Optimal temporal logic planning with cascading soft constraints

Hazhar Rahmani and Jason M. O'Kane

***Abstract*— In this paper, we address the problem of temporal logic planning given both hard specifications of the robot's mission and soft preferences on the plans that achieve the mission. In particular, we consider a problem whose inputs are a transition system, a linear temporal logic (LTL) formula specifying the robot's mission, and an ordered sequence of formulas expressed in linear dynamic logic over finite traces ($\mathbf{LDL}_f$) specifying the user's preferences for how the mission should be completed. The planner's objective is to synthesize, on this transition system, an infinite trajectory that best fits the user's preferences over finite prefixes of that trajectory while nonetheless satisfying the overall objective. We describe an algorithm for this problem that constructs, from the inputs, a product automaton —which is, in fact, a special kind of state-weighted Büchi automaton— over which an optimal trajectory is synthesized. This synthesis problem is solved via reduction to the *minimax path problem in vertex weighted graphs*, which can be solved by variants of the standard algorithms for computing shortest paths in a graph or by algorithms for the all-pairs bottleneck paths problem on vertex-weighted graphs. We show the applicability of the approach via some case studies, for which we present results computed by an implementation.**

I. Introduction

As techniques to solve traditional problems in motion and path planning —those concerned with constructing a finite trajectory that starts from an initial state and ends in a goal state while avoiding obstacles [19]— have matured, the community's focus has expanded to include richer classes of problems, including those grounded in temporal logic. Temporal logic planning problems [4], [10], [27] extend the classical conception of motion planning in several ways, most notably by generalizing the constraint of merely avoiding obstacles to any user-specified temporal or spatial constraints, and by allowing the trajectory to be infinite. Progress on these problems has been achievable thanks, first, to the maturity of temporal logics, primarily Linear Temporal Logic (LTL), which allow the user to use simple, high-level logical formulas to express complex missions; and second, to the maturity of model checking techniques that enable the robot to automatically make plans for those missions.

In this paper, we consider an extension to temporal logic motion planning in which an LTL task specification is augmented with an ordered collection of *preferences*, which function as soft constraints on the robot's motion. As an illustrating example of the kind of missions our approach can handle, consider the environment in Figure 1a, inspired by the examples in Lahijanian et al. [17], in which a mobile robot in a home is tasked to patrol the kitchen (k) and bathroom (h) to detect water leakage. This environment is modeled as a transition system, as shown in Figure 1; such a model might be formed in a variety of ways, for example using the approach proposed in [20]. The patrolling mission for this robot can be characterized as "visit the bathroom and the kitchen, each one infinitely often," expressed in the LTL formula $\square\lozenge k \wedge \square\lozenge h$.

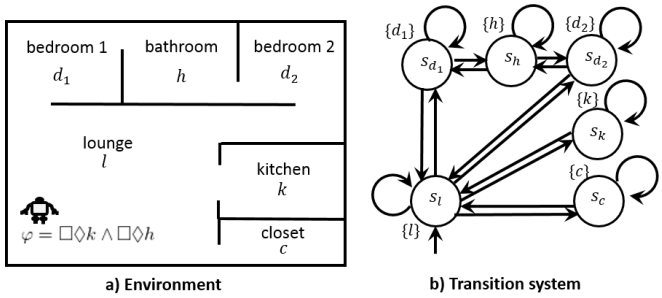

Fig. 1. **a)** An environment in which a mobile robot patrols the bathroom and the kitchen such that each must be visited infinitely often. **b)** A transition system model of this environment.

Such a mission can, in general, be satisfied by many plans. But the user may prefer, if possible, some additional properties to hold on the robot's plan. Therefore, he or she may, in advance, guide the planner by specifying their preferences. The planner must generate a plan that satisfies these preferences, if it is possible to do so. For example, the user of the robot in Fig. 1 might have preferences such as:

(a) "The robot should not go into room d_1."
(b) "The robot should not go into room d_2."
(c) "If the robot enters a bedroom d_1 or d_2, it should first put in the slippers from the closet c.

In general, it will not be possible for the robot to satisfy all of the specified preferences at once. We assume, therefore, that the user has specified the preferences in order of importance, from most important to least important.

We are interested in planning algorithms that allow the robot to complete its mission while satisfying as many of the preferences as possible, subject to the stipulation that a higher priority preference should not be sacrificed in order to satisfy lower-priority preferences. In our example, an optimal trajectory would, starting from l, travel to c, then alternate between k and h, traveling via d_2. This infinite trajectory would satisfy preferences (a) and (c), but not (b). Note that no trajectory can satisfy all three of the given preferences.

The contribution of this paper is a general formulation of this type of problem, in which the overall mission is specified as an LTL formula and the preferences are expressed as

The authors are with the Department of Computer Science and Engineering, University of South Carolina, 550 Assembly St, Columbia, SC 29201 USA. hrahmani@email.sc.edu, jokane@cse.sc.edu. This material is based upon work supported by the National Science Foundation under Grants 1513203, 1526862, and 1659514.

an ordered set of LDL_f formulas. To achieve this goal, we first review some related work (Section II), and preliminary definitions (Section III). We then provide a precise problem statement (Section IV). Then we describe an algorithm to solve this problem, based on the construction and analysis of a certain type of state-weighted Büchi automaton (Section V), along with some case studies solved by our implementation of this algorithm (Section VI) and some concluding discussion (Section VII).

II. Related work

A vibrant community of researchers has been investigating the use of formal methods to plan the motions of robots for some time. Such techniques play an established and growing role in robotics for specifying the robots' goals, verifying correctness of parts of systems, and synthesizing controllers with desired properties [5], [6], [8], [11], [12], [14], [15], [21], [24], [26]. A recent survey by Kress-Gazit, Lahijanian, and Raman reviews the current status of the field [16].

A number of papers in this area address problems that are related to the problem we address in this paper. Smith *et al.* [27] consider synthesizing an optimal trajectory satisfying an LTL formula. The generated plans are optimal in the sense of minimizing the maximum time delay between visiting states of a certain property. Our approach uses a different optimality criterion, based on user-specified preferences.

Tumova *et al.* [30] show how, for the finite horizon case, to synthesize a strategy violating the least important safety rule for the shortest amount of time. Our work differs in our focus on an infinite rather than finite horizon, which complicates the notion of the amount of time the preferences are violated.

More broadly, other research considers automatic synthesis under LTL specifications for which the system has no trajectory satisfying the specifications. Lahijanian *et al.* [17] consider synthesizing a trajectory that has "the closet distance" to a given co-safe LTL formula, quantified by assigning costs to violations of atomic propositions. Kim *et al.* [15] consider the problem of making a specification automaton for which the system has a satisfiable trajectory such that the made automaton has minimal distance to the original specification automaton. A similar problem has been addressed for probabilistic systems by Lahijanian and Kwiatkowska [18].

