\ mall}$. Clearly, the satisfaction and violation of the formulas is not influenced by a visit to any state of $\mathcal{T}$ that is not labeled with* E, B, *and* H, *and* $\mathsf{shopping\ mall}$, *respectively. We can thus replace these states in $\mathcal{T}$ with transitions representing the minimum duration paths between states that contribute to the satisfaction of active demands.*

C. General MILP formulation

The three cost functions J defined in Sec. III-C are particular choices of goals for optimal planning. In this section, we propose a general solution to Problem 1 that can handle a large class of cost functions $J(\tau, Pick, j)$ that depend on (a) the execution durations $\mathbf{d}(t_j) = (d_{i_1}, \dots, d_{i_{m_j}})$, (b) the deadlines $\mathbf{T}(t_j) = (T_{i_1}, \dots, T_{i_{m_j}})$, and (c) the priorities $\mathbf{p}(t_j) = (p_{i_1}, \dots, p_{i_{m_j}})$ of the active demands $\mathcal{D}activ(t_j)$ at time t_j:

$$\begin{aligned} &J(\tau, Pick, j) : \mathbb{R}_0^{m_j} \times \mathbb{R}_0^{m_j} \times \mathbb{N}^{m_j} \to \mathbb{R}_0 \\ &J(\tau, Pick, j) = f\left(\mathbf{d}(t_j), \mathbf{T}(t_j), \mathbf{p}(t_j)\right), \end{aligned} \tag{5}$$

where $m_j = |\mathcal{D}activ(t_j)|$.

Assumption 1. *We assume that the cost function is the maximum of linear functions in the delays and delay violations. Formally, we assume*

$$f(\mathbf{d}, \mathbf{T}, \mathbf{p}) = \max_{\ell \in \langle m_j \rangle} \{C_\Delta^T(\mathbf{T}, \mathbf{p}, \ell)\Delta + C_I^T(\mathbf{T}, \mathbf{p}, \ell) I\}, \tag{6}$$

where $\Delta = \mathbf{d} - \mathbf{T}$, $I = \mathbf{1}_{\Delta \geq 0}$, $C_\Delta : \mathbb{R}_0^{m_j} \times \mathbb{N}^{m_j} \times \langle m_j \rangle \to \mathbb{R}_0^{m_j}$, and $C_I : \mathbb{R}_0^{m_j} \times \mathbb{N}^{m_j} \times \langle m_j \rangle \to \mathbb{R}_0^{m_j}$ are the stacked delay vector, the delay violation indicator vector, and the delay and indicator gain functions, respectively.

Each of the three cost functions listed above – the highest-priority-first, the bottleneck-delay, and the cumulative delay one, satisfies Assumption 1, which is derived as follows:

1) Highest-priority-first cost function: Let $C_\Delta(\mathbf{T}, \mathbf{p}, \ell) = \mathbf{1}$ and $C_I(\mathbf{T}, \mathbf{p}, \ell) = M \sum_{\ell' \in \langle m_j \rangle} |\mathcal{D}activ(t_j)|^{p_{\ell'}} \zeta_{\ell'}$ for all $\ell \in \langle m_j \rangle$. Then we have

$$\begin{aligned} f(\mathbf{d}, \mathbf{T}, \mathbf{p}) &= \max_{\ell \in \langle m_j \rangle} \left\{ \mathbf{1}^T \Delta + M \sum_{\ell' \in \langle m_j \rangle} |\mathcal{D}activ(t_j)|^{p_{\ell'}} \zeta_{\ell'}^T I \right\} \\ &= \max_{\ell \in \langle m_j \rangle} \left\{ \sum_{\ell' \in \langle m_j \rangle} (\Delta_{\ell'} + M |\mathcal{D}activ(t_j)|^{p_{\ell'}} I(\ell')) \right\} \\ &= \sum_{\ell \in \langle m_j \rangle} \Delta_\ell + M \sum_{\ell \in \langle m_j \rangle} |\mathcal{D}activ(t_j)|^{p_\ell} I(\ell), \end{aligned}$$

where the max operator vanishes because the enclosed sum does not depend on the optimizing variable ℓ.

2) Bottleneck-delay cost function: Let $C_\Delta(\mathbf{T}, \mathbf{p}, \ell) = p_\ell \zeta_\ell$ and $C_I(\mathbf{T}, \mathbf{p}, \ell) = 0$ for all $\ell \in \langle m_j \rangle$.

$$f(\mathbf{d}, \mathbf{T}, \mathbf{p}) = \max_{\ell \in \langle m_j \rangle} \left\{ p_\ell \zeta_\ell^T \Delta + 0^T I \right\} = \max_{\ell \in \langle m_j \rangle} \{p_\ell \Delta_\ell\}.$$

3) Cumulative-delay cost function: Let $C_\Delta(\mathbf{T},\mathbf{p},\ell)=\mathbf{p}$ and $C_I(\mathbf{T},\mathbf{p},\ell)=0$ for all $\ell\in\langle m_j\rangle$.

$$f(\mathbf{d},\mathbf{T},\mathbf{p})=\max_{\ell\in\langle m_j\rangle}\left\{\mathbf{p}^T\Delta+0^T I\right\}=\sum_{\ell=1}^{m_j}p_\ell\Delta_\ell,$$

where the max operator vanishes because the enclosed quantity does not depend on the optimizing variable ℓ.

We formulate Problem 1 for the class of cost functions given by (6) as a Mixed Integer Linear Programming (MILP) problem. The formulation is based on the on the flow MILP formulation of the shortest path problem between a source and a sink. Thus, we add a source state $q^{\bowtie}_{init,\mathcal{P}}$ and a sink state $q^{\bowtie}_{final,\mathcal{P}}$ to the product automaton $\mathcal{P}(j)$. The source is connected only to the initial states $Q_{init,\mathcal{P}}$ by transitions of weight $E_{init,\mathcal{P}}$. The sink receives transitions of weight 0 from all final states of $\mathcal{P}(j)$, i.e., the set of weighted transitions $\{(q,q^{\bowtie}_{final,\mathcal{P}},0_{m_j})\mid q\in F_{\mathcal{P}}\}$ are added to $\mathcal{P}(j)$.

The MILP formulation is as follows

$$\min\ \lambda \tag{7}$$

Subject to:

$$\lambda\geq C_\Delta(\mathbf{T},\mathbf{p},\ell)\Delta+C_I(\mathbf{T},\mathbf{p},\ell)I,\forall\ell\in\langle m_j\rangle \tag{8}$$

$$\xi_{v,v'}\in\{0,1\},\forall(v,v')\in\delta_{\mathcal{P}} \tag{9}$$

$$\sum_{(v,v')\in\delta_{\mathcal{P}}}\xi_{v,v'}-\sum_{(v',v)\in\delta_{\mathcal{P}}}\xi_{v',v}=\begin{cases}1, & v=q^{\bowtie}_{init,\mathcal{P}}\\ -1, & v=q^{\bowtie}_{final,\mathcal{P}}\\ 0, & \text{otherwise}\end{cases} \tag{10}$$

$$\sum_{(v,v')\in\delta_{\mathcal{P}}}\xi_{v,v'}\leq 1,\forall v\in Q_{\mathcal{P}} \tag{11}$$

$$\Delta=\sum_{(v,v')\in\delta_{\mathcal{P}}}\xi_{v,v'}W_{\mathcal{P}}(v,v') \tag{12}$$

$$I\in\{0,1\}^{m_j} \tag{13}$$

$$-M(\mathbf{1}_{m_j}-I)\leq\Delta\leq M\cdot I \tag{14}$$

where $M\in\mathbb{N}$ is a large enough constant.

In the formulation, the max operator in the cost function is translated into the set of linear constraints in (8), and an auxiliary variable λ is used as a cost function instead. We associate with each transition of $\mathcal{P}(j)$ a binary variable $\xi_{v,v'}$ (9) that indicates whether (v,v') is present in the solution. The solution can only be composed of a single simple (acyclic) path from the source $q^{\bowtie}_{init,\mathcal{P}}$ to the sink $q^{\bowtie}_{final,\mathcal{P}}$, and is enforced by the flow conservation constraint in (10), and the bound on the number of nodes' successors in (11). Thus, the delay vector is simply the weighted sum of all transition variables (12). The binary vector I of delay violation (13) indicates if the demands' deadlines have been met. It is captured by the constraints in (14) using the *big M* trick, where M is a constant larger than the maximum absolute value of the delay (which can be computed from $\mathcal{P}(j)$ as the longest simple path from $Q_{init,P}$ to a state $q\in F_{\mathcal{P}}$).

