Autonomous Robots (2022) 46:149–174
https://doi.org/10.1007/s10514-021-10004-x

Learning temporal logic formulas from suboptimal demonstrations: theory and experiments

Glen Chou[1] · Necmiye Ozay[1] · Dmitry Berenson[1]

Received: 14 February 2021 / Accepted: 6 July 2021 / Published online: 30 July 2021

Abstract
We present a method for learning multi-stage tasks from demonstrations by learning the logical structure and atomic propositions of a consistent linear temporal logic (LTL) formula. The learner is given successful but potentially suboptimal demonstrations, where the demonstrator is optimizing a cost function while satisfying the LTL formula, and the cost function is uncertain to the learner. Our algorithm uses the Karush-Kuhn-Tucker (KKT) optimality conditions of the demonstrations together with a counterexample-guided falsification strategy to learn the atomic proposition parameters and logical structure of the LTL formula, respectively. We provide theoretical guarantees on the conservativeness of the recovered atomic proposition sets, as well as completeness in the search for finding an LTL formula consistent with the demonstrations. We evaluate our method on high-dimensional nonlinear systems by learning LTL formulas explaining multi-stage tasks on a simulated 7-DOF arm and a quadrotor, and show that it outperforms competing methods for learning LTL formulas from positive examples. Finally, we demonstrate that our approach can learn a real-world multi-stage tabletop manipulation task on a physical 7-DOF Kuka iiwa arm.

Keywords Learning from demonstration · Linear temporal logic · Motion planning

1 Introduction

Imagine demonstrating a multi-stage task to a robot arm delivery worker, such as finding and delivering a set of objects from a storage area to some customers (Fig. 1). How should the robot understand and generalize the demonstration? One popular method is inverse reinforcement learning (IRL), which assumes a level of optimality on the demonstrations, and aims to learn a reward function that replicates the demonstrator's behavior when optimized (Ratliff et al. 2006; Abbeel and Ng 2004; Argall et al. 2009; Ng and Russell 2000). Due to this representation, IRL works well on short-horizon goal-directed tasks, but can struggle to scale to multi-stage, constrained tasks (Krishnan et al. 2019; Vazquez-Chanlatte et al. 2018; Chou et al. 2018). Transferring reward functions across environments (e.g. from one storage area to another) can also be difficult, as IRL may overfit to aspects of the training environment. It may instead be fruitful to decouple the high- and low-level task structure, learning a logic-based temporal abstraction of the task that is valid for different environments which can combine low-level, environment-dependent skills. Linear temporal logic (LTL) is well-suited for representing this abstraction, since it can unambiguously specify high-level temporally-extended constraints (Baier and Katoen 2008) as a function of atomic propositions (APs), which can be used to describe salient low-level state-space regions. To this end, a growing community in controls and anomaly detection has focused on learning linear temporal logic (LTL) formulas to explain trajectory data. However, the vast majority of these methods require both positive and negative examples in order to regularize the learning problem. While this is acceptable in anomaly detection, where one expects to observe formula-violating trajectories, in the context of robotics, it can be unsafe to ask a demonstrator

This is one of the several papers published in Autonomous Robots comprising the Special Issue on Robotics: Science and Systems 2020.

✉ Glen Chou
gchou@umich.edu

Necmiye Ozay
necmiye@umich.edu

Dmitry Berenson
dmitryb@umich.edu

[1] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, 1301 Beal Avenue, Ann Arbor, MI 48105, USA

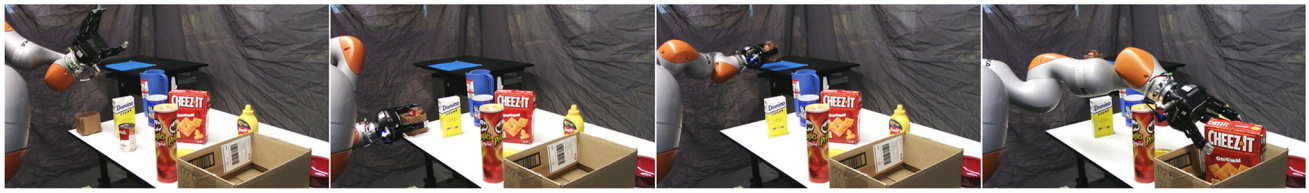

Fig. 1 Multi-stage delivery task: place the soup in an open-top box and deliver it, then deliver the Cheez-Its to a second delivery location. To avoid spills, a pose constraint is enforced while the soup is being delivered in the open-top box

to execute formula-violating behavior, such as dropping a fragile object or crashing into obstacles.