Alur *et al.* [1] addressed the problem of assigning priorities on infinite words. They extended the classical definition of Büchi automaton with a function assigning to each state a rank; accordingly, words which pass infinitely often through states with higher ranks have higher priorities. However, constructing such automata may be challenging for users. The problem we consider in this paper ultimately leads to finding, on a state-weighted Büchi automaton, an accepting run with the highest priority in sense of their definition.

Two other recent papers [9], [29] consider a *maximum realizability problem* that similar to our problem is a synthesis problem under hard specifications and a set of soft constraints. Their problem, however, calls for the synthesis of a transition system, rather than of a trajectory over a given transition system.

Finally, recent work by Wilde *et al.* [32] and by Nardi and Stachniss [22] shows how to elicit and exploit users' preferences about the robot's movements through its environment.

III. Preliminary definitions: Words, transition systems, LTL, Büchi automata, and LDL_f

In this section, we review the definitions of several formal tools that we utilize to define and solve the preference-guided temporal planning problem. We present these standard definitions here to help ensure that this paper remains self-contained; readers already familiar with these tools may wish to skip ahead to Section IV.

A. Words, finite and infinite

The set of all finite words over an alphabet Σ is denoted Σ^*; the set of all infinite words on the same alphabet is denoted Σ^ω. For any integer $j \geq 0$ and infinite word $w = a_0a_1a_2 \cdots \in \Sigma^\omega$, we use $w[..j]$ and $w[j..]$ to denote respectively the prefix $a_0a_1 \dots a_j$ and the postfix $a_ja_{j+1} \dots$ of w. We write $w[i..j]$ and $w[i]$ to denote $a_ia_{i+1} \cdots a_j$ and a_i, respectively. For finite words, these same four notations apply. The infinite repetition of a finite word $r \in \Sigma^*$, which is an infinite word in Σ^ω, is denoted r^ω.

B. Transition systems

Next we consider a structure to model the environment.

Definition 1: *A* transition system *is a tuple* $\mathcal{T} = (S, R, s_0, AP, L)$*, in which* S *is a finite set of states;* $R \subseteq S \times S$ *is a transition relation;* $s_0 \in S$ *is the initial state;* AP *is a set of atomic propositions; and* $L : S \to 2^{AP}$ *is a function associating atomic propositions to each state.*

A transition system may perhaps be most readily understood as a directed graph; see Figure 1. An *infinite path* on the transition system is a sequence of states $s_0s_1s_2 \cdots \in S^\omega$, starting from the initial state s_0, and for which $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. Since we are interested in infinite paths, we assume that the transition system does not have any *blocking states*, that is, states without any outgoing edges.

To each state s, the labeling function L assigns a (possibly empty) set of atomic propositions $L(s)$, describing the properties of interest that hold at that state. For any infinite path $\pi = s_0s_1s_2 \cdots \in S^\omega$, we define its *trace* as $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2) \cdots \in (2^{AP})^\omega$. The trace of a finite path is defined similarly.

C. Linear temporal logic

The overall mission for our robot is expressed as a formula expressed in LTL over the atomic propositions in the transition system.

Definition 2: *Given a set of atomic propositions* AP*, an LTL formula* φ *over* AP *is a word generated by the following grammar:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi\,\mathcal{U}\,\varphi,$$

in which $p \in AP$*;* $\neg$ *(negation),* $\wedge$ *(conjunction), and* $\vee$ *(disjunction) are Boolean operators;* $\bigcirc$ *(next),* $\Diamond$ *(eventually),* $\Box$ *(always), and* $\mathcal{U}$ *(until) are temporal operators.*

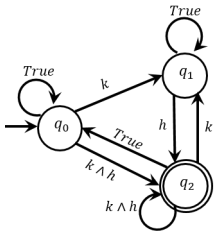

Fig. 2. A Büchi automaton A_φ for LTL formula $\varphi = \square\lozenge k \wedge \square\lozenge h$. For the transition system in Figure 1, both the formula and the corresponding automaton shown here specify trajectories that infinitely often visit both the kitchen and the bathroom.

Such a formula is interpreted as follows.

Definition 3: *Let $\sigma = A_0 A_1 A_2 \cdots \in (2^{AP})^\omega$ be a trace, and φ be an LTL formula. We say that σ* satisfies *φ, denoted $\sigma \vDash \varphi$, if the tuple (σ, φ) is related by the satisfaction relation $\vDash$, defined recursively as follows:*

- $\sigma \vDash p$ *iff* $p \in A_0$;
- $\sigma \vDash \neg\varphi$ *iff* $\sigma \nvDash \varphi$;
- $\sigma \vDash \varphi_1 \wedge \varphi_2$ *iff* $\sigma \vDash \varphi_1$ *and* $\sigma \vDash \varphi_2$;
- $\sigma \vDash \varphi_1 \vee \varphi_2$ *iff* $\sigma \vDash \varphi_1$ *or* $\sigma \vDash \varphi_2$;
- $\sigma \vDash \bigcirc\varphi$ *iff* $\sigma[1..] \vDash \varphi$;
- $\sigma \vDash \lozenge\varphi$ *iff* $\exists k \geq 0, \sigma[k..] \vDash \varphi$;
- $\sigma \vDash \square\varphi$ *iff* $\forall k \geq 0, \sigma[k..] \vDash \varphi$;
- $\sigma \vDash \varphi_1 \mathcal{U} \varphi_2$ *iff* $\exists j \geq 0, \sigma[j..] \vDash \varphi_2$ *and* $\forall 0 \leq i < j, \sigma[i..] \vDash \varphi_1$;

As an example, recall the mission of the robot in Figure 1, $\varphi = \square\lozenge k \wedge \square\lozenge h$, which describes trajectories that always eventually visit the kitchen (that is, visit the kitchen infinitely often), and also always eventually visit the bathroom. The formula $\varphi_2 = \lozenge(k \wedge \lozenge(c \wedge \lozenge\square h))$ specifies all trajectories in which the robot visits the kitchen, the closet, and the bathroom, in that order at least once, and then ultimately stays in the bathroom forever. The LTL formula $\varphi_3 = \varphi_2 \wedge \square(\neg k \vee \bigcirc\square\neg k)$ imposes the restriction of "not allowing the kitchen to be visited more than once" onto φ_2.

D. Büchi automata

Though our approach accepts the mission specification as an LTL formula, the operation of our algorithm constructs and utilizes a different representation of the mission, called a Büchi automaton, defined as follows.

Definition 4: *A* Büchi automaton *is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, in which Q is a finite set of states; Σ is an alphabet; $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation; $q_0 \in Q$ is the initial state; and $F \subseteq Q$ is a set of accepting states.*

Büchi automata may readily be visualized as directed graphs. See Figure 2. Note the structural similarities to nondeterministic finite automata.

