Least-violating planning in road networks from temporal logic specifications

Jana Tumova*, Sertac Karaman†, Calin Belta‡, Daniela Rus†
*KTH Royal Institute of Technology, Stockholm, Sweden
Email: tumova@kth.se
†Massachusetts Institute of Technology, Cambridge, MA, USA
Email: sertac@mit.edu, rus@csail.mit.edu
‡Boston University, Boston, MA, USA
Email: cbelta@bu.edu

***Abstract*—In this paper we consider the problem of automated plan synthesis for a vehicle operating in a road network, which is modeled as a weighted transition system. The vehicle is assigned a set of demands, each of which involves a task specification in the form of a syntactically co-safe LTL formula, a deadline for achieving this task, and a demand priority. The demands arrive gradually, upon the run of the vehicle, and hence periodical replanning is needed. We particularly focus on cases, where all tasks cannot be accomplished within the desired deadlines and propose several different ways to measure the degree of demand violation that take into account the demand priorities. We develop a general solution to the problem of least-violating planning and replanning based on a translation to linear programming problem. Furthermore, for a particular subclass of demands, we provide a more efficient solution based on graph search algorithms. The benefits of the approach are demonstrated through illustrative simulations inspired by mobility-on-demand scenarios.**

I. Introduction

This work is motivated by mobility-on-demand scenarios, where a single autonomous vehicle receives various demands from various customers over time and needs to be routed to satisfy them. For instance, a customer would like to be picked up at location A and brought to location B. Another customer is interested in visiting locations C, D and E, in this order, and yet another customer would like to be taken to any shopping mall in the city. The customers may also give deadlines by which their demands should be accomplished and furthermore, their demands may have different priorities, based e.g., on the type of membership the customers have in the mobility-on-demand system. The goal is to plan the trip of the vehicle to meet all the demands within their respective deadlines. However, it is often the case that this goal cannot be achieved. How should the vehicle proceed then? Which demands should be postponed and how much? In this work, we propose a rigorous formulation of these questions and a systematic procedure to address them. A typical scenario that we aim to address involves a low-priority demand followed by a later arrival of a high-priority demand. Initially, the vehicle should plan its path through the road network in accordance with the low-priority demand, but upon the arrival of the high-priority one, it should reconfigure its planned path and possibly delay the service of the low-priority demand for the sake of the high-priority one.

Specifically, we consider the vehicle in a road network represented as a discrete Weighted Transition System (WTS). The states of the WTS represent the locations of interest in the road network while the transitions model the road network segments connecting the locations and the vehicle's capability to move along them. The demands are given as formulas in the syntactically co-safe fragment of Linear Temporal Logic (scLTL), whose choice is motivated by its resemblance to natural language, rigorousness, and rich expressive power. Each demand is assigned its arrival time, its deadline and its priority. Loosely speaking, our goal is to plan the trace of the vehicle, i.e. a sequence of states in the WTS, to satisfy all the demands with the least possible delays, while taking into account that the importance of minimizing a task delay is proportional to the corresponding demand's priority. We propose different criteria to measure how much a given trace of the vehicle violates the achievement of the demands taking into account the delay of the highest-priority demands, the bottleneck delay or the cumulative delay. Based on these measures, we develop an algorithm that generates the least-violating trace. Moreover, for a subclass of measures, we provide a more efficient solution. As the demands arrive gradually, the trace is periodically recomputed and it holds that at any time instant, it is provably the least-violating one among all the traces that have the same history.

Related work on temporal logic-based planning under unsatisfiable specifications includes e.g., [1], [2], which aim at finite least-violating planning. The authors therein focus on finding the maximal part of the specification that can be satisfied by the system model and generating the corresponding plan for this part of specification only. In [3], [4], the given temporal logic formula is systematically revised to be satisfiable by the given model and close to the original formula. However, none of these works took into account deadlines on specification satisfaction. Quantitative models and specifications have been recently considered e.g., in [5], [6], where additionally to temporal logic satisfaction, minimization of the time elapsed between revisits to a certain subset of locations is required. Timed temporal logics have been chosen as a specification

978-1-5090-1772-0/16/$31.00 ©2016 IEEE

language e.g., in [7], [8], [9], [10], [11]. In contrast to our work, the problem they focus on is correct-by-design synthesis and not least-violating planning. Related work on periodic replanning under knowledge updates includes e.g., [12], [13], [2]. The source of the updates come from the changes in the model of the environment obtained by as opposed to changes to the specifications that are of our interest. Planning for autonomous cars in the context of a mobility-on-demand system was considered e.g., in [14], where a real-time rebalancing policy was developed to maximize the throughput of the system. To our best knowledge, this work is the first one that integrates planning for an infinite sequence of gradually arriving temporal logic specifications and planning under infeasible deadlines.

The rest of the paper is structured as follows. In Sec. II we introduce necessary notation and preliminaries. In Sec. III we formalize the measures of demand violation and state the problem of least-violating planning. Sec. IV provides a general solution and Sec. V discusses a specialized, more efficient solution for a subclass of demand specifications. In Sec. VI we provide simulation results. Finally, in Sec. VII we conclude and outline several directions for future work.

II. Preliminaries

We use $\mathbb{R}_+$ and $\mathbb{R}_0$ to denote positive and nonnegative real numbers, respectively. Given a set S, we denote by 2^{S}, and $|\mathsf{S}|$ the set of all subsets of S, and the cardinality of S, respectively. A finite and an infinite sequence of elements from S are called a finite and an infinite *word*, respectively. Given an infinite word $w = \mathsf{s}_1\mathsf{s}_2\mathsf{s}_3\ldots$, we use w_j, $w_{\leadsto j}$, and w^j to denote the *j-th element* of the sequence s_j, the *prefix* $\mathsf{s}_1\ldots\mathsf{s}_j$ ending at the j-th position of w, and the *suffix* $\mathsf{s}_j\mathsf{s}_{j+1}\mathsf{s}_{j+2}\ldots$ starting at the j-th position of w, respectively. A *fragment* of a finite or infinite word $w = \mathsf{s}_1\mathsf{s}_2\mathsf{s}_3\ldots$ is a finite subword $w^j_{\leadsto n} = \mathsf{s}_j\mathsf{s}_{j+1}\ldots\mathsf{s}_{n-1}\mathsf{s}_n$. The *concatenation* of a finite word w and a finite or an infinite word w' is denoted by $w \cdot w'$. For simplicity of the presentation, we use $\mathsf{s} \in \mathsf{s}_1\mathsf{s}_2\mathsf{s}_3\ldots$ to denote the membership of the element s in the set $\{\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3, \ldots\}$, i.e. the fact that $\mathsf{s} = \mathsf{s}_j$, for some $j \geq 1$. The *i-th projection* proj_i of a tuple $(\mathsf{s}_1, \ldots, \mathsf{s}_n)$ is $\mathrm{proj}_i(\mathsf{s}_1, \ldots, \mathsf{s}_n) = \mathsf{s}_i$. With a slight abuse of notation, the i-th projection proj_i of a sequence of tuples $(\mathsf{s}_{1,1}, \ldots, \mathsf{s}_{n,1})\ldots(\mathsf{s}_{1,m}, \ldots, \mathsf{s}_{n,m})$ is $\mathrm{proj}_i(\mathsf{s}_{1,1}, \ldots, \mathsf{s}_{n,1})\ldots(\mathsf{s}_{1,m}, \ldots, \mathsf{s}_{n,m}) = \mathsf{s}_{i,1}\ldots\mathsf{s}_{i,m}$. Given two tuples $(a_1, \ldots, a_m), (b_1, \ldots, b_m) \in \mathbb{R}_0^m$, we define their *piecewise summation* as $(a_1, \ldots, a_m) \oplus (b_1, \ldots, b_m) = (a_1 + b_1, \ldots, a_m + b_m)$.