In this paper, our insight is that by assuming that demonstrators are goal-directed (i.e. that they approximately optimize an objective function that may be uncertain to the learner), we can regularize the LTL learning problem without being provided any formula-violating behavior. In particular, we learn LTL formulas which are parameterized by their high-level logical structure and low-level AP regions, and we show that to do so, it is important to consider demonstration optimality both in terms of the quality of the discrete high-level logical decisions and the continuous low-level control actions. We use the Karush-Kuhn-Tucker (KKT) optimality conditions from continuous optimization to learn the shape of the low-level APs, along with notions of discrete optimality to learn the high-level task structure. We solve a mixed integer linear program (MILP) to jointly recover LTL and cost function parameters which are consistent with the demonstrations. We make the following contributions:

1. We develop a method for time-varying, constrained inverse optimal control, where the demonstrator optimizes a cost function while respecting an LTL formula, where the parameters of the atomic propositions, formula structure, and an uncertain cost function are to be learned. We require only positive demonstrations, can handle demonstration suboptimality, and for fixed formula structure, can extract guaranteed conservative estimates of the AP regions.
2. We develop conditions on demonstrator optimality needed to learn high- and low-level task structure: AP regions can be learned with discrete feasibility, while logical structure requires various levels of discrete optimality. We develop variants of our method under these different assumptions.
3. We provide theoretical analysis of our method, showing that under mild assumptions, it is guaranteed to return the shortest LTL formula which is consistent with the demonstrations, if one exists. We also prove various results on our method's conservativeness and on formula learnability.
4. We evaluate our method on learning complex LTL formulas demonstrated on nonlinear, high-dimensional systems, show that we can use demonstrations of the same task on different environments to learn shared high-level task structure, and show that we outperform previous approaches.

Components of this work were first presented in our Robotics: Science and Systems conference paper (Chou et al. 2020a). The primary contributions specific to this journal paper include a hardware demonstration of our approach on a real-world 7-DOF manipulation task, an overview of extensions and variants of the method in Chou et al. (2020a), expanded theoretical analysis, including proofs that were omitted from Chou et al. (2020a), and expanded discussion.

2 Related work

There is extensive literature on inferring temporal logic formulas from data via decision trees (Bombara et al. 2016), genetic algorithms (Bufo et al. 2014), and Bayesian inference (Vazquez-Chanlatte et al. 2018; Shah et al. 2018). However, most of these methods require positive and negative examples as input (Camacho and McIlraith 2019; Kong et al. 2014, 2017; Neider and Gavran 2018), while our method is designed to only use positive examples. Other methods require a space-discretization (Vaidyanathan et al. 2017; Araki et al. 2019; Vazquez-Chanlatte et al. 2018), while our approach learns LTL formulas in the original continuous space. Some methods learn AP parameters, but do not learn logical structure or perform an incomplete search, relying on formula templates (Leung et al. 2019; Bakhirkin et al. 2018; Xu et al. 2019), while other methods learn structure but not AP parameters (Shah et al. 2018). Perhaps the method most similar to ours is Jha et al. (2019), which learns parametric signal temporal logic (pSTL) formulas from positive examples by fitting formulas that the data tightly satisfies. However, the search over logical structure in Jha et al. (2019) is incomplete, and tightness may not be the most informative metric given goal-directed demonstrations (cf. Sect. 9). To our knowledge, this is the first method for learning LTL

formula structure and parameters in continuous spaces on high-dimensional systems from only positive examples.

IRL (Ratliff et al. 2006; Abbeel and Ng 2004; Keshavarz et al. 2011; Englert et al. 2017; Johnson et al. 2013; Sadigh et al. 2017) searches for a reward function that replicates a demonstrator's behavior when optimized, but these methods can struggle to represent multi-stage, long-horizon tasks (Krishnan et al. 2019). To alleviate this, Krishnan et al. (2019), Ranchod et al. (2015) learn sequences of reward functions, but in contrast to temporal logic, these methods are restricted to learning tasks which can be described by a single fixed sequence. Temporal logic (Baier and Katoen 2008; Kress-Gazit et al. 2009) generalizes this, being able to represent tasks that involve more choices and can be completed with multiple different sequences. Some work (Papusha et al. 2018; Zhou and Li 2018) aims to learn a reward function given that the demonstrator satisfies a known temporal logic formula; we will learn both jointly.