An *infinite run* r over a Büchi automaton is an infinite sequence of states $r = q_0 q_1 q_2 \cdots \in Q^\omega$ starting at the initial state q_0, such that for all $i \geq 0$, there exists an $a \in \Sigma$ such that $(q_i, a, q_{i+1}) \in \delta$. A sequence $r = q_0 q_1 q_2 \cdots \in Q^\omega$ is a run for a word $w = a_0 a_1 \cdots \in \Sigma^\omega$ if $(q_i, a_i, q_{i+1}) \in \delta$ for each $i \geq 0$. We use $\inf(r)$ to denote the set of states that appear infinitely many times in an infinite run $r = q_0 q_1 q_2 \ldots$, that is, $\inf(r) = \{q \in Q \mid \forall i \geq 0, \exists j \geq i, q_j = q\}$. Run r is *accepting* if $\inf(r) \cap F \neq \emptyset$. The *language* of a Büchi automaton $\mathcal{A}$, denoted $\mathcal{L}_\omega(\mathcal{A})$, is defined as follows:

$$\mathcal{L}_\omega(\mathcal{A}) = \{w \in \Sigma^\omega \mid \text{there exists an accepting run } r \text{ for } w\}.$$

Büchi automata are of interest to us because they can encode our robot's LTL mission in a form more amenable to algorithmic analysis. Specifically, for any LTL formula φ over a set of atomic propositions AP, there exists a Büchi automaton $\mathcal{A}_\varphi$ with alphabet $\Sigma = 2^{AP}$ such that $\mathcal{L}_\omega(\mathcal{A}_\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \vDash \varphi\}$, that is, automaton $\mathcal{A}_\varphi$ accepts exactly all traces satisfying φ. Algorithms to perform this kind of construction of a Büchi automaton from an LTL formula are well-known [2], [13], [28], [31].

E. Linear dynamic logic over finite traces

Finally, we review the formal language, namely Linear Dynamic Logic Over Finite Traces (LDL_f), used to specify preferences as part of the problem input. The syntax of LDL_f is defined below.

Definition 5: *Given a set of atomic propositions AP, an* LDL_f formula *ψ over AP is a word generated by the following grammar, with start symbol ψ:*

$$\psi ::= p \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle\rho\rangle\psi \mid [\rho]\psi$$
$$\rho ::= \phi \mid \psi? \mid \rho; \rho \mid \rho + \rho \mid \rho^*$$

in which p is an atomic proposition in AP and ϕ is a propositional formula over the atomic propositions in AP.

Notice that LDL_f formulas can combine two types of constructions, realized in the symbols ψ and ρ. The first type, denoted ψ, is similar to an LTL formula except that it has modal operators ($\langle\rangle$ and $[]$) instead of temporal operators. The second type, denoted ρ, is called a *path expression*, in which operators ';'(concatenation), '+'(union), and '*'(Kleene star) are exactly the operators used within regular expressions. As an example path expression, consider $\rho = k; (true)^*; h$ over $AP = \{l, k, d, h, c\}$, the set of automatic propositions used for the example in Figure 1. This formula specifies all finite traces that start from the kitchen and end in the bathroom.

LDL_f uses the operators $\langle\rangle$ and $[]$ to encapsulate an RE_f expression ρ to create formulas of the form $\langle\rho\rangle\psi$ and $[\rho]\psi$. Informally, the former means that from the current time instance (the current position), with a trace satisfying ρ we can reach a time satisfying formula ψ; while the later means that from the current time instance, *all* traces satisfying ρ end in a time instance satisfying ψ.

Each LDL_f formula ψ over AP specifies a set of finite traces $\text{traces}(\psi) \subseteq (2^{AP})^*$ such that each finite trace $\hat{\sigma} \in \text{traces}(\psi)$ satisfies ψ, which is denoted $\hat{\sigma}, 0 \vDash \psi$. The semantics of LDL_f is defined as follows.

Definition 6: *Given a set of atomic propositions AP and a finite trace $\hat{\sigma} \in (2^{AP})^*$, it is inductively defined whether formula ψ is true at an instant $0 \leq i \leq |\hat{\sigma}| - 1$ of $\hat{\sigma}$—denoted $\hat{\sigma}, i \vDash \psi$—as follows:*

- $\hat{\sigma}, i \vDash p$ *iff* $p \in \hat{\sigma}[i]$

- $\hat{\sigma}, i \vDash \neg\psi$ *iff* $\hat{\sigma}, i \nvDash \psi$
- $\hat{\sigma}, i \vDash \psi_1 \wedge \psi_2$ *iff* $\hat{\sigma}, i \vDash \psi_1$ *and* $\hat{\sigma}, i \vDash \psi_2$
- $\hat{\sigma}, i \vDash \psi_1 \vee \psi_2$ *iff* $\hat{\sigma}, i \vDash \psi_1$ *or* $\hat{\sigma}, i \vee \psi_2$
- $\hat{\sigma}, i \vDash \langle\rho\rangle\psi$ *iff for some* $i \leq j \leq |\hat{\sigma}| - 1$*, it holds that* $(i, j) \in \mathcal{R}(\rho, \hat{\sigma})$ *and* $\hat{\sigma}, j \vDash \psi$
- $\hat{\sigma}, i \vDash [\rho]\psi$ *iff for all* $i \leq j \leq |\hat{\sigma}| - 1$ *such that* $(i, j) \in \mathcal{R}(\rho, \hat{\sigma})$*, it holds that* $\hat{\sigma}, j \vDash \psi$

where the relation $\mathcal{R}(\rho, s)$ *is defined recursively as follows:*

- $\mathcal{R}(\phi, s) = \{(i, i+1) \mid s, i \vDash \phi\}$
- $\mathcal{R}(\psi?, s) = \{(i, i) \mid s, i \vDash \psi\}$
- $\mathcal{R}(\rho_1; \rho_2, s) = \{(i, j) \mid$ *exists* k *such that* $(i, k) \in \mathcal{R}(\rho_1, s)$ *and* $(k, j) \in \mathcal{R}(\rho_2, s)\}$
- $\mathcal{R}(\rho_1 + \rho_2, s) = \mathcal{R}(\rho_1, s) \cup \mathcal{R}(\rho_2, s)$
- $\mathcal{R}(\rho^*, s) = \{(i, i)\} \cup \{(i, j) \mid$ *exists* k *such that* $(i, k) \in \mathcal{R}(\rho, s)$ *and* $(k, j) \in \mathcal{R}(\rho^*, s)\}$.

To illustrate the idea of LDL$_f$, recall the example preferences from Section I. Preference (a), which states that the robot should ideally not go into d_1, can be expressed by $\psi_1 = [true*]\neg d_1$, or by the formula $\psi_1' = \neg\langle true*\rangle d_1$, which is obtained via the duality rule between the operators $\langle\rangle$ and [], according to which $[\rho]\neg\psi \Leftrightarrow \neg\langle\rho\rangle\psi$ for any ψ and ρ. Similarly, preference (b) can be expressed in LDL$_f$ as $\psi_2 = [true*]\neg d_2$ and preference (c) as $\psi_3 = ([true*]\neg(d_1 \vee d_2)) \vee (\langle(\neg(d_1 \vee d_2))^*\rangle c)$. The first part of ψ_3 says that the robot does not go into any of d_1 and d_2, and the second part says that the robot should go to c before going to d_1 or d_2.