After solving the MILP problem, we can recover the optimal accepting run $\rho^\star=v_j\dots v_n$ as follows: starting from the initial state $v_j\in Q_{init,\mathcal{P}}$ with $\xi_{q^{\bowtie}_{init,\mathcal{P}},v_j}=1$, we iteratively compute the next state v_{k+1}, $k\in\{j,\dots,n-1\}$, satisfying $\xi_{v_k,v_{k+1}}=1$ until a final state $v_n\in F_{\mathcal{P}}$ is reached.

Note that $\rho^\star$ defines a prefix $\tau^\star_{j:n}=\mathfrak{s}_j s_{j+1}\dots s_n$ of the desired optimal trace suffix $\tau^\star_{j:\infty}$ via projection onto $\mathcal{T}$ (i.e., first component of product automaton states); in fact any trace suffix $\tau_{j:\infty}$ with the prefix $\tau^\star_{j:n}$ is a solution to Problem 1. Hence, we take $\tau^\star_{j:\infty}=\mathfrak{s}_j s_{j+1}\dots s_n\cdot s_n^\omega=\tau^\star_{j:n}\cdot s_n^\omega$. The optimal pick-up decision function $Pick^\star_{j:\infty}$ is also obtained from $\rho^\star$: assuming that $\forall j\leq k\leq n$, $v_k=(s_k,q_{1,k},\dots,q_{m_j,k},\mathcal{D}_k)$, we set $Pick^\star_{j:\infty}(D_i)=\ell$, where $D_i\notin\mathcal{D}_{\ell-1}$ and $D_i\in\mathcal{D}_\ell$ for all $D_i\in\mathcal{D}activ(t_j)\setminus\mathcal{D}inprog(t_{j-1})$.

Theorem 1. *The suggested trace suffix $\tau^\star_{j:\infty}$ and the pick-up decision function $Pick^\star_{j:\infty}$ are the optimal solution to Problem 1 at time t_j for the trace prefix $\tau^\star_{1:j}=s_{init}\dots\mathfrak{s}_j$, pick-up decision function $Pick$, and the set of active demands $\mathcal{D}activ(t_j)\neq\emptyset$.*

Proof. The proof follows directly from the discussion above. Namely, it holds that $\tau^\star_{j:n}$ is a minimal good prefix from the fact that $p_n\in F_{\mathcal{P}}$ and from the construction of $\mathcal{P}$. Furthermore, $\mathbf{d}(t_j)=\sum_{v,v'\in\rho^\star}W_{\mathcal{P}}(v,v')$ for the trace $\tau_{1:j-1}\cdot\tau^\star_{1:j}$ at time t_j. Hence, $f(\mathbf{d}(t_j),\mathbf{T}(t_j),\mathbf{p}(t_j))=J(\tau_{1:j-1}\cdot\tau^\star_{1:j},j)$ at time t_j. □

Since we solve Problem 1 iteratively online, the trace suffix can change influencing also servicing of demands that are already in progress; such a change, however, occurs only in a case when a new demand arrives and still ensures that the demands in progress get completed before new demands are picked up. In other words, frequent switching of solutions will not have undesired impact on overall performance.

D. MILP formulation for three particular cost functions

Next, we instantiate the MILP formulation for the three trace cost functions described in Sec. III.

1) Highest-priority-first cost function: Take $C_\Delta(\mathbf{T},\mathbf{p},\ell)=\mathbf{1}$ and $C_I(\mathbf{T},\mathbf{p},\ell)=M\sum_{\ell'\in\langle m_j\rangle}|\mathcal{D}activ(t_j)|^{p_{\ell'}}\zeta_{\ell'}$ for all $\ell\in\langle m_j\rangle$, which results in the MILP:

$$\min\ \sum_{\ell=1}^{m_j}\Delta_\ell+M\sum_{\ell=1}^{m_j}|\mathcal{D}activ(t_j)|^{p_\ell}I(\ell) \tag{15}$$

Subject to: (9), (10), (11), (12), (13), (14),

where all the constrains to encode the max operator are dropped because all weights C_I are equal and all C_Δ are one. For that reason, λ, and constraint (8) are not needed; instead we can just optimize the cost function directly.

2) Bottleneck-delay cost function: Setting $C_\Delta(\mathbf{T},\mathbf{p},\ell)=p_\ell\zeta_\ell$ and $C_I(\mathbf{T},\mathbf{p},\ell)=0$ for all $\ell\in\langle m_j\rangle$ results in the MILP:

$$\min\ \lambda \tag{16}$$

Subject to: (9), (10), (11), (12)

$$\lambda\geq p_\ell\Delta_\ell,\forall\ell\in\langle m_j\rangle \tag{17}$$

where the indicator variables I and corresponding constraints (14) were dropped because the weight functions C_I are zero for all demands. Rewriting constraint (8) explicitly using $C_\Delta(\mathbf{T},\mathbf{p},\ell)=p_\ell\zeta_\ell$ and $C_I(\mathbf{T},\mathbf{p},\ell)=0$, results in (17).

3) Cumulative-delay cost function: Setting $C_\Delta(\mathbf{T}, \mathbf{p}, \ell) = \mathbf{p}$ and $C_I(\mathbf{T}, \mathbf{p}, \ell) = 0$ for all $\ell \in \langle m_j \rangle$ results in the LP:

$$\min \sum_{(v,v') \in \delta_{\mathcal{P}}} \xi_{v,v'} \left(\mathbf{p}^T W_{\mathcal{P}}(v, v') \right) \tag{18}$$
$$\text{Subject to: (10) and } \xi_{v,v'} \geq 0$$

Since the weight functions C_I are zero for all demands, we discarded the indicator variables I and corresponding constraints (14). Furthermore, all weight functions C_Δ are equal, we can drop the max operator constraints, and the cost function becomes the trace function, i.e., the sum of all demands' delays weighted by their priorities. Lastly, the problem can be solved as an LP because the flow constraints are totally unimodular [44]. The successors bound constraints in (11) can also be dropped, because minimizing the cumulative delay cost ensures that the optimal solution is a simple path [44].

E. Execution procedure at time t_j

Given the optimal accepting run of $\rho^\star = v_j, \ldots, v_n$ of $\mathcal{P}(j)$, the optimal trace suffix $\tau^\star_{j:\infty} = \mathfrak{s}_j \ldots s_n \cdot s_n^\omega$ of $\mathcal{T}$, and the consistent pick-up decision function $Pick^\star_{j:\infty}$ at time t_j, the system execution proceeds as follows:

- The transition $(\mathfrak{s}_j, s_{j+1})$ is taken in $\mathcal{T}$ and the current state of $\mathcal{T}$ at time t_{j+1} becomes $\mathfrak{s}_{j+1} = s_{j+1}$;
- The transition $(\text{proj}_{i_\ell}(v_j), L(\mathfrak{s}_j), \text{proj}_{i_\ell}(v_{j+1})) \in \delta_{i_\ell}$ is taken in $\mathcal{A}_{i_\ell}$, for all $\ell \in \langle m_j \rangle$; If $\text{proj}_{i_\ell}(v_{j+1}) \notin F_{i_\ell}$ then include D_{i_ℓ} in $\mathcal{D}activ(t_{j+1})$ and the current state of $\mathcal{A}_{i_\ell}$ at time t_{j+1} becomes $\mathfrak{q}_{i_\ell, j+1} = \text{proj}_{i_\ell}(v_{j+1})$.

At time t_{j+1}

- Include all newly arrived demands D_i, such that $t_j < t_{D_i} \leq t_{j+1}$ in $\mathcal{D}activ(t_{j+1})$;
- Update the set of demands in progress $\mathcal{D}inprog(t_{j+1}) = \mathcal{D}activ(t_{j+1}) \cap \{D_i \mid {Pick^\star_{j:\infty}}^{-1}(D_i) \leq j+1\}$
- If $\mathcal{D}activ(t_{j+1}) = \{D_{i'_1}, \ldots D_{i_{m'_{j+1}}}\} \neq \emptyset \neq \mathcal{D}activ(t_j)$ then compute $\mathcal{P}(j+1) = \mathcal{T} \otimes \mathcal{A}_{i'_1} \otimes \ldots \otimes \mathcal{A}_{i'_{m_{j+1}}}$ at time t_{j+1} as in Def. 8, and compute $\rho^\star$ for $\mathcal{P}(j+1)$ and the optimal suffix $\tau^\star_{j+1:\infty}$ through the MILP problem from (7)–(13); else
- If $\mathcal{D}activ(t_{j+1}) = \emptyset$ then choose $\tau^\star_{j+1:\infty} = \mathfrak{s}^\omega_{j+1}$;
- Repeat the execution procedure at time t_{j+1} with $\tau^\star_{j+1:\infty}, Pick^\star_{j+1:\infty}$.