Definition 1 (Weighted transition system (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.*

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* $\tau = s_1s_2s_3\ldots$ 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 $\tau = s_1s_2s_3\ldots$ is associated with the *time sequence* $\mathbb{T}(\tau) = t_1t_2t_3\ldots$, 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 sum of the weights of the transitions executed, i.e. the time elapsed till reaching the j-th state s_i on the trace τ. A trace τ may also be viewed as a *control strategy* for the WTS $\mathcal{T}$. 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 as follows*

$$\varphi ::= \pi \mid \neg\pi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathcal{X}\,\varphi \mid \mathcal{F}\,\varphi \mid \varphi\,\mathcal{U}\,\varphi,$$

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

An scLTL formula is interpreted over infinite words over 2^Π, such as the ones produced by a WTS.

Definition 3 (scLTL semantics) *The* satisfaction *of an scLTL formula φ by a word $w = w_1w_2w_3\ldots$ over 2^Π is defined through the satisfaction relation $\models$ as follows:*

$$\begin{aligned}
&w \models \pi && \iff \pi \in w_1\\
&w \models \neg\pi && \iff \pi \notin w_1\\
&w \models \varphi \vee \psi && \iff w \models \varphi \vee w \models \psi\\
&w \models \varphi \wedge \psi && \iff w \models \varphi \wedge w \models \psi\\
&w \models \mathcal{X}\,\varphi && \iff w^2 \models \varphi\\
&w \models \mathcal{F}\,\varphi && \iff \exists i \geq 1.\, w^i \models \varphi\\
&w \models \varphi\,\mathcal{U}\,\psi && \iff \exists i \geq 1.\, w^i \models \psi \wedge \forall 1 \leq j < i.\, w^j \models \varphi
\end{aligned}$$

Although scLTL formulas are defined over infinite words, their satisfaction is decided in finite time. Specifically, a word w over 2^Π satisfies φ over Π if it contains a *good prefix* defined as a finite prefix $w_1w_2w_3\ldots w_n$, with the property that $w' = w_1w_2w_3\ldots w_nw'_{n+1}w'_{n+2}\ldots \models \varphi$, for all suffixes $w'_{n+1}w'_{n+2}\ldots$ over 2^Π.

Definition 4 (Minimal good prefix) *Given a word w over 2^Π and ϕ over Π, a good prefix $w_1w_2w_3\ldots w_n$ of w is* minimal*, if $w_1w_2w_3\ldots w_{n-1}$ is not a good prefix.*

The definition of the satisfaction relation is extended to traces of a WTS in the expected way: $\tau \models \varphi$ if and only if $w(\tau) \models \varphi$. A (minimal) *good trace prefix* is the one that produces a (minimal) good prefix.

Definition 5 (Finite automaton) *A nondeterministic finite automaton is 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; $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation; $F \subseteq Q$ is a set of finite states.*

A run of a finite automaton $\mathcal{A}$ over a finite word $w = \sigma_1\sigma_2 \dots \sigma_n$ is a sequence of states $\rho = q_1 \dots q_{n+1}$, 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 $q_{n+1} \in F$. The language of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \mid \exists$ accepting run ρ over $w\}$. A finite automaton is nonblocking and deterministic if for all $q \in Q$, $\sigma \in \Sigma$ there exists at least and most one q', such that $(q, \sigma, q') \in \delta$, respectively.

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 $L(\mathcal{A})$ is the set of all good prefixes of all words that satisfy φ [15].

III. Problem Formulation

We introduce the model and the demand specification involving (i) temporal logic formulas expressing the desired system behaviors, (ii) task deadlines and (iii) task priorities. We give three different notions of measuring the quality of a system behavior with respect to the satisfaction of a given set of tasks, and we formalize the problem of least violating planning.

A. Model

We consider a WTS $\mathcal{T} = (S, s_{init}, R, W, \Pi, L)$. The set of states S represents locations of interest of the road network, i.e., intersections, drop-off points, etc. The state s_{init} is the vehicle's initial location. The transition relation R represents the vehicle's motion capabilities between the locations of the road network. A transition $(s, s') \in R$ denotes that the vehicle can move from the location s to s'. The weight function W assigns time durations to the transitions. We assume that for all $s \in S$, $(s, s) \in R$, $W(s, s) = \epsilon$, for a very small $\epsilon > 0$ to reflect the vehicle's capability of staying at its current location for the following time unit. The set of atomic propositions Π is the set of properties of interest, such as "a pick-up location" or "a drop-off point". The labeling function L labels each state with the atomic proposition that holds true in there.

Example 1 *An example of a vehicle road network modeled as a WTS with the set of atomic propositions* $\Pi = \{\mathsf{A}, \dots, \mathsf{H}, \text{drop-off}\}$ *is depicted in Fig. 1.*

B. Specification

A vehicle in a road network represented as a WTS is given an infinite set of demands $\mathcal{D} = \{D_1, D_2, \dots\}$ that appear gradually, at times $t_{D_1} \leq t_{D_2} \leq \dots$, where $t_{D_i} \in \mathbb{R}_0$, for all $i \geq 1$. Each demand comprises of three elements: a task, a deadline, and a priority. Since each task specifies desired behavior only in a finite time window, the syntactically co-safe fragment of LTL is expressive enough, but easier to handle in comparison to full LTL. Formally, for all $i \geq 1$, the demand D_i is a tuple $D_i = (\varphi_i, T_i, p_i)$, where

- φ_i is an scLTL formula;
- $T_i \in \mathbb{R}_0$ is a deadline; and
- $p_i \in \mathbb{N}$ is a priority.

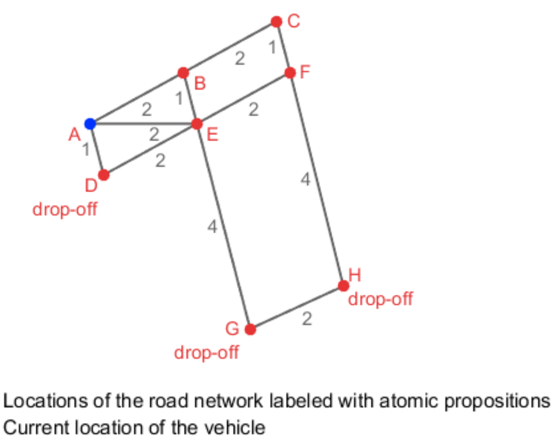

Fig. 1: An example of a WTS. The locations of interest (states of the WTS) are depicted as orange nodes and they are labeled with the atomic propositions that hold in there, e.g., the bottom right node is labeled with $\{\mathsf{H}, \text{drop-off}\}$. The vehicle's motion capabilities (transitions of the WTS) are illustrated as edges and they are labeled with the travel times (weights). In this WTS, $(s, s') \in R \Rightarrow (s', s) \in R$, and $W(s, s') = W(s', s)$, and hence we omit the edge orientation.