IV. Optimal preference planning

This section presents the definition of our optimal preference planning problem. An instance of this problem is defined by three elements: (1) a transition system $\mathcal{T}$, describing the connectivity of the environment in which the robot moves; (2) a goal specification, an LTL formula φ; and (3) an ordered list of n preferences, expressed as LDL$_f$ formulas $\psi_1, \psi_2, \ldots, \psi_n$ that the user would prefer to be satisfied.

We focus on preferences specified in LDL$_f$, rather than some other specification language, because of its balance of human legibility, expressivity, and efficiency of computation. One alternative would be Linear Temporal Logic for Finite Traces (LTL$_f$). However, De Giacomo and Vardi [7] showed that this language is not as powerful as it was traditionally assumed, and hence, they proposed LDL$_f$ as a strictly more expressive language with similar computational complexity to LTL$_f$.

Based on these preferences, we define for the traces of finite trajectories, a cost function $f : (2^{AP})^* \to \{0, 1, \cdots, n^n\}$ as follows:

$$f(\hat{\sigma}) = \Sigma_{\hat{\sigma} \notin \text{traces}(\psi_i)} n^{n-i}. \tag{1}$$

Notice that a trajectory that violates higher ranked preferences receives a higher cost compared to one who violates lower ranked preferences. Accordingly, among two finite trajectories, the one with lower cost is preferred.

Our plans, however, will be infinite trajectories, for the traces of which we define a new cost function $f_\omega : (2^{AP})^\omega \to \{0, 1, \cdots, n^n\}$, such that for any $\sigma \in (2^{AP})^\omega$,

$$f_\omega(\sigma) = \max_{i \geq 0} f(\sigma[..i]). \tag{2}$$

We combine these elements to form the OPP problem.

Problem: Optimal Preference Planning (OPP)

Input: A transition system $\mathcal{T}$, an LTL formula φ, and n LDL$_f$ formulas $\psi_1, \psi_2, \cdots, \psi_n$.

Output: An infinite path π over $\mathcal{T}$ such that $\text{trace}(\pi) \vDash \varphi$ and $f_\omega(\text{trace}(\pi))$ is minimized.

Algorithm 1: OptimalPreferencePlanning

Input: A transition system $\mathcal{T}$, an LTL formula φ, and a sequence of n LDL$_f$ formulas $\psi_1, \psi_2, \ldots, \psi_n$
Output: An infinite path $\pi = s_0 s_1 s_2 \cdots$ on $\mathcal{T}$ such that $trace(\pi) \vDash \varphi$ and $f_\omega(trace(\pi))$ is minimized.

for $i = 1$ *to* n **do**
  $D_i \leftarrow$ LDLf2DFA(ψ_i)
$\mathcal{F} \leftarrow$ IntegrateDFAs2Filter$(D_1, D_2, \ldots, D_n)$
$\mathcal{A}_\varphi \leftarrow$ LTL2BüchiAutomaton(φ)
$\mathcal{P} \leftarrow \mathcal{A}_\varphi \times \mathcal{T} \times \mathcal{F}$
if *not* HasAcceptingRun$(\mathcal{P})$ **then**
  return nil
$(r_1, q_f, r_2) \leftarrow$ MiniMaxAcceptingRun$(\mathcal{P})$
$\pi =$ Convert2PathOnTS(r_1, q_f, r_2)
return π

V. Algorithm description

This section describes an algorithm for the OPP problem. The algorithm operates in three phases. First, we construct a representation of the given preferences in the form of a combinatorial filter whose outputs are the costs as determined by Equations 1 and 2. Then, the algorithm forms a product graph of this filter with a Büchi automaton for the LTL goal specification, forming a certain type of state-weighted Büchi automaton the accounts for both the goal and the costs arising from violated preferences. Finally, we can generate the optimal solution trajectory on this automaton. Algorithm 1 summarizes the process.

A. Cost filters

The first step in Algorithm 1 is to construct a representation that integrates the effects of all of the LDL$_f$ formulas. This structure is defined as follows:

Definition 7: *A* filter *is a tuple* $\mathcal{F} = (V, Y, \tau, v_0, c)$*, in which* V *is a finite set of states;* Y *is an alphabet;* v_0 *is the initial state;* $\tau : V \times Y \to V$ *is a complete transition function; and* $c : V \to \{0\} \cup \mathbb{Z}^+$ *is a coloring function that assigns a non-negative cost to each state.*

This kind of filter can be viewed as a slight generalization of the standard deterministic finite automaton, in which, rather than a single distinction between accepting or non-accepting states, the filter can produce many different output values. For any word $w \in Y^*$, we write $\tau^*(v_0, w)$ to denote the state the automaton reaches when word w has been read.

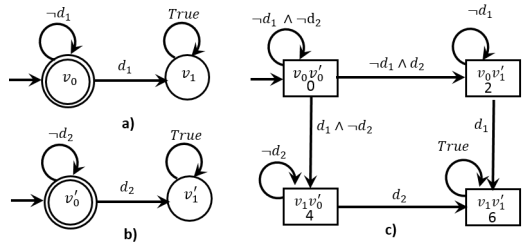

Fig. 3. **(a)** A DFA $\mathcal{D}_1$ accepting traces satisfying LDL$_f$ formula $\psi_1 = [true*]\neg d_1$ **(b)** A DFA $\mathcal{D}_2$ accepting all the traces satisfying LDL$_f$ formula $\psi_2 = [true*]\neg d_2$ **(c)** Filter $\mathcal{F}$ constructed by Lemma 1 for $\mathcal{D}_1$ and $\mathcal{D}_2$.

Note in particular that a filter partitions the set of all finite words in Y^* such that all words w for whom $c(\tau^*(v_0, w))$ are equal fall into a single equivalence class.

We leverage a filter to represent the cost function f (Equation 1), as implied by the following result.

Lemma 1: Let $\psi_1, \psi_2, \ldots, \psi_n$ be n LDL$_f$ formulas over a set of automatic proposition AP, and let f be the function in Equation 1 defined for these formulas. Then there exists a filter $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$ such that, for each $\hat{\sigma} \in (2^{AP})^*$, $c(\tau^*(v_0, \hat{\sigma})) = f(\hat{\sigma})$.