Remark 4. *Note that the way we define and approach the problem allows to apply the proposed solution in a straightforward way also to dynamic road networks, where the weight function $W : R \times \mathbb{R}_0 \to \mathbb{R}_+$ is time-dependent and captures e.g., different levels of congestion during different daytimes.*

Complexity. The complexity of our approach is driven by the two components: product model construction and MILP computation. The complexity of the product construction is given by the sizes of the factor models, i.e., $O(|R| \cdot \prod_{\ell=1}^{m_j} |Q_{\mathcal{A}_{i_\ell}}|)$. The construction of the MILP encoding is linear in the size of the product model $O(|Q_{\mathcal{P}}|)$. Solving MILPs is know to be an NP-complete problem. Nevertheless, modern solvers (e.g. Gurobi [45]) are able to solve large problem instances with millions of decision variables. Furthermore, the problems solved on the product models are shortest path problems with some additional decisions roughly of the order of $|\mathcal{D}|$. Since the shortest path problem (encoded as a network flow problem) is totally unimodular, the LP solution is actually also the ILP solution. Our hypothesis for our setting is that the MILP in our case is very close to a shortest path problem (i.e., "close" to being unimodular). Hence, the solver converges fast to an optimal solution. This contrasts with Traveling Salesman Problems which have a very different problem structure.

V. Linear-Time Graph Search-Based Solution

Section IV considered the Problem 1 for a large class of cost functions J. The proposed approach constructs a product automata and computes the solution path using a MILP encoding whenever new demands arrive. For a subclass of cost functions, we introduce a graph-search method to compute the optimal path $\rho^\star$ with linear complexity in the size of $\mathcal{P}(j)$.

Assumption 2 (Translation Monotonic)**.** *Let $\mathcal{D}activ(t_j) = \{D_{i_1}, \ldots, D_{m_j}\}$, and $\mathbf{d}(t_j)$, $\mathbf{d}'(t_j) \in \mathbb{R}_0^{m_j}$ with*

$$f\big(\mathbf{d}(t_j), \mathbf{T}(t_j), \mathbf{p}(t_j)\big) \leq f\big(\mathbf{d}'(t_j), \mathbf{T}(t_j), \mathbf{p}(t_j)\big).$$

We assume that for all $\boldsymbol{\nu} \in \mathbb{R}^{m_j}$

$$f\big(\mathbf{d}(t_j) + \boldsymbol{\nu}, \mathbf{T}(t_j), \mathbf{p}(t_j)\big) \leq f\big(\mathbf{d}'(t_j) + \boldsymbol{\nu}, \mathbf{T}(t_j), \mathbf{p}(t_j)\big).$$

Algorithm 1: `OptAcc`$(\mathcal{P}(j), f)$

Input: Product automaton $\mathcal{P}(j)$ and cost function f satisfying Assumption 2
Output: Optimal accepting run $\rho^\star$

forall $p \in Q_{\mathcal{P}}$ **do**
  $e(v) \leftarrow \infty$; $\text{pred}(v) \leftarrow None$
  Add v to *Unvisited*;
$e(q_{init,\mathcal{P}}) \leftarrow E_{init,\mathcal{P}}$, $\forall q_{init,\mathcal{P}} \in Q_{init,\mathcal{P}}$
while *Unvisited* $\neq \emptyset$ **do**
  $v \leftarrow \text{argmin}_{v' \in Unvisited} f(e(v'), \mathbf{T}(t_j), \mathbf{p}(t_j))$
  remove v from *Unvisited*
  forall $(v, v') \in \delta_{\mathcal{P}}$ **do**
    $cand \leftarrow f\big(e(v) + W_{\mathcal{P}}(v, v'), \mathbf{T}(t_j), \mathbf{p}(t_j)\big)$
    if $cand \leq f(e(v'), \mathbf{T}(t_j), \mathbf{p}(t_j))$ **then**
      $e(v') \leftarrow e(v) + W_{\mathcal{P}}(v, v')$
      $\text{pred}(v') \leftarrow v$
$v_f \leftarrow \text{argmin}_{v' \in F_{\mathcal{P}}} f(e(v'), \mathbf{T}(t_j), \mathbf{p}(t_j))$
$v \leftarrow v_f$; $\rho^\star \leftarrow v_f$
while $p \notin Q_{init,\mathcal{P}}$ **do**
  $v \leftarrow \text{pred}(p)$; $\rho^\star \leftarrow v \cdot \rho^\star$;
return $\rho^\star$

Trivially, Assumption 2 holds for the cumulative-delay cost function J_c in (3). For the other two costs, highest-priority-first in (1) and the bottleneck-delay in (2), the assumption does not hold. A modified version of the highest-priority-first cost function for which Assumption 2 is true is

$$J'_h(\tau, Pick, j) = \sum_{D_i \in \mathcal{D}activ(t_j)} |\mathcal{D}activ(t_j)|^{p_i} \cdot \Delta_i.$$

The cost J'_h captures the quantity of the delay rather than its presence and absence.

A. Graph search

Assumption 2 allows us to find the desired optimal accepting run in $\mathcal{P}(j)$ using a linear-time graph algorithm based on the well-known Dijkstra shortest path search instead of solving a more computationally demanding MILP problem.

The goal is to compute the path in $\mathcal{P}(j)$ with minimum weight that ends in an accepting state $F_{\mathcal{P}}(j)$. Similar to Dijkstra's algorithm, our method $\texttt{OptAcc}(\mathcal{P}(j), f)$ shown in Algorithm 1 constructs a spanning forest from the initial states in a greedy way. However, the weights in our case are vector-values, and the branch cost is evaluated using the cost function f. The initialization is step adds all states to the queue of unvisited states, and sets their predecessors to $None$ and branch cost to ∞ except for the initial states $Q_{init,\mathcal{P}}$ (lines 2 and 4). Next, the algorithm iterates until all states are visited. In each iteration, it find and removes from the unvisited state v with the smallest branch cost $f(e(v), \mathbf{T}(t_j), \mathbf{p}(t_j))$ (line 6). For all states v' that v can reach, the algorithm checks whether the cost to v' can be decreased using the path to v (line 10). In case this holds, the cost is updated and the predecessor of v' becomes v (lines 10-12). After the all states have been visited, the algorithm returns the run in $\mathcal{P}$ that ends in an accepting state and has minimum weight (lines 15–16).

Once we compute $\rho^\star$ using $\texttt{OptAcc}(\mathcal{P}(j), f)$ the remainder of the approach to Problem 1 does not differ from the solution in Sec. IV; we project $\rho^\star$ onto a trace fragment $\tau^\star_{j:n} = \mathfrak{s}_j s_{j+1} \dots s_n$ of the desired optimal trace suffix $\tau^\star_{j:\infty} = \tau^\star_{j:n} \cdot s_n^\omega$ and also use it to reconstruct $Pick^\star_{j:\infty}$.

B. Characterization

Assumption 2 is sufficient for Algorithm 1 to compute an optimal run. It does not provide a simple way to check cost functions. In this section, we characterize the class of cost functions induced by Assumption 2. We denote the set of all translation monotonic functions by $\mathcal{J}_{\leq}$.

Theorem 2 (Translation Monotonic Functions)**.** *If $f \in \mathcal{J}_{\leq}$ is a continuous function, then there exist $g : \mathbb{R} \to \mathbb{R}$ and $a \in \mathbb{R}^n$ such that $f(x) = g(a^T x)$.*

The proof is in Appendix A.

VI. Simulation Results

In this section, we present a series of simulation results to demonstrate the scalability of our approach in a large, realistic case study and compare the performance of the three proposed cost functions. We perform simulations on the middle part of Manhattan (see Fig. 1). The road network has 184 nodes and edges weighted by time-varying travel travel duration estimates obtained from real taxi driving data. For details about the dataset see the work by [46]. We randomly generate demands from 10 scLTL patterns that arrive at times according to a homogeneous Poisson process and have deadlines determined by an upper bound weighted by a uniformly distributed scalar weight. The simulation is ran over 24 hour time window. Since we focus on scalability, we present macro-scale analyses of the results in terms of: i) the three cost functions, and ii) runtime performance, rather than showing route plans.