C. Problem statement

Loosely speaking, our goal is to find a trace of the vehicle in the road network, i.e., a trace of $\mathcal{T}$, that is the least violating in terms of servicing the desired tasks within the prescribed deadlines while taking into account their priorities. Before stating our problem, we introduce several intermediate definitions that will allow us to formalize and measure the level of satisfaction of the demands.

Consider a fixed trace $\tau = s_1 s_2 s_3 \dots$ of $\mathcal{T}$, with the time sequence $\mathbb{T}(\tau) = t_1 t_2 t_3 \dots$, and a demand $D_i = (\varphi_i, T_i, p_i)$ that has arrived at time t_{D_i}.

Definition 6 (Task delay) *Let $j \in \mathbb{N}$ be the index with the property that $t_{j-1} < t_{D_i} \leq t_j$. Task φ_i that arrived at t_{D_i} is* serviced *if $\tau^j \models \varphi_i$, i.e. if φ_i is satisfied starting at t_j. The* time of the task service *is $t_{\varphi_i} = t_k$, such that $s_j s_{j+1} \dots s_{k-1} s_k$ is the minimal good prefix of τ^j with respect to φ_i according to Def. 4. The* task execution duration *is*

$$d_i = (t_{\varphi_i} - t_{D_i}); \text{ and}$$

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 the task 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. Note that $\Delta_i = \infty$ if and only if $\tau^j \not\models \varphi_i$.

Definition 7 (Active demand) *Demand D_i is* active *on the trace τ at time t if and only if $t_{D_i} \leq t < t_{\varphi_i}$, i.e. if the task φ_i has arrived, but has not been serviced yet. Let $\mathcal{D}_\tau(t)$ denote the set of active demands on the trace τ at time t.*

For simplicity, we use $\mathcal{D}(t)$ to denote $\mathcal{D}_\tau(t)$ whenever τ is clear from the context. Note that all traces τ that share a common prefix $\tau_{\rightsquigarrow j}$ share also the same $\mathcal{D}_\tau(t)$, for all time instants $t \leq t_j$. If the prefix $\tau_{\rightsquigarrow j}$ is clear from the context, we also use $\mathcal{D}(t)$ to represent the mentioned unique set of active demands at time $t \leq t_j$.

Since the demands arrive gradually, we re-compute the trace of $\mathcal{T}$ on-the-fly, upon the system execution. We assume that the re-computation can take place only at states, i.e. if a transition is being executed, it has to be completed before the plan for future transitions can be modified.

We define the degree to which a trace τ meets a given set of demands through the task execution durations together with the corresponding demand deadlines and priorities. Since the demands arrive gradually and are unknown prior the deployment of the system, we define this degree as a value that dynamically changes over time; at a time t, it is dependent on the task execution durations, deadlines and priorities of the active demands only. Our aim is then to maximize the degree of satisfaction at all times; more precisely, we aim to maximize the degree of satisfaction at the times of arrivals to states, i.e. the times when re-computation can take place.

Below, we give three different definitions of the degree to which a trace τ meets a set of active demands at a given time $t_j \in \mathbb{T}(\tau)$, which we call *trace penalty* $\lambda(\tau, j)$. In all three cases, the goal is the same: to find a trace of $\mathcal{T}$ that minimizes the trace penalty.

1) Highest priority first: We would like to find a trace, such that the largest subset $\mathcal{D}_{max} \subseteq \mathcal{D}(t_j)$ of the most prioritized demands is met within their respective deadlines at each time $t_j \in \mathbb{T}(\tau)$, i.e. such that there does not exist a set of demands $\mathcal{D}'_{max} \subseteq \mathcal{D}(t_j)$ that would contain a demand $D'_i \in \mathcal{D}'_{max} \setminus \mathcal{D}_{max}$ with the property that $p'_i \geq p_i$, for all $i \in D_{max}$.

A suitable *highest-priority-first penalty* function λ_h is defined as:

$$\lambda_h(\tau, j) = \sum_{D_i \in \mathcal{D}(t_j)} |\mathcal{D}(t_j)|^{p_i} \cdot I(i), \text{ where} \tag{1}$$

$$I(i) = \begin{cases} 0 & \text{if } \Delta_i \leq 0 \\ 1 & \text{if } \Delta_i > 0. \end{cases}$$

2) Bottleneck delay: We would like to find a trace that is as "fair" as possible with respect to the active demands. At any time, we aim to minimize the longest task delay, weighted by its priority.

A suitable *bottleneck-delay penalty* function λ_b is:

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

3) Cumulative delay: We would like to find a trace that minimizes the total sum of the active demands' delays weighted by their respective priorities.

A suitable *cumulative-delay penalty* function λ_c is:

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

The highest-priority-first penalty function is non-preemptive; satisfaction of the highest priority demands in time will be always preferred regardless how long the lower priority demands are delayed. On the other hand, the latter two penalty functions allow for a trade-off. The bottleneck delay one ensures that none of the demands is delayed for too long, whereas the cumulative delay targets the efficiency of servicing all active demands.

Example 1 (Cont.) *An example of two demands for the road network in Fig. 1 is given in Table I. Suppose that they both arrived at time $t_1 = 0$, i.e. $t_{D_1} = 0$ and $t_{D_2} = 0$. Thus, they are both active at time $t_1 = 0$. Task φ_1 is to visit locations* E, *then* B, *and then* H. *Task φ_2 is to visit any of the locations labeled with* $\mathsf{drop\text{-}off}$, *i.e. one of* $\mathsf{D}, \mathsf{G}, \mathsf{H}$. *Fig. 2 shows two different good trace prefixes for formulas φ_1 and φ_2. The one in Fig. 2.(A) satisfies φ_2 by visiting* H *at time $t_6 = 10$ and it is a minimal good prefix for both formulas φ_1 and φ_2. The one in Fig. 2.(B) is a minimal good prefix for φ_1 and satisfies φ_2 by visiting* D *at time $t_2 = 1$, i.e. demand D_2 is not active on it from time t_2 on. Table II summarizes the values of the task execution durations, the task delays, and the penalty functions from Eq.* (1), (2), *and* (3) *for the cases (A) and (B), respectively. Because D_1 has a higher priority than D_2, the highest-priority-first penalty function λ_h indicates that the trace prefix in Fig. 2.(A) is preferred. On the other hand, the cumulative-delay penalty function λ_c reflects that the accomplishment of φ_2 is excessively delayed on the trace prefix in Fig. 2.(A) and indicates that a trade-off for very short delay of the high-priority task should be made. However, if the priority of demand D_1 was $p_1 = 10$, both the bottleneck-delay and the cumulative-delay penalty function would prefer the trace prefix in Fig. 2.(B). On the other hand, if the priority of demand D_1 was lower, e.g., if $p_1 = 2$, the bottleneck-delay penalty function λ_b would favor case (A).*

	scLTL formula	Arrival	Deadline	Priority
D_1	$\varphi_1 = \mathcal{F}(\mathsf{E} \wedge \mathcal{F}(\mathsf{B} \wedge \mathcal{F}\mathsf{H}))$	$t_{D_1} = 0$	$T_1 = 10$	$p_1 = 7$
D_2	$\varphi_2 = \mathcal{F}\mathsf{drop\text{-}off}$	$t_{D_2} = 0$	$T_2 = 3$	$p_2 = 1$

TABLE I: An example of two demands for the WTS from Fig. 1.