Proof: The algorithm of De Giacomo and Vardi [7] ensures that, for any LDL$_f$ formula ψ_i, we can construct a DFA $\mathcal{D}_i = (V_i, 2^{AP}, \tau_i, v_{0,i}, F_i)$, for which $\mathcal{L}(\mathcal{D}_i) = \text{traces}(\psi_i)$. From these DFAs, construct $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$ such that

- $V = V_1 \times V_2 \times \cdots \times V_n$,
- $v_0 = (v_{0,1}, v_{0,2}, \ldots, v_{0,n})$,
- for any state $(v_1, v_2, \ldots, v_n) \in V$ and any $A \in 2^{AP}$,

$$\tau((v_1, v_2, \ldots, v_n), A) = (\tau_1(v_1, A), \tau_2(v_2, A), \ldots, \tau_n(v_n, A)), \quad (3)$$

and

- for each state $(v_1, v_2, \ldots, v_n) \in V$,

$$c((v_1, v_2, \ldots, v_n)) = \Sigma_{v_i \notin F_i} n^{n-i}.$$

From the construction, it is easy to observe that $f(\hat{\sigma}) = c(\tau^*(v_0, \hat{\sigma}))$ for any $\hat{\sigma} \in (2^{AP})^*$. ■

In Algorithm 1, lines 2–4 utilize Lemma 1 to construct a cost filter $\mathcal{F}$. Figure 3 shows an example. The interpretation of this $\mathcal{F}$ is that the value assigned in $\mathcal{F}$ to the state reached by some finite trajectory in $\mathcal{T}$ is equal to the cost determined by the given preferences for that finite trajectory.

B. The product automaton

The next portion of Algorithm 1 (lines 5 and 6) forms data structure that integrates both the preferences and the goal specification. To do so, a specific product of automata is constructed according to the following definition.

Definition 8: *Given a transition system $\mathcal{T} = (S, R, s_0, AP, L)$, a Büchi automaton $\mathcal{A} = (Q, 2^{AP}, \delta, q_0, F)$, and a filter $\mathcal{F} = (V, 2^{AP}, \tau, v_0, c)$, the* product automaton $\mathcal{P} = \mathcal{A} \times \mathcal{T} \times \mathcal{F}$ *is a tuple $\mathcal{P} = (Q_\mathcal{P}, \delta_\mathcal{P}, q_{0,\mathcal{P}}, F_\mathcal{P}, w_\mathcal{P})$ in which*

1) *$Q_\mathcal{P} = Q \times S \times V$ is a finite set of states;*
2) *$q_{0,\mathcal{P}} = (q_0, s_0, v_0)$ is the initial state;*
3) *$\delta_\mathcal{P} \subseteq Q_\mathcal{P} \times Q_\mathcal{P}$ is a transition relation, such that $((q, s, v), (q', s', v')) \in \delta_P$ if and only if $(s, s') \in R$, $(q, L(s), q') \in \delta$, and $\tau(v, L(s)) = v'$;*
4) *$F_\mathcal{P} = F \times S \times V$ is a set of accepting states;*
5) *$w_\mathcal{P} : Q_\mathcal{P} \to \{0\} \cup \mathbb{Z}^+$ is a state-weighting function, such that $w_\mathcal{P}((q, s, v)) = c(v)$ for each $(q, s, v) \in Q_\mathcal{P}$.*

Notice that this product automaton $\mathcal{P}$ is itself a Büchi automaton with the trivial alphabet, but to each state of it a weight has been assigned.

The purpose of the product automaton is to encapsulate the effects of both the goal mission and the preferences. To demonstrate how, the following lemma establishes the relationship between $\mathcal{P}$ and the Büchi automaton $\mathcal{A}$ derived from the goal φ.

Lemma 2: Let $\mathcal{T}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{P}$ be the structures in Definition 8. It holds that for any accepting run $r_\mathcal{P} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2)\cdots$ over $\mathcal{P}$, the sequence $\pi = s_0 s_1 s_2 \cdots$ is a path for $\mathcal{T}$ such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$. Moreover, for any path $\pi = s_0 s_1 s_2 \cdots$ in $\mathcal{T}$ such that $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$, there exists an accepting run $r_\mathcal{P} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2)\cdots$ over $\mathcal{P}$.

Proof: For the first claim, given the definitions of $Q_\mathcal{P}$, $Q_{0,\mathcal{P}}$, and $\delta_\mathcal{P}$, the sequence $\pi = s_0 s_1 s_2 \cdots$ is a path over $\mathcal{T}$, and that the sequence $r = q_0 q_1 q_2 \cdots$ is a run for $\text{trace}(\pi) = L(s_0)L(s_1)L(s_2)\cdots$ over $\mathcal{A}$. Given the definition of $F_\mathcal{P}$ and that $r_\mathcal{P}$ is an accepting run for $\mathcal{P}$, sequence r, which is a run for π, is an accepting run over $\mathcal{A}$. Thus, $\text{trace}(\pi) \in \mathcal{L}_\omega(\mathcal{A})$.

For the second claim, given that $\pi \in \mathcal{L}_\omega(\mathcal{A})$, there exists an accepting run $r = q_0 q_1 q_2 \cdots \in Q^\omega$ for $\text{trace}(\pi)$ over $\mathcal{A}$. Also duo to that τ is a complete function, there exists a unique run $v_0 v_1 v_2 \cdots$ for $trace(\pi)$ on $\mathcal{F}$. Considering the way the components of $\mathcal{P}$ are constructed, it is easy to observe that $r_\mathcal{P} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2)\cdots$ is a run over $\mathcal{P}$. Having the definition of $F_\mathcal{P}$ and that the sequence $r = q_0 q_1 q_2 \cdots$ is an accepting run over $\mathcal{A}$, it holds that $inf(r) \cap F \neq \emptyset$, which implies that for infinitely many i's, $(q_i, s_i, v_i) \in F_\mathcal{P}$. Thus, run $r_\mathcal{P}$ is accepting over $\mathcal{P}$. ■

Lemma 2 establishes that, for any accepting run over the product automaton, there is a feasible solution for the OPP problem, and likewise for any feasible solution of OPP, there is an accepting run over the product automaton.

The key remaining question is when that solution is optimal, in the sense of minimizing the cost of preference violations. To answer that question, we first define a function $f_\mathcal{P}$ over the infinite runs of $\mathcal{P}$. Specifically, for any infinite run $r_\mathcal{P} = p_0 p_1 p_2 \cdots \in Q_\mathcal{P}^\omega$, we let

$$f_\mathcal{P}(r_\mathcal{P}) = \max_{i \geq 0} w_\mathcal{P}(p_i). \quad (4)$$

Between function $f_\mathcal{P}$ and f_ω there is a special connection, revealed by the following lemma.

Lemma 3: Given the structures $\mathcal{T}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{P}$ from Definition 8, let $\pi = s_0 s_1 s_2 \cdots$ be an infinite path over $\mathcal{T}$ such that $trace(\pi) \in \mathcal{L}_\omega(\mathcal{A})$ and $r_\mathcal{P} = (q_0, s_0, v_0)(q_1, s_1, v_1)(q_2, s_2, v_2)\cdots$ be any run over $\mathcal{P}$. Then $f_\mathcal{P}(r_\mathcal{P}) = f_\omega(\text{trace}(\pi))$.