The algorithms were implemented in Python2 based on *LOMAP* [27] and networkx [47]. The LTL to automata translation within LOMAP is done using the *Spot* suite [48]. Gurobi was used to solve the MILPs associated with the bottleneck delay and highest priority cost functions [45]. For the cumulative delay cost function, we used the graph search method described in Alg. 1.

Vehicle: The vehicle's capacity is limited to 3.

Network: The transition system corresponds to a 125-block area in mid-Manhattan, and has 184 nodes and 355 weighted transitions. The transition weights are taken from the taxi dataset, and are available in one hour increments. We used travel duration estimates corresponding to a weekday. We assume that the transition system is strongly connected such that the vehicle can visit any location from any other location in the network.

Demands: We randomly generated demands from a set of template scLTL formulas:

$$
\begin{aligned}
\phi_1(p_0,p_1,p_2) &= \mathcal{F}p_0 \wedge \mathcal{F}p_1 \wedge \mathcal{F}p_2\\
\phi_2(p_0,p_1,p_2) &= \mathcal{F}(p_0 \wedge \mathcal{F}(p_1 \wedge \mathcal{F}p_2))\\
\phi_3(p_0,p_1,p_2) &= \mathcal{F}p_0 \wedge \mathcal{F}(p_1 \vee p_2)\\
\phi_4(p_0,p_1,p_2) &= \mathcal{F}(p_0 \wedge \mathcal{F}(p_1 \vee p_2))\\
\phi_5(p_0,p_1,p_2,n_0) &= \mathcal{F}p_0 \wedge \mathcal{F}p_1 \wedge \neg n_0 \,\mathcal{U}\, p_2\\
\phi_6(p_0,p_1,p_2,n_0) &= \mathcal{F}(p_0 \wedge \mathcal{F}p_1 \wedge \neg n_0 \,\mathcal{U}\, p_2)\\
\phi_7(p_0,\dots,p_5) &= \mathcal{F}(p_0 \vee p_1) \wedge \mathcal{F}(p_2 \vee p_3) \wedge \mathcal{F}(p_4 \vee p_5)\\
\phi_8(p_0,\dots,p_5) &= \mathcal{F}((p_0 \vee p_1) \wedge \mathcal{F}((p_2 \vee p_3) \wedge \mathcal{F}(p_4 \vee p_5)))\\
\phi_9(p_0,\dots,p_6) &= \mathcal{F}p_0 \wedge \mathcal{F}(p_1 \wedge \mathcal{F}(p_2 \vee p_3))\\
&\quad \wedge \mathcal{F}(p_4 \vee (p_5 \wedge \mathcal{F}p_6))\\
\phi_{10}(p_0,\dots,p_6) &= \mathcal{F}(p_0 \wedge \mathcal{F}(p_1 \wedge \mathcal{F}p_2 \wedge \mathcal{F}p_3))\\
&\quad \vee \mathcal{F}(p_1 \wedge \mathcal{F}(p_4 \vee p_5) \wedge \mathcal{F}p_6)
\end{aligned}
$$

where the parameters p_0, …, p_6, and n_0 are labels associated with states of the transition system. The labels p_0, …, p_6 hold true at a single location, respectively, define the locations the vehicles needs to visit to satisfy the specification. Label n_0 holds true at multiple locations, and determines a circular area that the vehicle should avoid before visiting p_2. Locations are generated such that the demands can be satisfied, i.e., the graphs remains strongly connected after removing the states labeled with n_0. Additionally, we generate random pick-up locations, distinct from all other locations with labels from the associated demand.

Next we define the demands' arrival times and deadlines. The demands' arrival is modeled as a homogeneous Poisson process with rate ϖ, i.e., the inter-arrival times are i.i.d. exponentially distributed with mean $1/\varpi$. The deadlines are computed as αT_i, where $\alpha \sim Unif(1,2)$ is an uniformly distributed scalar weight, and T_i is an upper bound on the demand i's service time. Formally, we have

$$T_i = \frac{1}{|S|}\sum_{s\in S} \overline{W}(s, s_{pick,i}) + |\overline{S}_i| * \max_{s,s'\in\overline{S}_i} \overline{W}(s,s'), \tag{19}$$

where the first term is mean duration to arrive at the pickup location $s_{pick,i}$ of demand i, the second term is the an upper bound on visiting all locations of interest $\overline{S}_i$ =

$\{s_{pick,i}, p_0, \ldots, p_6\}$ for demand i, and $\overline{W}(s, s')$ is the minimum duration of a path between s and s' in $\mathcal{T}$. For this computation, we used the estimated travel durations at 9am from [46]. The rate of the Poisson process ϖ was chosen such that within 19 hours we have a mean of 30 demand arrivals, i.e. $\varpi = 19 \cdot 3600/30$ $[sec]$. Within the 24 hour simulation window, 36 demands were generated as shown in Fig. 4.

Finally, the demands' priorities and capacities are randomly generated uniformly between 1 and 5, and between 1 and 2, respectively. The finite state automata for $\phi_1, \ldots, \phi_{10}$ are computed before the simulation, and have 8, 4, 4, 3, 8, 5, 8, 4, 18, 18 nodes, and 20, 7, 8, 5, 20, 12, 20, 7, 57, 55 transitions.

	J_c	J_b	J_h	J_{EDF}
Routing runtime	0.101 ± 0.121 $[0.038, 0.841]$	4.133 ± 10.428 $[0.053, 48.670]$	0.369 ± 0.531 $[0.042, 2.242]$	0.082 ± 0.044 $[0.038, 0.200]$
Build $\mathcal{P}$ runtime	0.098 ± 0.116 $[0.037, 0.803]$	1.413 ± 3.137 $[0.050, 13.996]$	0.217 ± 0.246 $[0.039, 1.073]$	0.081 ± 0.043 $[0.038, 0.199]$
$\lvert S \rvert$ in $\mathcal{T}$	8.2 ± 4.01 $[4, 30]$	14.02 ± 7.38 $[5, 37]$	11.38 ± 4.95 $[4, 24]$	8.15 ± 4.54 $[4, 20]$
$\lvert R \rvert$ in $\mathcal{T}$	61.7 ± 62.4 $[12, 400]$	195.3 ± 176.5 $[20, 698]$	125.0 ± 104.6 $[12, 387]$	45.4 ± 28.6 $[12, 110]$
$\lvert Q_{\mathcal{P}} \rvert$ in $\mathcal{P}$	140 ± 428 $[10, 2711]$	3671 ± 8165 $[14, 32419]$	491 ± 765 $[14, 2909]$	34 ± 23 $[10, 116]$
$\lvert \delta_{\mathcal{P}} \rvert$ in $\mathcal{P}$	1611 ± 5800 $[30, 36342]$	73628 ± 173443 $[52, 728792]$	7569 ± 14072 $[42, 58168]$	180 ± 150 $[30, 896]$
$\lvert Demands \rvert$	1.184 ± 0.481 $[1, 3]$	2.083 ± 1.037 $[1, 4]$	1.809 ± 0.914 $[1, 4]$	1.417 ± 0.666 $[1, 3]$
Δ	-7123 ± 3907 $[-15101, -1278]$	-4931 ± 3912 $[-15015, 1031]$	-5123 ± 6146 $[-14248, 18402]$	-6355 ± 4336 $[-15101, 1768]$

TABLE IV: Performance results are shown from 24-hour simulations of mid-Manhattan for the three cost functions: cumulative delay J_c, bottleneck delay J_b, and highest priority first J_h, with capacity constraints and pick-up locations. Each entry of the table shows the mean and standard deviation of each metric, and below its range.

A. Results and discussion

We report results for the routing with the three cost functions: cumulative delay cost J_c, bottleneck delay cost J_b, and highest priority first cost J_h, with capacity and pick-up location constraints. We also consider a heuristic baseline method, earliest deadline first, denoted by J_{EDF}. A summary of the algorithms' performance is shown in Table IV. Note that the reported size of $\mathcal{T}$ there takes into account the reduction of redundant states, see Remark 3.