Our problem is stated formally as follows:

Problem 1 (Least violating planning) Given *a WTS $\mathcal{T}$ and an infinite set of demands $\mathcal{D}$,* find *a trace τ^* of $\mathcal{T}$, such that for all $j \geq 1$, the trace penalty $\lambda(\tau^*, j)$ is minimized among $\{\tau \mid \tau$ is a trace of $\mathcal{T}$ and $\tau_{\rightsquigarrow j} = \tau^*_{\rightsquigarrow j}\}$, i.e. the traces that have the same prefix up to t_j as the trace τ^*.*

IV. General Solution

The three trace qualities λ defined above represent three different goals of least violating planning. However, these are only particular examples of trace penalty functions that can be employed in Problem 1. In this section, we introduce a general

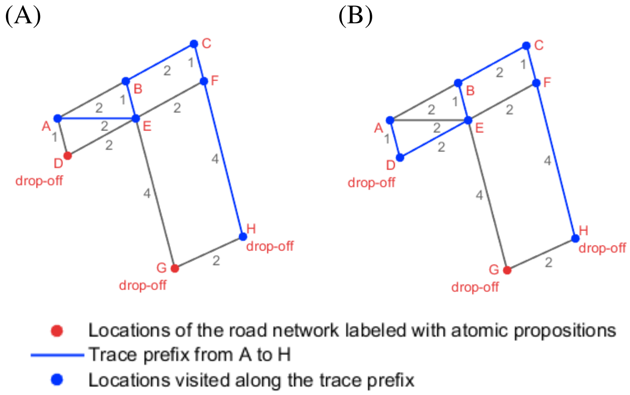

Fig. 2: An example of two good prefixes for a WTS from Fig. 1 and demands from Table I leading from the current location A to location H.

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

TABLE II: Values for the trace prefixes in Fig. 2.(A) and 2.(B), respectively.

solution to Problem 1 that can be applied to any instance, where $\lambda(\tau, j)$ is defined as a function of the task execution durations d_i, task deadlines T_i and the priorities p_i of the active demands $D_i \in D(t_j)$ at time t_j:

$$\lambda(\tau, j) : \mathbb{R}_0^{m_j} \times \mathbb{R}_0^{m_j} \times \mathbb{N}^{m_j} \to \mathbb{R}_0$$
$$\lambda(\tau, j) = f\big((d_{i_1}, \ldots, d_{i_{m_j}}), (T_{i_1}, \ldots, T_{i_{m_j}}), (p_{i_1}, \ldots, p_{i_{m_j}})\big), \quad (4)$$

where $m_j = |\mathcal{D}(t_j)|$. In the next section, we discuss a subclass of trace quality functions that allow for a more efficient solution in terms of computational complexity.

A. Solution Overview

To address Problem 1, we first focus on solving the following problem:

Problem 2 (Least violating suffix search) Given *a trace prefix $\tau_{\rightsquigarrow j} = \tau_{\rightsquigarrow (j-1)} \cdot s_j$ at time $t_j \in \mathbb{R}_0$ and a set of active demands $\mathcal{D}(t_j)$* find *a trace suffix $\tau^{j*} = s_j \cdot \tau^{j+1}$ that minimizes the trace penalty $\lambda(\tau_{\rightsquigarrow (j-1)} \cdot \tau^{j*}, j)$.*

In order to solve Problem 1, we initialize $j := 1$, $\tau_{\rightsquigarrow j} := s_{init}$, and $t_1 := 0$ and we iteratively

(i) find a solution $\tau^{j*} = s_j s_{j+1} s_{j+2} \ldots$ to Problem 2;
(ii) execute the first transition (s_j, s_{j+1}) of τ^{j*} ; and
(iii) repeat the procedure starting with step (i) with $\tau_{\rightsquigarrow j} := \tau_{\rightsquigarrow j} \cdot s_j$, $t_j := t_j + W(s_j, s_{j+1})$ and $j := j + 1$.

B. Solution to Problem 2

We propose a solution to Problem 2 leveraging ideas from automata-based model checking. At each time t_j when the $(j-1)$-th transition of the planned trace is completed, we build a finite weighted product automaton $\mathcal{P}(j)$ between the transition system $\mathcal{T}$ and the finite automata that represent the tasks of the currently active demands $\mathcal{D}(t_j)$. Intuitively, the initial state of the product automaton $\mathcal{P}(j)$ reflects the current state of the transition system as well as the progress towards the satisfaction of the active demands. Furthermore, the weight function of $\mathcal{P}(j)$ is derived from $\mathcal{T}$. This allows us to translate the considered problem into a search for an accepting run in $\mathcal{P}(j)$ that is optimal with respect to a certain optimality criterion captured through the trace penalty function λ. The general idea of translating a least violating planning problem into a search in a weighted product automaton has been introduced earlier, e.g., in [1]. The key step in the solution here is the design of the weight function and the initial state that allows us to do so.

1) Finding optimal trace suffix τ^{j*} at time t_j: Consider the WTS $\mathcal{T}$, its trace prefix $\tau_{\rightsquigarrow j}$ executed up to time t_j, a set of active demands $\mathcal{D}(t_j) = \{D_{i_1}, \ldots, D_{i_{m_j}}\} \neq \emptyset$, and a trace penalty function λ. Let each of the formulas φ_i, for all $D_i \in \mathcal{D}(t_j)$ be translated into a nonblocking deterministic finite automaton $\mathcal{A}_i = (Q_i, q_{init,i}, 2^\Pi, \delta_i, F_i)$, such that $\delta_i(q, \sigma) \in F_i$ for all $q \in F_i$ and $\sigma \in 2^\Pi$. This assumption is not restrictive due to the fact that φ_i is an scLTL formula. Let $\mathfrak{s}_j$ denote the current state of the transition $\mathcal{T}$ at time t_j, i.e. the last state of $\tau_{\rightsquigarrow j} = \tau_{\rightsquigarrow (j-1)} \cdot \mathfrak{s}_j$ 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 $t_j = t_1 = 0$, $\mathfrak{s}_1 = s_{init}$, and $\mathfrak{q}_{i,j} = q_{init,i}$, for all $D_i \in \mathcal{D}(t_1)$.

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*

- $Q_\mathcal{P} = S \times Q_{i_1} \times \ldots \times Q_{i_{m_j}}$;
- $q_{init,\mathcal{P}} = (\mathfrak{s}_j, \mathfrak{q}_{i_1,j}, \ldots, \mathfrak{q}_{i_{m_j},j})$;
- $\delta_\mathcal{P} \subseteq Q_\mathcal{P} \times Q_\mathcal{P}$ *is a transition relation;* $\big((s, q_1, \ldots, q_{m_j}), (s', q_1', \ldots, q_{m_j}')\big) \in \delta_\mathcal{P}$ *if*
  - $(s, s') \in R$ *and*
  - $(q_i, L(s), q_i') \in \delta_i, \forall i \in \{i_1, \ldots, i_{m_j}\}$;
- $F_\mathcal{P} = \{(s, q_{i_1}, \ldots, q_{i_{m_j}}) \mid q_i \in F_i, \forall i \in \{i_1, \ldots, i_{m_j}\}\}$;
- $E_{init,\mathcal{P}} = (\nu_1, \ldots, \nu_{m_j})$, *where* $\forall \ell \in \{1, \ldots, m_j\}$, $\nu_\ell = t_{i_\ell} - t_{D_{i_\ell}}$ *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* $W_\mathcal{P}\big((s, q_1, \ldots, q_{m_j}), (s', q_1', \ldots, q_{m_j}')\big) = (\nu_1, \ldots, \nu_{m_j})$, *such that* $\forall \ell \in \{1, \ldots, m_j\}$,
$$\nu_i = \begin{cases} W(s, s') & \text{if } q_{i_\ell} \notin F_{i_\ell} \\ 0 & \text{otherwise.} \end{cases}$$

The weighted product $\mathcal{P}(j)$ can be viewed as a nonblocking nondeterministic finite automaton enhanced with the initial state evaluation and the transition weights. In what follows, the alphabet is not significant, and therefore we omit it.