Proof: Due to the construction of $\mathcal{P}$, the sequence $r = q_0q_1q_2\cdots$ is a run for $\mathrm{trace}(\pi)$ over $\mathcal{A}$, and $t = v_0v_1v_2\cdots$ is a run for $\mathrm{trace}(\pi)$ over $\mathcal{F}$. For any $i \geq 0$, $w_{\mathcal{P}}((q_i, s_i, v_i)) = c(v_i)$. Therefore, $\max_{i\geq 0} w_{\mathcal{P}}((q_i, s_i, v_i)) = max_{i\geq 0} c(v_i)$, which implies that $f_{\mathcal{P}}(r_{\mathcal{P}}) = f_\omega(\mathrm{trace}(\pi))$. ∎

An impact of Lemma 2 combined with Lemma 3 is that to solve OPP, we can compute the state-weighted Büchi automaton $\mathcal{P}$, and then find on $\mathcal{P}$ an accepting run $r_{\mathcal{P}}$ that minimizes $f_{\mathcal{P}}(r_{\mathcal{P}})$. Subsequently, the projection of $r_{\mathcal{P}}$ on $\mathcal{T}$ will be a solution to OPP.

C. Trajectory generation

The final phase of Algorithm 1, shown in lines 7–10, is to find an optimal accepting run over $\mathcal{P}$. As a first step, using standard algorithms for checking emptiness of the language of a Büchi automaton, one can decide in $\mathcal{O}(|Q_{\mathcal{P}}|)$ time whether the input $\mathcal{P}$ has a feasible solution or not. If not, Algorithm 1 reports an error and terminates.

However, if there is a feasible solution, then there may be many —even infinitely many— optimal solutions. Fortunately, we need to find only a single solution. The following result establishes that we need only search for solutions with a specific structure.

Lemma 4: If the OPP instance with product automaton $\mathcal{P}$ has a feasible solution, then it has an optimal solution of the form $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ where $r_1, r_2 \in Q^*_{\mathcal{P}}$, $q_f \in F_{\mathcal{P}}$, $r_1[i] \neq q_f$ for any $0 \leq i < |r_1|$, and $r_2[j] \neq q_f$ for any $0 \leq j < |r_2|$.

Proof: The idea is that from any optimal solution $r'_{\mathcal{P}}$, we can construct an optimal solution of the form $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ such that $f_{\mathcal{P}}(r'_{\mathcal{P}}) = f_{\mathcal{P}}(r_{\mathcal{P}})$. Given that $r'_{\mathcal{P}}$ is accepting—going through an accepting state infinitely many times—we have that $r'_{\mathcal{P}} = r_1q_fr_2q_fr_3\cdots$ for a $q_f \in F_{\mathcal{P}}$ such that $r_i \in Q^*_{\mathcal{P}}$ for any $i \geq 1$, and q_f does not appear in any of the r_i's. Now, from $r'_{\mathcal{P}}$, we construct $r_{\mathcal{P}} = r_1q_fr_2q_fr_2\cdots = r_1(q_fr_2)^\omega$. Clearly, run $r_{\mathcal{P}}$ is defined over the automaton given that $r_1q_fr_2q_f$ was a prefix of $r'_{\mathcal{P}}$, and that $r_{\mathcal{P}}$ is accepting given that $q_f \in inf(r_{\mathcal{P}})$. To show that $f_{\mathcal{P}}(r'_{\mathcal{P}}) = f_{\mathcal{P}}(r_{\mathcal{P}})$, observe that it holds that $f_{\mathcal{P}}(r'_{\mathcal{P}}) \leq f_{\mathcal{P}}(r_{\mathcal{P}})$ given that $r'_{\mathcal{P}}$ is optimal, and it holds that $f_{\mathcal{P}}(r'_{\mathcal{P}}) \geq f_{\mathcal{P}}(r_{\mathcal{P}})$ given the definition of $f_{\mathcal{P}}$ (Equation 4) and that $r'_{\mathcal{P}}$ contains all the states of $r_{\mathcal{P}}$. ∎

The upshot of Lemma 4 is that we can restrict our attention to solutions composed of a prefix that reaches some accepting state q_f, followed by repetitions of a cycle including q_f. The following lemma restricts this form of run further.

Lemma 5: There exists a run $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ for Lemma 4 in which both r_1 and r_2 are simple*; that is, $r_1[i] \neq r_1[j]$ for any $0 \leq i \neq j < |r_1|$, and $r_2[i] \neq r_2[j]$ for any $0 \leq i \neq j < |r_2|$.*

Proof: The idea is to remove duplicate states from r_1 and r_2 until they become simple. Then, we need to show that the new run is mapped by function $f_{\mathcal{P}}$ to the same value to which the original run was mapped. Assume that r_2 is not simple, that is, it passes through a state q at least twice. Let $r_2 = q_0q_1\cdots q_i\cdots q_jq_{j+1}\cdots q_{|r_2|-1}$ such that $q_i = q_j = q$ where i and j are respectively the first and the last positions at which q appears. There are two cases: the first one where $j < |r_2| - 1$, and the second one where $j = |r_2| - 1$. In the former case, we replace r_2 by a new sequence $q_0q_1\cdots q_iq_{j+1}\cdots q_{|r_2|-1}$, and in the later case, we replace r_2 by a new sequence $q_0q_1\cdots q_i$. In both cases, the new sequence, which is clearly a run for the automaton, passes through q exactly once. This process of removing duplicate states is continued until r_2 becomes a simple cycle. The same process applicable on r_1. Let assume that the new run is $r'_{\mathcal{P}}$. Run $r_{\mathcal{P}}$ was optimal, so it holds that $f_{\mathcal{P}}(r_{\mathcal{P}}) \leq f_{\mathcal{P}}(r'_{\mathcal{P}})$. Given the definition of $f_{\mathcal{P}}$ and that $r_{\mathcal{P}}$ contains all states appeared in $r'_{\mathcal{P}}$, it holds that $f_{\mathcal{P}}(r_{\mathcal{P}}) \geq f_{\mathcal{P}}(r'_{\mathcal{P}})$. Therefore, $f_{\mathcal{P}}(r_{\mathcal{P}}) = f_{\mathcal{P}}(r'_{\mathcal{P}})$. ∎

The combined impact of Lemma 4 and Lemma 5 is that we can use some graph algorithms to synthesize an optimal run. Specifically, if we think of $\mathcal{P}$ as a directed graph with $Q_{\mathcal{P}}$ as its vertex set and with $\delta_{\mathcal{P}}$ as its edge set, there is an optimal run $r_{\mathcal{P}} = r_1(q_f r_2)^\omega$ for which r_1 and r_2 are simple paths on this graph and, moreover,

1) path r_1q_f is a simple path from q_0 to q_f such that the maximum weight of its vertices (states) is minimum among all simple paths starting from q_0 and ending at q_f, and
2) cycle $q_fr_2q_f$ is a simple cycle starting at q_f such that the maximum weight of its vertices (states) is minimum among all simple cycles starting from q_f.