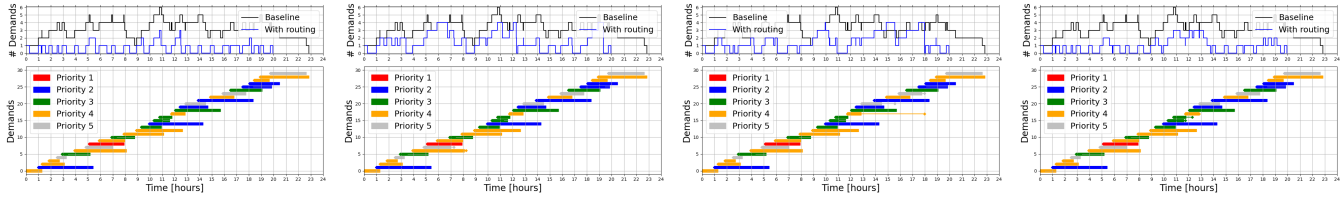
Runtime performance. In Table IV, we observe that the construction of the weighted product automata $\mathcal{P}$ dominates the execution time for the cumulative delay cost function, while it is significantly smaller fraction of the total routing time in the case of the bottleneck delay and highest priority first cost functions. This is not surprising, since in the cumulative case the computation of the optimal policy is an LP, which can be solved with Dijkstra's algorithm. The other two cases require binary variables, and thus can be in worst case exponential in the number of binary variables. *Remarkably, the MILP solver is able to handle problems with up to a few hundred thousand binary variables within a couple of minutes.* We hypothesize that the performance is due to the structure of our problems, which are very close shortest path problems with combinatorial choices that may depend mostly on the number of demands.

Cost performance. The routing algorithms satisfied the generated request in such a way that only a small number demands were active at any given time, see Table. IV. The delays on the last row of the table are comparable for J_c, J_b and J_h. A detailed account of the performance with respect to cost of the algorithm is in Fig. 4, Fig. 5 and Fig. 6. The triplet of plots in each of these figures are obtained with the use of the same, randomly generated, sequence of demands. *The magnitude of all three cost functions depends on the number of active demands*. Fig. 5 shows this dependency for each level of cost throughout the simulation of 24-hours.

To show the differences between the three cost functions, we computed all three of them on the simulation trajectory obtained by the each routing scheme, see Fig. 6. In the graphs for J_c shown in Fig. 6a, observe that as the routing is performed to benefit all demands on average, the ones with maximum priority are not the first to be satisfied. This is suggested by the rising bottleneck cost. The graphs for J_b shown in Fig. 6b again highlight the difference between the routing strategies in the cases where falls or jumps in the cumulative delay cost are not mirrored by the bottleneck cost, which masks lower-priority weighted delays. In this regard, J_b might be less affected than J_c by dynamics of demand arrival, and is more stable. The case of J_h is the most striking in the difference of routing decisions as shown Fig. 6c. The rising costs of J_c and J_b are due to an interesting masking behavior of J_h. If a demand cannot be satisfied by its deadline regardless of priority, then it is essentially a constant term in the highest priority first cost function. Thus, such a demand must wait until all other demands that can be satisfied by their deadline are done.

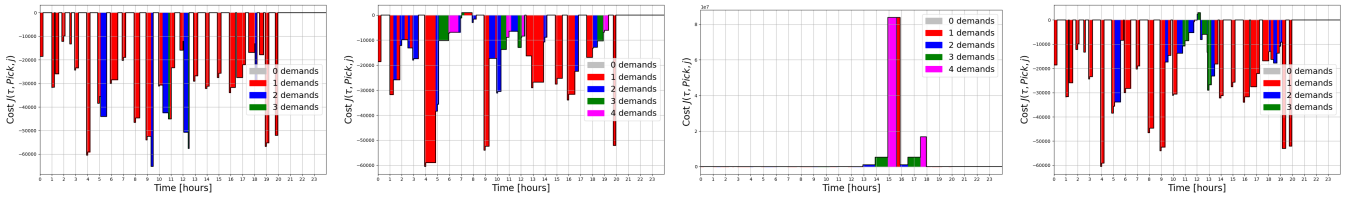
These observations are corroborated by looking at the arrival, deadline, and service time of each demand given in Fig. 4. The graphs highlight the way demands are preempted based on the cost functions. For J_c smooths out large delays in favor of average optima Fig. 4a. However, lower priority demands may wait longer, until their cost becomes critical, i.e., their associated term in the cost becomes large enough to influence the routing decision. In the case of J_b in Fig. 4b we can observe how lower priority demands are preempted due to bottleneck masking. The J_h case has fewer deadline violations. Some of the demands are preempted for a long time due to the capacity constraints that limit the number of demands in progress at any given time (in this case, 2). Lastly, observe in Fig. 4c the cause of the phenomenon mentioned above for the highest priority first cost function regarding demands that cannot be satisfied by their deadline. The demands that need to wait until all others are done, induce the rising costs J_c and J_b in the graph from Fig. 6c.

Baseline. The earliest deadline first method J_{EDF} selects the demand with the smallest deadline among the ones in progress. If there are no demands in progress, then it selects a demand among the active ones. Unsurprisingly, J_{EDF} is the fastest method. It also has reasonable cost performance. However, compared to J_c it is not able to fulfill all demands by their deadlines. The mean cost performance is between J_c



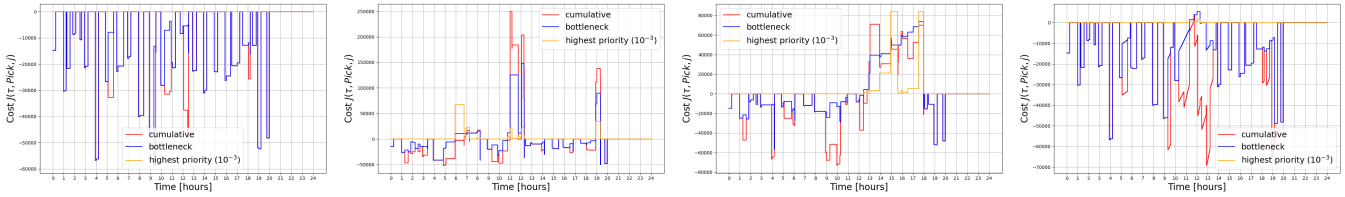
(a) J_c (b) J_b (c) J_h (d) J_{EDF}

Fig. 4: The demands' timelines over a 24-hour simulation are shown for each of the four cost functions with capacity constraints and pick-up locations. In the lower graph, for each demand, the time interval between its arrival and desired deadline is marked with a solid box, while the time interval to the actual deadline is marked with a line of the same color with diamond endpoints. The demands' colors indicate their priority between 1 and 5. The upper graph shows the number of active demands in the ideal case disregarding routing in black, and in the actual case obtained by executing the routing algorithm with the corresponding cost function in blue.



(a) J_c (b) J_b (c) J_h (d) J_{EDF}

Fig. 5: The performance of the algorithms with respect to the costs J_c, J_b, J_h and J_{EDF} are shown in relation to the number of demands. The cost levels are maintained between routing calls. The colors of the column represent the number of demands associated with the routing call and cost value.



(a) J_c (b) J_b (c) J_h (d) J_{EDF}

Fig. 6: The cross cost performance of the routing algorithm using cost functions J_c, J_b, J_h and J_{EDF} are shown in Fig. 6a, Fig. 6b, Fig. 6c, and Fig. 6d, respectively. In each subfigure, the evaluation of the three cost functions J_c, J_b and J_h is plotted at each routing decision. Only the associated cost function is used for routing. The cost graphs change whenever the routing procedure is executed.

and J_b, but with larger deviations. This may be due to the greedy nature of the procedure. The nature of J_h makes it mostly incompatible with J_{EDF} which does not account for demand priorities.

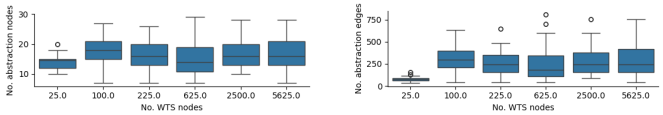
Discussion. The main insights of the empirical analysis is that (1) the cumulative delay cost function provides a good balance of runtime and cost performance and is appropriate for most cases; (2) the bottleneck delay cost function induces a less spread out distribution of delays and is appropriate for cases where we care about equitable demand satisfaction (albeit with a runtime performance penalty); (3) the highest priority first cost function disregard overall cost performance and is appropriate for cases where priorities must be enforced (e.g., due to premium fees); and (4) the earliest deadline first method may induce larger delays and is appropriate for cases where runtime performance is more important than cost performance.

B. Scalability Analysis

In this section, we investigate the scalability of the proposed routing algorithms. For the analysis, we randomly generated 20 square $N \times N$ grid networks for each $N \in \{5, 10, 15, 25, 50, 75\}$. The number of states in each network is N^2, while the set of transitions is random and induces a strongly connected network. The transition durations are drawn uniformly from $\{1, \ldots, 20\}$. For each network, we generated a set of demands the same in Sec. VI. with scLTL templates $\{\phi_1, \ldots, \phi_{10}\}$. The number of demands is uniformly drawn from $\{2, 3, 4\}$. The vehicle capacity is 3.