A finite run of $\mathcal{P}(j)$ is a sequence $\rho = p_j p_{j+1} \dots p_n$, where $\forall j \leq k \leq n$, $p_k = (s_k, q_{1,k}, \dots, q_{m_j,k})$, $p_j = q_{init,\mathcal{P}}$, and $\forall j \leq k < n$ it holds that $(p_k, p_{k+1}) \in \delta_{\mathcal{P}}$. It is accepting if $p_n \in F_{\mathcal{P}}$. From the construction of $\mathcal{P}(j)$, it follows that ρ projects onto a trace fragment $\mathrm{proj}_1(\rho) = \tau^j_{\rightsquigarrow n} = \mathfrak{s}_j s_{j+1} \dots s_n$ of $\mathcal{T}$ and a finite run fragment $\mathrm{proj}_{\ell+1}(\rho) = \varrho^j_{i_\ell, \rightsquigarrow n} = \mathfrak{q}_{i_\ell,j} q_{i_\ell,j+1} \dots q_{i_\ell,n}$ of $\mathcal{A}_{i_\ell}$, for all $\ell \in \{1, \dots, m_j\}$. If ρ is accepting then $\tau_{\rightsquigarrow(j-1)} \cdot \tau^j_{\rightsquigarrow n}$ 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 $D_{i_\ell} \in \mathcal{D}(t_j)$. Vice versa, for each good trace prefix $\tau_{\rightsquigarrow n}$ of $\mathcal{T}$ that produces a word accepted by $\mathcal{A}_i$ via an accepting run ϱ_i for each demand $D_i \in \mathcal{D}(t_j)$, there exists a finite accepting run of $\mathcal{P}$ that projects onto a suffix $\tau^j_{\rightsquigarrow n}$ of $\tau_{\rightsquigarrow n}$ that starts in $\mathfrak{s}(t_j)$ and onto a suffix $\varrho^j_{i,\rightsquigarrow n}$ of each ϱ_i that starts in $\mathfrak{q}_{i,j}$ and ends in an accepting state.

We associate each finite accepting run $\rho = p_j p_{j+1} \dots p_n$ with a sequence of *state evaluations*

$$E(\rho, j) = E_{init,\mathcal{P}}, \text{ and}$$
$$E(\rho, k) = E(\rho, k-1) \oplus W_{\mathcal{P}}(p_{k-1}, p_k)$$

being a piecewise summation (see Sec. II), for all $j < k \leq n$. The value of the i-th projection of $E(\rho, n)$ represents the task execution duration d_{i_ℓ} of the task φ_{i_ℓ}, where $\ell \in \{1, \dots, m_j\}$, for any trace τ with $\tau_{\rightsquigarrow j}$ being the trace prefix up to t_j and $\tau^j_{\rightsquigarrow n} = \mathrm{proj}_1(\rho)$ being the trace fragment obtained by the first projection from ρ. Henceforth, from $E(\rho, n)$ we can also easily compute common value of $\lambda(\tau, 1)$ for all traces τ with the trace prefix with $\tau_{\rightsquigarrow n} = \tau_{\rightsquigarrow(j-1)} \cdot \tau^j_{\rightsquigarrow n}$. Thus, in order to find a solution to Problem 2 at time t_j it is sufficient to find a finite accepting run $\rho = p_1, \dots, p_n$ of $\mathcal{P}$ that ensures the minimization of

$$\lambda(\tau, j) = f\big(E(\rho, n), (T_{i_1}, \dots, T_{i_{m_j}}), (p_{i_1}, \dots, p_{i_{m_j}})\big).$$

We formulate this as a Linear Programming (LP) problem:

$$\min_{p_f \in F_{\mathcal{P}}} f(e(p_f), T_{i_1}, \dots, T_{i_{m_j}}, p_{i_1} \dots, p_{i_{m_j}}) \tag{5}$$

Subject to:

$$e(q_{init,P}) = E_{init,\mathcal{P}}$$
$$e(p) \geq e(p') \oplus W_{\mathcal{P}}(p', p), \text{ for all } p \in Q_{\mathcal{P}}, (p', p) \in \delta_{\mathcal{P}} \tag{6}$$

Once solving the above LP problem, the reconstruction of the optimal accepting run $\rho^* = p_j \dots p_n$ is straightforward: starting from the final state

$$p_n = \mathrm{argmin}_{p_f \in F_{\mathcal{P}}} f(e(p_f), T_{i_1}, \dots, T_{i_{m_j}}, p_{i_1} \dots, p_{i_{m_j}}),$$

we iteratively compute p_{k-1} as a state satisfying that

$$e(p_k) = e(p_{k-1}) + W_P(p_{k-1}, p_k)$$

until $p_{k-1} = q_{init,\mathcal{P}}$.

Note that ρ^* defines a prefix $\tau^{j*}_{\rightsquigarrow n} = \mathfrak{s}_j s_{j+1} \dots s_n$ of the desired optimal trace suffix τ^{j*} via the first projection; in fact any trace suffix τ^j with the prefix $\tau^{j*}_{\rightsquigarrow n}$ is a solution to Problem 2. Without loss of generality, we choose $\tau^{j*} = \mathfrak{s}_j s_{j+1} \dots s_n \cdot s_n^\omega = \tau^{j*}_{\rightsquigarrow n} \cdot s_n^\omega$.

Lemma 1 *The suggested trace suffix τ^{j*} is a solution to Problem 2 at time t_j for the trace prefix $\tau_{\rightsquigarrow j} = s_{init} \dots \mathfrak{s}_j$ and the set of active demands $D(t_j) \neq \emptyset$.*

Proof. The proof follows directly from the discussion above. Namely, it holds that $\tau^{j*}_{\rightsquigarrow n}$ is a minimal good prefix from the fact that $p_n \in F_{\mathcal{P}}$ and from the construction of $\mathcal{P}$. Furthermore, $e(p_n) = (d_{i_1}, \dots, d_{i_{m_j}})$ for the trace $\tau_{\rightsquigarrow(j-1)} \cdot \tau^{j*}$ at time t_j. Hence, $f(e(p_n), T_{i_1}, \dots, T_{i_{m_j}}, p_{i_1} \dots, p_{i_{m_j}})$ = $\lambda(\tau_{\rightsquigarrow(j-1)} \cdot \tau^{j*}, j)$ at time t_j.

2) Execution procedure at time t_j: Given the optimal accepting run of $\rho^* = p_j, \dots, p_n$ of $\mathcal{P}(j)$ and the optimal trace suffix $\tau^{j*} = \mathfrak{s}_j \dots s_n \cdot s_n^\omega$ of $\mathcal{T}$ 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 $(\mathrm{proj}_i(p_1), L(\mathfrak{s}_j), \mathrm{proj}_i(p_2)) \in \delta_i$ is taken in $\mathcal{A}_{i_\ell}$, for all $\ell \in \{1, \dots, m_j\}$; If $\mathrm{proj}_i(p_2) \notin F_{i_\ell}$ then include D_{i_ℓ} in $\mathcal{D}(t_{j+1})$ and the current state of $\mathcal{A}_{i_\ell}$ at time t_{j+1} becomes $\mathfrak{q}_{i_\ell,j+1} = \mathrm{proj}_i(p_2)$.