Accordingly, for each $q_f \in F_{\mathcal{P}}$, we can synthesize a run $r_{\mathcal{P},q_f} = r_1(q_fr_2)^\omega$ by finding a path r_1q_f and a cycle $q_fr_2q_f$ with those properties. Subsequently, we synthesize a solution by choosing an optimal run among those runs synthesized for all vertices (states) $q_f \in F_{\mathcal{P}}$.

With these conditions in mind, the only thing remains is to find, from a given source vertex to a given destination vertex, a path that minimizes the maximum weight of its vertices. Notice that for path r_1q_f, the source vertex is the initial state and the destination vertex is q_f, while for the cycle $q_fr_2q_f$, both the vertex and destination vertex are q_f. Finding this kind of path is the concern of the problem *minimax path for vertex-weighted graphs*, which can be solved in several different ways.

a) Path extraction via Dijkstra's algorithm: One option is to find the paths by a modified version of one of the well-known algorithms for *the shortest path problem*, including Dijkstra's algorithm. We run this algorithm for each state $q_f \in F_{\mathcal{P}}$ twice, one to find a minimax path from the initial state to q_f (path r_1q_f), and the other to find minimax path from q_f to q_f ($q_fr_2q_f$). By combining these two parts, for each accepting state q_f, we synthesize a run of the form $r_{\mathcal{P}} = r_1(q_fr_2)$ as described in Lemma 4 and Lemma 5. Among the synthesized runs for all accepting states, we choose the one with the minimum weight as the optimal solution. Hence, the total running time is $\mathcal{O}(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}|\log|Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|))$, as the time complexity of Dijkstra's algorithm to compute a minimax path is $\mathcal{O}(|Q_{\mathcal{P}}|\log|Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|)$. For sparse automata, this descends to $\mathcal{O}(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}|\log|Q_{\mathcal{P}}|))$, and even to $\mathcal{O}(|Q_{\mathcal{P}}|\log|Q_{\mathcal{P}}|)$ for sparse automata whose numbers of

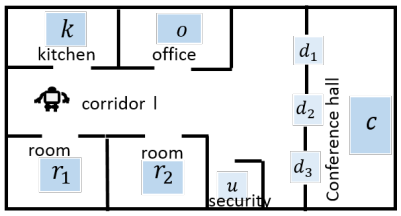

Fig. 4. An environment consisting of a corridor, an office, a kitchen, two rooms, a conference hall, and a security room.

accepting states are constant. However, this complexity is $\mathcal{O}(|Q_{\mathcal{P}}|^3)$ for dense automata with many accepting states. Therefore, for dense automata we use the second algorithm, described below.

b) Path extraction via Shapira-Yuster-Zwick: Alternatively, one can solve the minimax path problem using any algorithm for the *maximum bottleneck path* problem (also called the *maximum capacity path* or *widest path* problem), which is concerned with finding a path maximizing the minimum weight of vertices in the path. For example, one might use the algorithm of Shapira *et al.* [25]. For this purpose, the weight of each vertex p is replaced by $(\max_{p' \in Q_{\mathcal{P}}} w_{\mathcal{P}}(p')) - w_{\mathcal{P}}(p)$. By doing so, any bottleneck path in the converted graph becomes a minimax path from that source to that destination in the original graph. Finding the all-pairs bottleneck paths using this approach takes $\mathcal{O}(|Q_{\mathcal{P}}|^{2.575})$ time.

Combining these options gives the following analysis of the process of generating the optimal path in $\mathcal{P}$.

Lemma 6: For any state-weighted Büchi automaton $\mathcal{P} = (Q_{\mathcal{P}}, \delta_{\mathcal{P}}, Q_{0,\mathcal{P}}, F_{\mathcal{P}}, c_{\mathcal{P}})$, an optimal trajectory can be generated in time $\mathcal{O}(\min(|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|), |Q_{\mathcal{P}}|^{2.575}))$.

Thus, our algorithm for synthesizing an optimal accepting run on $\mathcal{P}$ is based on this lemma. This algorithm decides based on the sizes of $\delta_{\mathcal{P}}$ and $F_{\mathcal{P}}$—or more precisely, whether $|F_{\mathcal{P}}|(|Q_{\mathcal{P}}| \log |Q_{\mathcal{P}}| + |\delta_{\mathcal{P}}|) < |Q_{\mathcal{P}}|^{2.575}$)—one of the two algorithms mentioned above.

VI. Implementation and Computed Examples

We have implemented Algorithm 1 in Java. In this section, we present example instances, to illustrate its operation in solving OPP problems. The computed results were executed on an Ubuntu 16.04 computer with a 3.6GHz processor.

For the first example, called `dept`, we use the environment in Figure 4, in which a mobile robots moves within a university department consisting of a corridor, an office, a kitchen, two rooms, a security section, and a conference hall. Blue rectangles show regions of interest to the robot, consisting of c in the conference hall, d_1 the first entrance of the conference hall, d_2 the second door of the conference hall, d_3 the third door of the conference hall, r_1 in the first room, r_2 in the second room, u in the security room, k in the kitchen, and o in the office.

Suppose the robot is tasked with this non-trivial goal:

> *Take the keys from office o but before that do not go to any of d_1, d_2, r_1, r_2, or k. Additionally, visit c, r_1, r_2, and k in that order. After completing these tasks, stay in k.*

In completing this goal, the user prefers the robot to maintain several behaviors, modeled as preferences, if possible:

1) Do not go to the office.
2) Do not use door d_1.
3) Do not exit through door d_2 from the conference hall.
4) Do not use door d_3.
5) Check the security room before going to room r_1.

Details of the encoding into an instance of OPP and a solution trajectory, as computed by our implementation, and shown in the left column of Figure 5. Notice that due to the goal specification, the first preference cannot be satisfied. Also, since preferences 2, 3, and 4 cannot be satisfied together, the planner satisfies preferences 2 and 3, which have higher priority. In addition, preference 5 is also satisfied.

Figure 6 illustrates a second example, called `race`, in which an autonomous race car travels around a cyclic track. Its goal is to circle the track repeatedly, visiting the pit (p) infinitely often to refuel. States in the transition system model both the region in which the race car is moving (r_1, r_2, r_3, r_4, p) and its current speed ('f'ast, 'm'edium, or 's'low). Several preferences are associated with this robot's motion.

1) Do not accelerate or decelerate abruptly. That is, do not change speed from fast directly to slow, nor from slow directly to fast.
2) Drive slowly in the pit.
3) Do not drive fast on the curves.
4) Do drive fast on the straight parts.

An encoding and solution are shown in Figure 5.

VII. Conclusions and Future Work

This paper presented a framework for solving temporal motion planning problem, in the presence of user-specified preferences on the solution trajectories. It is essential to make the size of the product automaton as small as possible. To do so, one need to abstract away from the transition system those propositions that are irrelevant to the logical formulas of the mission and the preferences, to minimize those automata for the missions and the preferences, and also to minimize the filter. To reduce the size of the automata and the filter, one can use the well-known technique of *bisimulation minimization* (See [3], [23]).