Results and discussion: As in the previous Manhattan case study, we report results for the three cost functions: cumulative

delay cost J_c, bottleneck delay cost J_b, and highest priority first cost J_h, with and without pick-up location constraints.



(a) States in the abstraction. (b) Transitions in the abstraction.

Fig. 7: Size of abstraction versus number of states in the network.

The network size does not significantly impact the scalability of the routing algorithm. This is expected, since the routing algorithm first constructs an abstraction of the network using minimum cost paths between location of interest required for the active demands, see Remark 3. Fig. 7a and Fig. 7b show that the number of states and transitions do not increase significantly with the network size. However, the increased computation to construct the abstractions of road networks can be observed in Fig. 8a and Fig. 8b corresponding to the product automaton construction and overall routing procedure runtimes, respectively.

The more significant driver of scalability is the number of active demands. Fig. 8c and Fig. 8d show the product automaton construction and overall routing procedure runtimes for each number of active demands in our analysis, respectively. The increase in runtime is again expected, since the size of the product automaton used for routing (Definition 8) increases exponentially with the number of active demands. Fig. 8 shows that routing without pick-up constraints take less time. This follows from the additional locations of interests that need to be considered in the network abstraction. The product automata construction is the same between the three algorithms (with and without pick-ups), and the runtime does not vary significantly across them as seen in Fig. 8a and Fig. 8c.

VII. Conclusions and Future Work

We introduced a route planning framework that enables to consider a variety of complex transportation requests by using syntactically co-safe Linear Temporal Logic (scLTL) and to systematically handle situations, when all of these requests cannot be met simultaneously by a set deadline. scLTL is rigorous, yet it builds on temporal operators, whose semantics are aligned with the intuitive meaning – operators next, eventually, until express that an event or a propositions should hold in the next time step, eventually, or until another event or proposition becomes true. Nesting and combining these operators gives scLTL a rich expressive power. For example, one may express that certain sub-tasks should happen in a sequence (a customer should visit location A, then B, then C expressed as $\mathcal{F}(\mathsf{A}\wedge\mathcal{F}(\mathsf{B}\wedge\mathcal{F}\mathsf{C}))$), that a set of sub-tasks should happen in an arbitrary order (packages should be dropped of at locations A, B, and C, expressed as $\mathcal{F}\mathsf{A}\wedge\mathcal{F}\mathsf{B}\wedge\mathcal{F}\mathsf{C}$), that a sub-task can be met in various way (a customer should visit any shopping mall, expressed as $\mathcal{F}\mathsf{mall}$, where multiple locations in the map are labeled with the atomic proposition mall). The route planning framework can handle any combination of such requests. Furthermore, each demand in scLTL is associated with an arrival time, a pick-up location, a deadline, a priority, and the needed capacity of the servicing vehicle, e.g., if the demand is issued by a group of customers that wish to travel together.

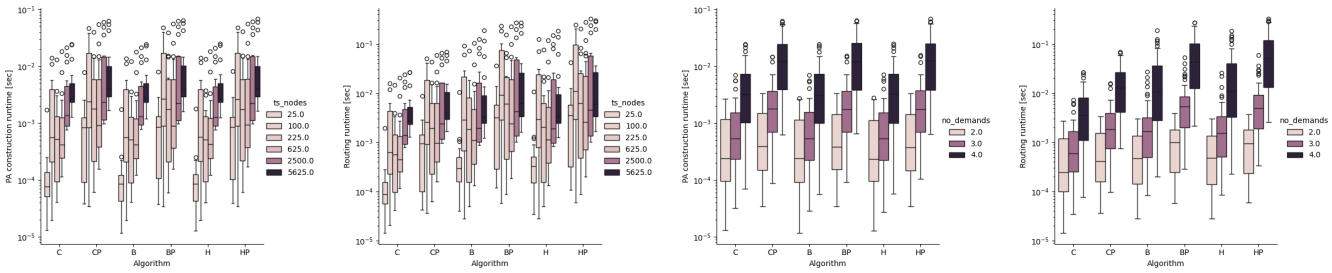
We modeled the motion of the vehicle in a road network as a finite Weighted Transition System (WTS). The locations of interests represent the WTS states connected by road network segments and are correspondingly labeled with atomic propositions that hold true therein (such as with the atomic proposition mall mentioned above). The transitions of the WTS model the capability of the vehicle to traverse the road segments. Estimated travel durations along roads are captured as time-varying weights on the WTS' transitions.

The goal was to plan the path of the vehicle, i.e., a sequence of WTS states, along with customer pick-up and drop-off decisions, to satisfy all the demands with the least possible overall cost that balances demands' delays and priorities. We proposed three different costs that a user of our framework may be interested in – the highest-priority-first cost, the bottleneck-delay cost, and the cumulative-delay cost. We introduced a MILP-based approach that computes minimal cost plans. For a special subclass of criteria, we derived an efficient, graph-search algorithm. As the requests arrive gradually, the plan is periodically recomputed and it holds that at any time instant, it is provably the optimal one among all the plans that have the same history. We showed the performance of the proposed framework in large scale simulations involving a large road network corresponding to part of mid-Manhattan with time-varying travel duration estimates, and large number of demands. In comparison to the state of the art, this work is to our best knowledge the first one to address route planning under complex, gradually arriving, prioritized and possibly mutually unsatisfiable transportation requests.

Our future work involves incorporating other aspects that will further generalize our framework: we aim to investigate extension to multi-agent vehicle routing problems, incorporation of time spent in the node of the network (e.g., corresponding to customers spending time in locations), timed temporal logic for request specification to allow for assignment of deadlines to sub-tasks, stochastic models of the road network to capture how the changing road network link capacity, including how our rooting affects congestion.

Acknowledgment

This work is supported in part by the Singapore MIT Alliance for Research and Technology (The Future of Urban Mobility project), by the NSF CNS-1446151 and IIS-1723995, the NSF Grant 1723943, and the Office of Naval Research (ONR) Grant N00014-18-1-2830. Toyota Research Institute ("TRI") provided funds to assist some of the authors with their research, but this article solely reflects the opinions and conclusions of its authors, and not TRI or any other Toyota entity.



(a) PA construction time (b) Overall routing time (c) PA construction time (d) Overall routing time

Fig. 8: Runtime performance for cumulative delay cost (C), bottleneck delay cost (B), and highest priority first cost (H). The cases with pick-up location constraints are marked with P. For each network size N^2, $N \in \{5, 10, 15, 25, 50, 75\}$, we generated 20 random networks with random weights and demands.

APPENDIX
PROOF OF THEOREM 2

First, we introduce a intermediate results before proceeding to the proof of Theorem 2. Note that if $f \in \mathcal{J}_{\leq}$, then $(f - f(0))$ is also in $\mathcal{J}_{\leq}$. Thus, without loss of generality, we assume that $f(0) = 0$ for all $f \in \mathcal{J}_{\leq}$ throughout the rest of the section.

Proposition 1. *If $f \in \mathcal{J}_{\leq}$, and $x_1, x_2 \in \mathbb{R}^n$ such that $f(x_1) \sim f(x_2)$, then $f(x_1 + w) \sim f(x_2 + w)$ for all $w \in \mathbb{R}^n$, where $\sim \in \{<, =\}$.*

Proof. Case $\sim$ is $=$. Let $w \in \mathbb{R}^n$. We have

$$f(x_1) = f(x_2) \equiv \begin{cases} f(x_1) \leq f(x_2) \\ f(x_1) \geq f(x_2) \end{cases}$$

$$\equiv \begin{cases} f(x_1 + w) \leq f(x_2 + w) \\ f(x_1 + w) \geq f(x_2 + w) \end{cases} \equiv f(x_1 + w) = f(x_2 + w)$$

Case $\sim$ is $<$. Assume there exists $w \in \mathbb{R}^n$ such that $f(x_1 + w) \geq f(x_2 + w)$. It follows that $f(x_1 + w + w') \geq f(x_2 + w + w')$ for all $w' \in \mathbb{R}^n$. If we set $w' = -w$, then we have $f(x_1) \geq f(x_2)$ which contradicts the hypothesis. □

Proposition 2. *If $f \in \mathcal{J}_{\leq}$ continuous, and $x_1, x_2 \in \mathbb{R}^n$ such that $f(x_1) \sim f(x_2)$, then $f(\alpha x_1) \sim f(\alpha x_2)$ for all $\alpha \in \mathbb{R}_{>0}$, where $\sim \in \{\leq, <, =\}$. For equality ($\sim$ is $=$), the property holds for any $\alpha \in \mathbb{R}$.*

Proof. Consider $\alpha = p \in \mathbb{Z}_+$. It follows by induction on p,

$$\begin{aligned} f((p+1)x_1) = f(x_1 + px_1) &\sim f(x_2 + px_1) \\ &\sim f(x_2 + px_2) = f(x_2 + px_2) \end{aligned}$$

where we used Prop. 1, and the induction hypothesis. The base case is given by Prop. 1.