At time t_{j+1}

- Include all newly arrived demands D_i, such that $t_j < t_{D_i} \leq t_{j+1}$ in $\mathcal{D}(t_{j+1})$;
- If $\mathcal{D}(t_{j+1}) = \{D_{i'_1}, \dots D_{i_{m'_{j+1}}}\} \neq \emptyset \neq \mathcal{D}_{t_j}$ then compute $\mathcal{P}(j+1) = \mathcal{T} \otimes \mathcal{A}_{i'_1} \otimes \dots \otimes \mathcal{A}_{i'_{m_{j+1}}}$ at time t_{j+1} as in Def. 8, and compute ρ^* for $\mathcal{P}(j+1)$ and the optimal suffix τ^{j+1*} through the LP problem from Eq. (5–6); else
- If $\mathcal{D}_{t_{j+1}} = \emptyset$ then choose $\tau^{j+1*} = \mathfrak{s}^\omega_{j+1}$;
- Repeat the execution procedure at time t_{j+1} with τ^{j+1*}.

Theorem 1 *The trace $\tau = s_1 s_2 \dots$ of $\mathcal{T}$, with the property that $s_1 = s_{init}$, and for all $j \geq 1$ it holds that $(s_j, s_{j+1}) \in \mathcal{R}$ is the first transition of the trace $\tau^{j*} = s_j s_{j+1} \dots$ computed as above, is the desired solution τ^* to Problem 1.*

Proof. The proof follows directly from Lemma 1 and from the construction of the execution procedure.

V. Specialized Solution

In Section IV, we introduced a general solution to Problem 1 for an arbitrary trace penalty function λ that solves an LP problem in each iteration of the proposed algorithm. Although an LP problem can be solved in polynomial time with respect to the number of variables, i.e. the number of states in the product automaton $\mathcal{P}(t_j)$, none of the existing solutions has achieved linear time computational complexity. In this section, we discuss that for specific subclass of penalty functions the worst-case complexity of finding the desired optimal run ρ^* is linear with respect to the size of $\mathcal{P}(t_j)$.

A. Subclass of trace penalty functions

Consider $\mathcal{D}(t_j) = \{D_{i_1}, \ldots, D_{m_j}\}$. For simplicity of the presentation, we denote $d(t_j) = (d_{i_1}, \ldots, d_{i_{m_j}})$, $T(t_j) = (T_{i_1}, \ldots, T_{i_{m_j}})$, $p(t_j) = (p_{i_1}, \ldots, p_{i_{m_j}})$. Then $\lambda(\tau, j)$ from Eq. (4) can be written as $\lambda(\tau, j) = f\big(d(t_j), T(t_j), p(t_j)\big)$.

Assumption 1 *Let d_{t_j}, $d'_{t_j} \in \mathbb{R}_0^{m_j}$ be such that $f\big(d(t_j), T(t_j), p(t_j)\big) \leq f\big(d(t_j)', T(t_j), p(t_j)\big)$. We assume that*

$$f\big(d(t_j) \oplus (\nu_{i_1}, \ldots, \nu_{i_{m_j}}), T(t_j), p(t_j)\big) \leq$$
$$f\big(d(t_j)' \oplus (\nu_{i_1}, \ldots, \nu_{i_{m_j}}), T(t_j), p(t_j)\big).$$

Remark 1 *It is easy to check that Assumption 1 holds for instance for the cummulative-delay penalty function λ from Eq. (3), while it does not hold for the highest-priority-first and the bottleneck-delay penalty functions from Eq. (1) and Eq. (2), respectively. However, it would also hold for a modified highest-priority-first penalty function, which reflects the quantity of the delay rather than its presence and absence:*

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

B. Solution Modification

Consider the product automaton $\mathcal{P}(j)$ from Def. 8. Informally, Assumption 1 will allow us to employ linear-time graph algorithms based on the well-known Dijkstra shortest path search in order to find the desired finite accepting run in $\mathcal{P}(j)$ instead of more computationally demanding LP.

First, let us view the product automaton $\mathcal{P}(j)$ as a weighted graph with the set of vertices $Q_\mathcal{P}$, and the set of edges and weights defined from $\delta_\mathcal{P}$ and $W_\mathcal{P}$ in the straightforward way. The proposed algorithm $\texttt{OptAcc}(\mathcal{P}(j), f)$ is summarized in Algorithm 1. With each vertex $p \in Q_\mathcal{P}$ we associate its evaluation $e(p)$ and its predecessor $\mathrm{pred}(p)$ that are gradually updated. Initially, $e(p)$ is set to ∞ for all states except for the initial one that is assigned the evaluation $E_{init,\mathcal{P}}$ (lines 2 and 5). Furthermore, $\mathrm{pred}(p)$ is set to $None$ for all states. The algorithm searches through the previously unvisited states in a particular order. At each iteration, the state p that minimizes the value $f(e(p), T(t_j), p(t_j))$ is selected to be visited next (line 7). Each of its successor p' is then inspected for an update. As opposed to traditional Dijkstra algorithm, the updates are based on comparisons of the values of $f(e(p), T(t_j), p(t_j))$ with $f(e(p') \oplus W_\mathcal{P}(p', p), T(t_j), p(t_j))$ (line 11). If the value of the latter is less or equal than the former, the update of $e(p')$ and $\mathrm{pred}(p')$ takes place (line 12). Upon the termination of the algorithm, the desired finite accepting run is reconstructed from the state $p_f \in F_\mathcal{P}$ that minimizes $f(e(p_f), T(t_j), p(t_j))$ (lines 18–20).

Once ρ^* is obtained as a solution to $\texttt{OptAcc}(\mathcal{P}(j), f)$ the remainder of the solution does not differ from the solution in the general case; ρ^* is projected onto a trace fragment $\tau^{j*}_{\rightsquigarrow n} = \mathfrak{s}_j s_{j+1} \ldots s_n$ of the desired optimal trace suffix, which is then obtained as $\tau^{j*} = \mathfrak{s}_j s_{j+1} \ldots s_n \cdot s_n^\omega = \tau^{j*}_{\rightsquigarrow n} \cdot s_n^\omega$.

Algorithm 1: Optimal accepting run, $\texttt{OptAcc}(\mathcal{P}(j), f)$

Input: Product automaton $\mathcal{P}(j)$ and penalty function f satisfying Assump. 1
Output: Optimal accepting run ρ^*

1 **forall the** $p \in Q_\mathcal{P}$ **do**
2 $\quad e(p) := \infty$; $\mathrm{pred}(p) := None$;
3 $\quad$ Add p to $Unvisited$;
4 **end**
5 $e(q_{init,\mathcal{P}}) := E_{init,\mathcal{P}}$;
6 **while** $Unvisited \neq \emptyset$ **do**
7 $\quad p := \mathrm{argmin}_{p' \in Unvisited} f(e(p'), T(t_j), p(t_j))$;
8 $\quad$ remove p from $Unvisited$;
9 $\quad$ **forall the** $(p, p') \in \delta_\mathcal{P}$ **do**
10 $\quad\quad cand := f\big(e(p) \oplus W_\mathcal{P}(p, p'), T(t_j), p(t_j)\big)$
11 $\quad\quad$ **if** $cand \leq f(e(p'), T(t_j), p(t_j))$ **then**
12 $\quad\quad\quad e(p') := e(p) \oplus W_\mathcal{P}(p, p')$; $\mathrm{pred}(p') := p$;
13 $\quad\quad$ **end**
14 $\quad$ **end**
15 **end**
16 $p_f := \mathrm{argmin}_{p' \in Q_\mathcal{P}} f(e(p'), T(t_j), p(t_j))$;
17 $p := p_f$; $\rho^* := p_f$;
18 **while** $p \neq q_{init,\mathcal{P}}$ **do**
19 $\quad p := \mathrm{pred}(p)$; $\rho^* := p \cdot \rho^*$;
20 **end**
21 **return** ρ^*

VI. Simulation results

We implemented the proposed solution in MATLAB and ran simulations for several inputs.