Future work, inspired by [32], will consider learning user preferences, for the sorts of the infinite horizon scenarios addressed here. We also consider a more general case where the preferences are LTL formulas rather than LDL$_f$ formulas.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant Nos. 1526862 and 1849291.

References

[1] R. Alur, A. Kanade, and G. Weiss, "Ranking automata and games for prioritized requirements," in *Proc. International Conference on Computer Aided Verification*, 2008.

Scenario	`dept`	`race`
Illustration	Figure 4	Figure 6
Goal	$\varphi = (\neg(d_1 \vee d_2 \vee r_1 \vee r_2 \vee k)\mathcal{U}o)$ $\wedge\Diamond(c \wedge \bigcirc\Diamond(r_1 \wedge \bigcirc\Diamond(r_2 \wedge \bigcirc\Diamond\Box k)))$	$\varphi = (\Box\Diamond r_2) \wedge (\Box\Diamond p)$
Preferences	$\psi_1 = \neg\langle true^*\rangle o$	$\psi_1 = \neg\langle true^*\rangle(f \wedge \langle true\rangle s) \wedge \neg\langle true^*\rangle(s \wedge \langle true\rangle f)$
	$\psi_2 = \neg\langle true^*\rangle d_1$	$\psi_2 = [true^*](p \rightarrow s)$
	$\psi_3 = \neg\langle true^*\rangle(c \wedge \langle(\neg d_3)^*\rangle d_2)$	$\psi_3 = \neg\langle true^*\rangle(r_1 \wedge f) \wedge \neg\langle true^*\rangle(r_3 \wedge f)$
	$\psi_4 = \neg\langle true^*\rangle d_3$	$\psi_4 = [true^*]((r_2 \vee r_4) \rightarrow f))$
	$\psi_5 = \neg\langle(\neg u)^*\rangle r_1.$	
Solution	$s_l s_u s_l s_o s_l s_{d_2} s_c s_{d_3} s_l s_{r_1} s_l s_{r_2} s_l (s_k)^\omega.$	$r_1^m r_2^f r_3^m r_4^f r_1^m p^s (r_3^m r_4^f r_1^m r_2^f r_3^m r_4^f r_1^m p^s)^\omega$
Satisfied prefs	ψ_2, ψ_3, ψ_5	$\psi_1, \psi_2, \psi_3, \psi_4$
Computation time	18.7s	5.7s

Fig. 5. Formulation and results for two instances of the OPP problem.

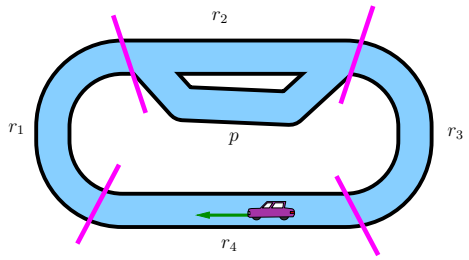

Fig. 6. A race car traveling around a circular track.

[2] T. Babiak, M. Křetínskỳ, V. Řehák, and J. Strejček, "LTL to Büchi automata translation: Fast and more deterministic," in *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2012.

[3] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[4] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *Proc. IEEE International Conference on Robotics and Automation*, 2010.

[5] S. Chinchali, S. C. Livingston, M. Pavone, and J. W. Burdick, "Simultaneous model identification and task satisfaction in the presence of temporal logic constraints," in *Proc. IEEE International Conference on Robotics and Automation*, 2016.

[6] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, "Valet parking without a valet," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[7] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces." in *IJCAI*, vol. 13, 2013, pp. 854–860.

[8] J. DeCastro, R. Ehlers, M. Rungger, A. Balkan, and H. Kress-Gazit, "Automated generation of dynamics-based runtime certificates for high-level control," *Discrete Event Dynamic Systems*, vol. 27, pp. 371–405, 2017.

[9] R. Dimitrova, M. Ghasemi, and U. Topcu, "Maximum realizability for linear temporal logic specifications," in *Proc. International Symposium on Automated Technology for Verification and Analysis*, 2018.

[10] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[11] G. E. Fainekos, A. Girard, and G. J. Pappas, "Hierarchical synthesis of hybrid controllers from temporal logic specifications," in *Proc. International Workshop on Hybrid Systems: Computation and Control*, 2007.

[12] S. Feyzabadi and S. Carpin, "Multi-objective planning with multiple high level task specifications," in *Proc. IEEE International Conference on Robotics and Automation*, 2016.

[13] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proc. International Conference on Computer Aided Verification*, 2001.

[14] M. Hekmatnejad and G. Fainekos, "Optimal multi-valued ltl planning for systems with access right levels," in *Proc. American Control Conference*, 2018.

[15] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *International Journal of Robotics Research*, vol. 34, no. 12, pp. 1515–1535, 2015.

[16] 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.

[17] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction," in *Proc. AAAI Conference on Artificial Intelligence*, 2015.

[18] M. Lahijanian and M. Kwiatkowska, "Specification revision for Markov decision processes with optimal trade-off," in *Proc. Conference on Decision and Control*, 2016.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[20] 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 *Proc. International Conference on Hybrid Systems: Computation and control*, 2013.

[21] S. Moarref and H. Kress-Gazit, "Decentralized control of robotic swarms from high-level temporal logic specifications," in *Proc. International Symposium on Multi-Robot and Multi-Agent Systems*, 2017.

[22] L. Nardi and C. Stachniss, "Experience-based path planning for mobile robots exploiting user preferences," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[23] H. Rahmani and J. M. O'Kane, "On the relationship between bisimulation and combinatorial filter reduction," in *Proc. IEEE International Conference on Robotics and Automation*, 2018.

[24] V. Raman and H. Kress-Gazit, "Explaining impossible high-level robot behaviors," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 94–104, 2013.

[25] A. Shapira, R. Yuster, and U. Zwick, "All-pairs bottleneck paths in vertex weighted graphs," *Algorithmica*, vol. 59, no. 4, pp. 621–633, 2011.

[26] C. Sloth, G. J. Pappas, and R. Wisniewski, "Compositional safety analysis using barrier certificates," in *Proc. ACM International Conference on Hybrid Systems: Computation and Control*, 2012.

[27] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3288–3293.

[28] F. Somenzi and R. Bloem, "Efficient Büchi automata from LTL formulae," in *International Conference on Computer Aided Verification*, 2000, pp. 248–263.

[29] T. Tomita, A. Ueno, M. Shimakawa, S. Hagihara, and N. Yonezaki, "Safraless LTL synthesis considering maximal realizability," *Acta Informatica*, vol. 54, no. 7, pp. 655–692, 2017.

[30] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proc. International Conference on Hybrid Systems: Computation and Control*, 2013.

[31] M. Y. Vardi and P. Wolper, "Reasoning about infinite computations," *Information and computation*, vol. 115, no. 1, pp. 1–37, 1994.

[32] N. Wilde, D. Kulić, and S. L. Smith, "Learning user preferences in robot motion planning through interaction," in *Proc. IEEE International Conference on Robotics and Automation*, 2018.