Consider $\alpha = \frac{1}{q}$, $q \in \mathbb{Z}_+$. Denote by $\not\sim$ be negation of the inequality $\sim$. Assume $f(\alpha x_1) \not\sim f(\alpha x_2)$, it follows that $f(q\alpha x_1) \not\sim f(q\alpha x_2)$, or equivalently $f(x_1) \not\sim f(x_2)$, where we used the first case $\alpha \in \mathbb{Z}_+$ to obtain the implication. This leads to a contradiction with the hypothesis $f(x_1) \sim f(x_2)$.

The previous two cases imply that the proposition holds for $\alpha = \frac{p}{q} \in \mathbb{Q}_+$ rational. The full statement follows from the continuity of f.

Lastly, the case $\sim$ is $=$ follows from setting $w = \alpha \cdot (x_1 + x_2)$, $\alpha > 0$, i.e., $f(-\alpha x_1) = f(-\alpha x_2) \equiv f(\alpha x_2) = f(\alpha x_1)$, and the case for $\alpha > 0$ above. □

Proposition 3. *Let $f \in \mathcal{J}_{\leq}$ continuous, and $x_1, x_2 \in \mathbb{R}^n$. There exists $\lambda_1, \lambda_2 \in \mathbb{R}$ not both zero such that $f(\lambda_1 x_1) = f(\lambda_2 x_2)$.*

Proof. Assume that for all λ_1 and λ_2 not both zero we have $f(\lambda_1 x_1) \neq f(\lambda_2 x_2)$. Since f is continuous, it follows that the sets $I_{\lambda_1} = \{f(\lambda_1 x_1)\}_{\lambda_1}$, and $I_{\lambda_2} = \{f(\lambda_2 x_2)\}_{\lambda_2}$ are intervals. Thus, we have either $f(\lambda_1 x_1) < f(\lambda_2 x_2)$ or $f(\lambda_1 x_1) > f(\lambda_2 x_2)$ for all λ_1, λ_2. Without loss of generality, consider $f(\lambda_1 x_1) < f(\lambda_2 x_2)$. If we set $\lambda_1 = -1$, $\lambda_2 = 1$, and $w = x_1$, then $f(-x_1 + w) = f(0) = 0 < f(x_2 + w) = f(x_2 + x_1)$. If we set $\lambda_1 = 1$, $\lambda_2 = -1$, and $w = x_2$, then $f(x_1 + w) = f(x_1 + x_2) < f(-x_2 + w) = f(0) = 0$. We arrive at a contradiction. Thus, there exists λ_1 and λ_2, not both zero, such that $f(\lambda_1 x_1) = f(\lambda_2 x_2)$. □

Let $\{e_1, \ldots, e_n\}$ denote the standard basis of $\mathbb{R}^n$. Define α_{ij} such that $f(\alpha_{ji} e_i) = f(\alpha_{ij} e_j)$, for all $i \neq j$, $i, j \in \{1, \ldots, n\}$.

Proposition 4. *Let $f \in \mathcal{J}_{\leq}$ continuous. If $\alpha_{ji} = 0$, then $f(\alpha e_j) = 0$ for all $\alpha \in \mathbb{R}$ with $i, j \in \{1, \ldots, n\}$ and $\alpha_{ji} \in \mathbb{R}$.*

Proof. Let $\alpha_{ji} = 0$. By definition and Prop. 3, there exists $\alpha_{ij} \neq 0$ such that $f(\alpha_{ji} e_i) = f(\alpha_{ij} e_j) = f(0) = 0$. Thus, using Prop. 2 we have $f(\alpha e_j) = f(\alpha \frac{\alpha_{ji}}{\alpha_{ij}} e_i) = f(0) = 0$, where $\alpha \in \mathbb{R}$. □

Proposition 5. *Let $f \in \mathcal{F}$ continuous. If $f(\alpha e_i) = 0$ for all $\alpha \in \mathbb{R}$, then $f(x) = f(x - x_i e_i)$.*

Proof. By definition, we have $f(\alpha e_i) = f(0) \equiv f(x + \alpha e_i) = f(x)$, where we set $w = x \in \mathbb{R}^n$. Take $\alpha = -x_i$, and we obtain the result. □

Proof of Theorem 2. Let f be a translation monotonic function not identically zero. Without loss of generality due to Prop. 5, assume $f(\alpha e_1) \neq 0$. We show that $f(x)$ can be written as a function of the first component of x as follows

$$\begin{aligned} f(x) &= f(x_1 e_1 + \ldots + x_{n-1} e_{n-1} + x_n e_n) \\ &= f(x_1 e_1 + \ldots + x_{n-1} e_{n-1} + x_n a_n e_1) \end{aligned}$$

where $a_n = \frac{\alpha_{n1}}{\alpha_{1n}}$. If we repeat the process for all other $i \in \{2, \ldots, n-1\}$, then we arrive at

$$f(x) = f((a_1 x_1 + \ldots + a_n x_n) e_1) = f((a^T x) e_1)\,,$$

where $a_1 = 1$, and $a_i = \frac{\alpha_{i1}}{\alpha_{1i}}$.

We need to show that $\alpha_{1i} \neq 0$ for all $i \in \{2, \ldots, n\}$. Assume that $\alpha_{1i} = 0$, then by Prop. 4 we have $f(\alpha e_1) = 0$ for all α, which leads to a contradiction.

Lastly, we obtain the statement by defining $g(y) = f(ye_1)$. Thus, $f(x) = g(a^T x)$. □

REFERENCES

[1] A. Buchegger, K. Lassnig, S. Loigge, C. Mühlbacher, and G. Steinbauer, "An Autonomous Vehicle for Parcel Delivery in Urban Areas," in *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2961–2967.

[2] S. Tanaka, T. Senoo, and M. Ishikawa, "High-speed UAV Delivery System with Non-stop Parcel Handover Using High-speed Visual Control," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 4449–4455.

[3] D. N. Das, R. Sewani, J. Wang, and M. K. Tiwari, "Synchronized Truck and Drone Routing in Package Delivery Logistics," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2020.

[4] S. Sawadsitang, D. Niyato, P.-S. Tan, and P. Wang, "Joint Ground and Aerial Package Delivery Services: A Stochastic Optimization Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2241–2254, 2019.

[5] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic Load Balancing for Mobility-on-Demand Systems," *International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[6] J. Wen, J. Zhao, and P. Jaillet, "Rebalancing shared mobility-on-demand systems: A reinforcement learning approach," in *IEEE 20th International Conference on Intelligent Transportation Systems*, 2017, pp. 220–225.

[7] F. Rossi, R. Zhang, Y. Hindy, and M. Pavone, "Routing autonomous vehicles in congested transportation networks: structural properties and coordination algorithms," *Autonomous Robots*, vol. 42, no. 7, pp. 1427–1442, October 2018.

[8] M. Gueriau, F. Cugurullo, R. A. Acheampong, and I. Dusparic, "Shared Autonomous Mobility on Demand: A Learning-Based Approach and Its Performance in the Presence of Traffic Congestion," *IEEE Intelligent Transportation Systems Magazine*, vol. 12, no. 4, pp. 208–218, 2020.

[9] F. Miao, S. Han, A. M. Hendawi, M. E. Khalefa, J. A. Stankovic, and G. J. Pappas, "Data-Driven Distributionally Robust Vehicle Balancing Using Dynamic Region Partitions," in *ACM/IEEE 8th International Conference on Cyber-Physical Systems*, April 2017, pp. 261–272.

[10] M. Salazar, M. Tsao, I. Aguiar, M. Schiffer, and M. Pavone, "A congestion-aware routing scheme for autonomous mobility-on-demand systems," in *European Control Conference*, 2019.

[11] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2017, pp. 3583–3590.

[12] D. O. Santos and E. C. Xavier, "Taxi and Ride Sharing: A Dynamic Dial-a-Ride Problem with Money as an Incentive," *Expert Syst. Appl.*, vol. 42, no. 19, pp. 6728–6737, 2015.

[13] F. Bullo, E. Frazzoli, M. Pavone, K. Savla, and S. L. Smith, "Dynamic vehicle routing for robotic systems," *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1482–1504, 2011.