For an illustrative example, we consider the road network with 8 nodes from Example 1 presented in Fig. 1. Three demands occur that are summarized in Table III. The first one aims to visit locations B and H, the second one locations B and C, and the third one E and G. The first two demands are of a low priority and arrive at time $t_1 = 0$, while the third one is of a high priority and arrives at time $t = 4$. Fig. 3 depicts the simulation results for the cummulative-delay penalty function λ_c from Eq. 3.

At time $t_1 = 0$, only demands D_1 and D_2 are active and the synthesized trace prefix $\tau^{1*}_{\rightsquigarrow 5}$ of the desired trace τ^{1*} illustrated in Fig. 3.(A) is good since it visits B, C, and H and thus satisfies formulas φ_1 and φ_2. It is least-violating in the sense that it minimizes the cummulative-delay among all trace prefixes that visit B, C, and H. Fig. 3.(B) shows the progress of the vehicle in the road network after execution of the first transition in the computed trace prefix. Since the set of active demands has not changed, the suffix $\tau^{2*}_{\rightsquigarrow 5}$ of the previously computed $\tau^{1*}_{\rightsquigarrow 5}$ is set as the optimal one to be followed. Fig. 3.(C) illustrates the situation at time $t_3 = 4$, when demand D_2 has been completed and demand D_3 has arrived. Hence, the set of active demands at t_3 contains D_1 and D_3. The trace computation leads to reconfiguration of the optimal good prefix of the desired trace suffix τ^{3*} to satisfy D_1 with delay $\Delta_1 = (13-0)-9 = 4$ and D_2 with delay $\Delta_2 = (11-4)-7 = 0$ and achieve $\lambda_c(\tau*, 3) = 4$. An alternative solution would be to continue with the execution of originally computed $\tau^{1*}_{\rightsquigarrow 5}$ followed by a visit to G and E. On such a good trace prefix, D_1 would be satisfied without any delay, whereas the delay of D_3 would be $\Delta_2 = (15-4)-7 = 4$. However, due to the high priority of demand D_3, the cummulative delay would be $\lambda_c(\tau^*, 3) = 4 \cdot 5 = 20$. Fig 3.(D) – Fig. 3.(F) depict the remainder of the vehicle's trip. Since no new active demands arrive, the prefix $\tau^{4*}_{\rightsquigarrow 7}$ shown in Fig. 3.(C) is followed as expected. The vehicle then waits in the location H till the arrival of new demand.

	scLTL formula	Arrival	Deadline	Priority
D_1	$\varphi_1 = \mathcal{F}B \wedge \mathcal{F}H$	$t_{D_1} = 0$	$T_1 = 9$	$p_1 = 1$
D_2	$\varphi_2 = \mathcal{F}B \wedge \mathcal{F}C$	$t_{D_2} = 0$	$T_2 = 4$	$p_2 = 1$
D_3	$\varphi_3 = \mathcal{F}E \wedge \mathcal{F}G$	$t_{D_3} = 4$	$T_3 = 7$	$p_1 = 5$

TABLE III: Example of three demands.

An example of a road network with 50 nodes is presented in Fig. 4.(A). The nodes represent points of interest and intersections, and the edges the road fragments that connect them. For simplicity, the weights of the transitions are omitted; they all were randomly generated from interval $[1, 3]$ and they are proportional to the lengths of the corresponding edges between nodes in the figure. The set of atomic propositions is $\Pi = \{\mathsf{A}, \ldots, \mathsf{H}, *\}$. Four demands gradually appear as summarized in Table IV. Fig. 4.(B)–4.(F) depicts the results for the cummulative-delay penalty function λ_c.

At time $t_1 = 0$, only D_1 is active that requires the vehicle to visit F, then A, and then D. Fig. 4.(B) shows the computed trace prefix. As expected, it corresponds to the shortest path in the road network from the current location of the vehicle through F, A, to D. Demand D_1 will be satisfied with delay 1.77 time units. Fig. 4.(C) shows the reconfigured trace prefix after demand D_2 has arrived that requires a visit to B and a visit to any of the locations marked with $*$, but G. Since D_2 is of a high priority and has a soon deadline, the path in the road network changes to first satisfy it and then continue with the satisfaction of D_1, causing extra delay. The delay of D_1 is now 6.35. Fig. 4.(D) shows the situation after demand D_3 has arrived which requires a visit to F and any location labeled with $*$. Although D_3 is of low priority, it needs to be completed in a short time and that is why the path is again reconfigured to first complete the high-priority D_2, then turn around, and proceed with satisfaction of D_3. Thus, D_1 is delayed even further. Its delay is now 13.90. Fig. 4.(E) illustrates plan after the arrival of D_4, which aims to visit D and H, in an arbitrary order. This demand has higher priority than D_1 and D_2, but it also has a generous deadline, which does not cause the vehicle to sacrifice the time of satisfaction