[14] M. Owais and A. Alshehri, "Pareto optimal path generation algorithm in stochastic transportation networks," *IEEE Access*, vol. 8, pp. 58 970–58 981, 2020.

[15] H.-K. Chen, C.-F. Hsueh, and M.-S. Chang, "The real-time time-dependent vehicle routing problem," *Transp. Res. Part E: Logistics and Transportation Review*, vol. 42, no. 5, pp. 383–408, 2006.

[16] Q. Mu, Z. Fu, J. Lysgaard, and R. Eglese, "Disruption management of the vehicle routing problem with vehicle breakdown," *Journal of the Operational Research Society*, vol. 62, no. 4, pp. 742–749, 2011.

[17] J. Zhang, K. Luo, A. M. Florio, and T. Van Woensel, "Solving large-scale dynamic vehicle routing problems with stochastic requests," *European Journal of Operational Research*, vol. 306, no. 2, pp. 596–614, 2023.

[18] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 211–236, 2018.

[19] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[20] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: Control of multi-drone fleets with temporal logic objectives," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '18. IEEE Press, 2018, p. 186ñ197. [Online]. Available: https://doi.org/10.1109/ICCPS.2018.00026

[21] J. Tumova, L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation ltl planning with conflicting specifications," in *IEEE American Control Conference*, 2013, pp. 200–205.

[22] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating Control Strategy Synthesis with Safety Rules," in *Intl Conf on Hybrid Systems: Computation and Control*, 2013, pp. 1–10.

[23] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments," in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2013, pp. 353–362.

[24] K. Kim and G. Fainekos, "Approximate Solutions for the Minimal Revision Problem of Specification Automata," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 265–271.

[25] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[26] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal Path Planning for Surveillance with Temporal Logic Constraints," *International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[27] A. Ulusoy, S. Smith, X. Ding, C. Belta, and D. Rus, "Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints," *International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[28] M. Quottrup, T. Bak, and R. Zamanabadi, "Multi-Robot Planning: A Timed Automata Approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2004, pp. 4417–4422.

[29] J. Liu and P. Prabhakar, "Switching control of dynamical systems from metric temporal logic specifications," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5333–5338.

[30] C. I. Vasile, V. Raman, and S. Karaman, "Sampling-based Synthesis of Maximally-Satisfying Controllers for Temporal Logic Specifications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, September 2017, pp. 3840–3847.

[31] S. Karaman and E. Frazzoli, "Vehicle Routing Problem with Metric Temporal Logic Specifications," in *IEEE Conference on Decision and Control (CDC)*, 2008, pp. 3953–3958.

[32] Y. Zhou, D. Maity, and J. S. Baras, "Optimal Mission Planner with Timed Temporal Logic Constraints," in *European Control Conference (ECC)*. IEEE, 2015.

[33] V. Raman, A. Donzé, D. Sadigh, R. Murray, and S. Seshia, "Reactive Synthesis from Signal Temporal Logic Specifications," in *Intl Conf on Hybrid Systems: Computation and Control*, 2015, pp. 239–248.

[34] S. Hustiu, D. V. Dimarogonas, C. Mahulea, and M. Kloetzer, "Multi-robot motion planning under mitl specifications based on time petri nets," in *2023 European Control Conference (ECC)*, 2023, pp. 1–8.

[35] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5010–5017.

[36] Y. Chen, J. Tumova, A. Ulusoy, and C. Belta, "Temporal Logic Robot Control Based on Automata Learning of Environmental Dynamics," *International Journal of Robotics Research*, vol. 32, no. 5, pp. 547–565, April 2013.

[37] C. I. Vasile, D. Aksaray, and C. Belta, "Time Window Temporal Logic," *Theoretical Computer Science*, vol. 691, no. Supp. C, pp. 27–54, 2017.

[38] D. Aksaray, C. I. Vasile, and C. Belta, "Dynamic Routing of Energy-Aware Vehicles with Temporal Logic Constraints," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3141–3146.

[39] J. Tumova, S. Karaman, C. Belta, and D. Rus, "Least-Violating Planning in Road Networks from Temporal Logic Specifications," in *ACM/IEEE 7th International Conference on Cyber-Physical Systems*, 2016, pp. 1–9.

[40] C. I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTL motion planning for mobility-on-demand," in *International Conference on Robotics and Automation*, 2017, pp. 1481–1488.

[41] O. Kupferman and M. Y. Vardi, "Model Checking of Safety Properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[42] C. I. Vasile and C. Belta, "Sampling-Based Temporal Logic Path Planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, November 2013, pp. 4817–4822.

[43] J. Tumova and D. V. Dimarogonas, "Decomposition of multi-agent planning under distributed motion and task LTL specifications," in *IEEE Annual Conference on Decision and Control*, 2015, pp. 7448–7453.

[44] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, 1st ed. Prentice Hall, February 1993.
[45] I. Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2016.
[46] 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.
[47] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, CA USA, 2008, pp. 11–15.
[48] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 – a framework for LTL and ω-automata manipulation," in *International Symposium on Automated Technology for Verification and Analysis*, ser. LNCS, vol. 9938. Springer, 2016, pp. 122–129.

Cristian-Ioan Vasile (M'11) received B.Sc., M.Eng., and Ph.D. degrees from Politehnica University of Bucharest, Romania in 2009, 2011, and 2015, and a Ph.D. degree from Boston University, USA in 2016. He was a postdoctoral associate in the Laboratory for Information and Decision Systems (LIDS), and the Computer Science and Artificial Intelligence Laboratory (CSAIL) at Massachusetts Institute of Technology (MIT). He is an Assistant Professor in the Mechanical Engineering and Mechanics department at Lehigh University, Pennsylvania, USA. He leads the Explainable Robotics Lab (ERL). He is also affiliated with the Computer Science and Engineering department and the Autonomous and Intelligent Robotics Laboratory (AIRLab) at Lehigh University. His research goal is to enable robot autonomy via scalable automated synthesis of explainable plans using motion planning and machine learning.

Jana Tumova is an associate professor at the School of Electrical Engineering and Computer Science at KTH Royal Institute of Technology. She received PhD in computer science from Masaryk University and was awarded ACCESS postdoctoral fellowship at KTH in 2013. She was also a visiting researcher at MIT, Boston University, and Singapore-MIT Alliance for Research and Technology. Her research interests include formal methods applied in decision making, motion planning, and control of autonomous systems. Among other projects, she is a recipient of a Swedish Research Council Starting Grant to explore compositional planning for multi-agent systems under temporal logic goals and a WASP Expeditions project focusing on design of correct-by-design and socially acceptable autonomous systems. She is a recipient of the Early Career Spotlight award at Robotics: Science and Systems 2021.

Sertac Karaman (M) received the S.M. degree in mechanical engineering and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2009 and 2012, respectively. He is currently an Associate Professor of aeronautics and astronautics with MIT. He studies the applications of probability theory, stochastic processes, stochastic geometry, formal methods, and optimization for the design and analysis of high-performance cyber–physical systems. The application areas of his research include driverless cars, unmanned aerial vehicles, distributed aerial surveillance systems, air traffic control, certification and verification of control systems software, and many others. Dr. Karaman was a recipient of the IEEE Robotics and Automation Society Early Career Award in 2017, the Office of Naval Research Young Investigator Award in 2017, the Army Research Office Young Investigator Award in 2015, and the National Science Foundation Faculty Career Development (CAREER) Award in 2014.

Calin Belta (F'17) received B.Sc. and M.Sc, degrees from the Technical University of Iasi, Romania in 1995 and 1997, and M.Sc. and Ph.D. degrees from the University of Pennsylvania, Philadelphia, USA in 2001 and 2003. He is the Brendan Iribe Endowed Professor in the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Maryland, College Park. His research focuses on dynamics and control theory, with particular emphasis on hybrid and cyber-physical systems, formal synthesis and verification, and applications in robotics and systems biology. He received the Air Force Office of Scientific Research Young Investigator Award and the National Science Foundation CAREER Award. He is a fellow of IEEE.

Daniela Rus (F'09) Daniela Rus is the Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and Computer Science and Director of the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. Her research interests are in robotics and artificial intelligence. The key focus of her research is to develop the science and engineering of autonomy. She is a Class of 2002 MacArthur Fellow, a fellow of ACM, AAAI and IEEE, and a member of the National Academy of Engineering and of the American Academy of Arts and Sciences. She is the recipient of the Engelberger Award for robotics. She earned her Ph.D. in Computer Science from Cornell University.