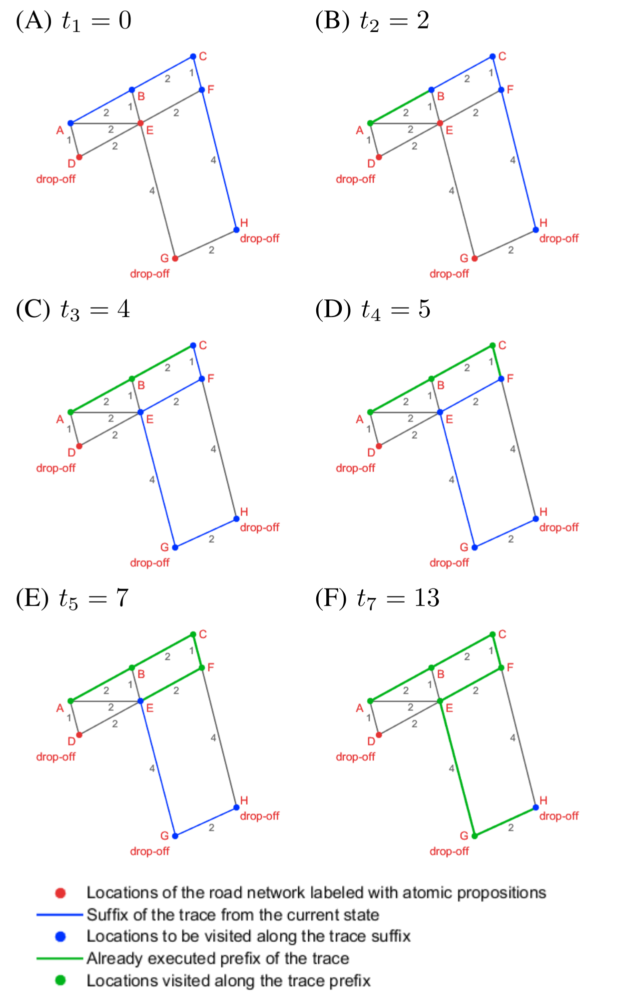

Fig. 3: Computed good trace prefixes of the desired trace suffices are in blue, already executed trace prefixes are highlighted in green. (A) Situation at time $t_1 = 0$. The computed trace fragment visits B after 2 time units, C after additional 2, i.e., demand D_2 is satisfied without any delay. Furthermore, H is visited after additional 5 time units, i.e., demand D_1 is also satisfied without delay. (B) Executed trace prefix and planned trace fragment at time $t_2 = 2$. (C) Executed trace prefix and planned trace fragment at time $t_3 = 4$. Demand D_2 has been completed and is not active any more. Demand D_3 has arrived and has a higher priority than D_1, hence, the planned trace fragment is reconfigured. Demand D_3 will be satisfied within 7 time units after its arrival, while demand D_1 after 13, i.e., with the delay of 4 time units. (D)-(E) The remainder of the execution.

of D_1 and D_2 for the sake of D_4. The computed trace only extends the previous one with the visit to H and the delay of D_1 remains unchanged. Finally, Fig. 4.(F) shows the trace of the vehicle up to the point when demands $D_1, \ldots, D_4$ were all satisfied.

The simulations were run on a laptop with 8GB memory and 2.70GHz processor. For the network with 50 nodes, the sizes of $\mathcal{P}(j)$ varied from 800 to 3200 states depending on the number of active demands at time t_j. Even on the largest $\mathcal{P}(j)$, the computation of Alg. 1 took up to 0.3 sec.

	scLTL formula	Arrival	Deadline	Priority
D_1	$\varphi_1 = \mathcal{F}(\mathsf{F} \wedge \mathcal{F}(\mathsf{A} \wedge \mathcal{F}\mathsf{D}))$	$t_{D_1} = 0$	$T_1 = 25$	$p_1 = 1$
D_2	$\varphi_2 = \mathcal{F}(B \wedge \mathcal{F}(* \wedge \neg \mathsf{G}))$	$t_{D_2} = 7$	$T_2 = 5$	$p_2 = 8$
D_3	$\varphi_3 = \mathcal{F}(E \wedge \mathcal{F}*)$	$t_{D_3} = 14$	$T_3 = 5$	$p_3 = 1$
D_4	$\varphi_4 = \mathcal{F}\mathsf{D} \wedge \mathcal{F}\mathsf{H}$	$t_{D_4} = 21$	$T_4 = 25$	$p_4 = 2$

TABLE IV: Example of four demands for the WTS in Fig. 4.(A)

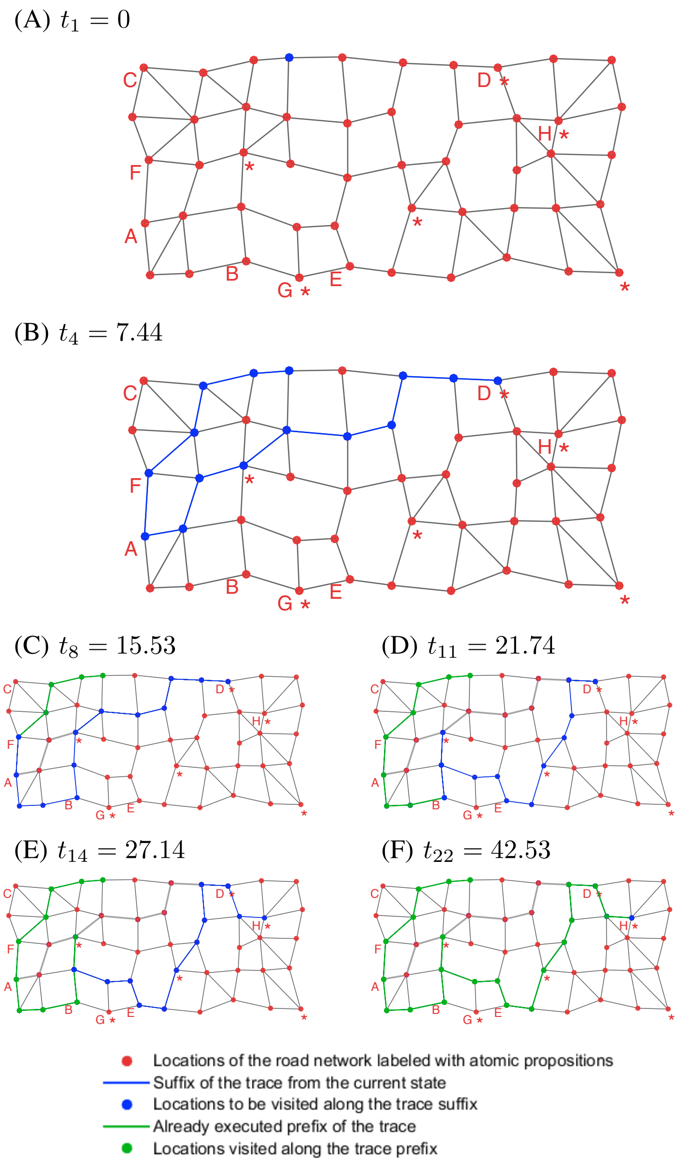

Fig. 4: An example of a road network (a WTS) with 50 nodes (states of the WTS). Some of the nodes in the figure are not labeled, meaning that the corresponding state in the WTS is not labeled with any atomic proposition. For instance for the state s in the left bottom corner, $L(s) = \emptyset$. (A) shows the network with the initial position of the vehicle in blue. (B)-(E) illustrate already executed trace prefixes in green and computed trace suffixes in blue after demands D_1 – D_4 from Table IV arrived, respectively. (F) depicts the final least-violating path of the vehicle in the network at the moment when all four demands D_1 – D_4 were satisfied.

VII. Conclusions and Future Work

We have presented a systematic way to synthesize least-violating plans for mobility-on-demand scenarios involving an autonomous vehicle in a road network that is given a set of gradually appearing demands that include sc-LTL task specification, deadline and priority. Our future work involves incorporation of time spent in the nodes of the network as well as a timed temporal logic for the task specification.

Acknowledgments

This work is supported in part by the Singapore MIT Alliance for Research and Technology (The Future of Urban Mobility project), by the MURI SMARTS program under ONR N00014-09-1051, by the US National Science Foundation grants NSF CPS-1446151 and NSF CMMI-1400167, and by the Swedish Research Council (VR).

References

[1] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2013, pp. 1–10.

[2] 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 *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2013, pp. 353–362.

[3] K. Kim and G. Fainekos, "Approximate solutions for the minimal revision problem of specification automata," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 265–271.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] V. Raman, A. Donzé, D. Sadigh, R. Murray, and S. Seshia, "Reactive synthesis from signal temporal logic specifications," in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2015, pp. 239–248.

[11] Y. Zhou, D. Maity, and J. S. Baras, "Optimal mission planner with timed temporal logic constraints," in *European Control Conference (ECC)*. IEEE, 2015.

[12] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 5010–5017.

[13] 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, Apr. 2013.

[14] E. F. M. Pavone, S. L. Smith and D. Rus, "Robotic load balancing for mobility-on-demand systems," *International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[15] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, Oct. 2001.