Finally, there is relevant work in constraint learning. These methods generally focus on learning time-invariant constraints (Chou et al. 2018, 2019, 2020c; Calinon and Billard 2008) or a fixed sequence of task constraints (Pais et al. 2013), which our method subsumes by learning time-dependent constraints that can be satisfied by different sequences.

3 Preliminaries and problem statement

We consider discrete-time nonlinear systems

$$x_{t+1} = f(x_t, u_t, t),$$

with state $x \in \mathcal{X}$ and control $u \in \mathcal{U}$, where we denote state/control trajectories of the system as $\xi_{xu} \doteq (\xi_x, \xi_u)$.

We use linear temporal logic (LTL) (Baier and Katoen 2008), which augments standard propositional logic to express properties holding on trajectories over (potentially infinite) periods of time. In this paper, we will be given finite-length trajectories demonstrating tasks that can be completed in finite time. To ensure that the formulas we learn can be evaluated on finite trajectories, we focus on learning formulas, given in positive normal form, which are described in a parametric temporal logic similar to bounded LTL (Jha et al. 2009), and which can be written with the grammar

$$\varphi ::= p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box_{[t_1,t_2]}\varphi \mid \varphi_1 \, \mathcal{U}_{[t_1,t_2]} \varphi_2, \tag{1}$$

where $p \in \mathcal{P} \doteq \{p_i\}_{i=1}^{N_{\text{AP}}}$ are atomic propositions (APs) and N_{AP} is known to the learner. $t_1 \leq t_2$ are nonnegative integers. Here, $\neg p$ denotes the negation of atomic proposition p, the "or" operator $\varphi_1 \vee \varphi_2$ denotes the disjunction of formulas φ_1 and φ_2, the "and" operator $\varphi_1 \wedge \varphi_2$ denotes the conjunction of formulas φ_1 and φ_2, the "bounded-time always" operator $\Box_{[t_1,t_2]}\varphi$ denotes that φ "always" has to hold over the interval $[t_1, t_2]$, and the "bounded-time until" operator $\varphi_1 \, \mathcal{U}_{[t_1,t_2]} \, \varphi_2$ denotes that φ_2 must eventually hold during the interval $[t_1, t_2]$, and φ_1 must hold for all timesteps prior to that. Due to the positive normal form structure, negation can only appear directly before APs. Let the size of the grammar be $N_{\text{g}} = N_{\text{AP}} + N_{\text{o}}$, where N_{o} is the number of temporal/boolean operators in the grammar. A useful derived operator is "bounded-time eventually" $\Diamond_{[t_1,t_2]}\varphi \doteq \top \, \mathcal{U}_{[t_1,t_2]} \, \varphi$, which denotes that a formula φ eventually has to hold during the interval $[t_1, t_2]$.

In this paper, we will consider LTL formulas $\varphi(\theta^s, \theta^p)$ that are parameterized by $\theta^s \in \Theta^s$, which encode the logical and temporal structure of the formula, and by $\theta^p \doteq \{\theta_i^p\}_{i=1}^{N_{\text{AP}}}$, where $\theta_i^p \in \Theta_i^p$ defines the shape of the region where p_i holds. Furthermore, we will consider APs of the form: $x \models p_i \Leftrightarrow \mathbf{g}_i(\eta_i(x), \theta_i^p) \leq \mathbf{0}$, where $\eta_i(\cdot) : \mathcal{X} \to \mathcal{C}$ is a known nonlinear function, $\mathbf{g}_i(\cdot,\cdot) \doteq [g_{i,1}(\cdot,\cdot), \ldots, g_{i,N_i^{\text{ineq}}}(\cdot,\cdot)]^\top$ is a vector-valued parametric function, and $\mathcal{C}$ is the space in which the AP constraint is evaluated, elements of which are denoted *constraint states* $\kappa \in \mathcal{C}$.

To show how this notation maps onto a concrete robotics example, consider a 7-DOF arm. We can define the state x as the joint angles, the control u as the joint velocities, the constraint state κ as the end effector pose, and the mapping from the state to constraint state space $\eta : \mathcal{X} \to \mathcal{C} \subseteq \mathbb{R}^3$ as the forward kinematics, mapping from joint space to workspace. One possible atomic proposition is $x \models p \Leftrightarrow \mathbf{g}(\eta(x), \theta^p) = A\eta(x) - \theta^p \leq \mathbf{0}$, where $A = [I_{3\times3}, -I_{3\times3}]^\top$ and $I_{n\times n}$ is the $n \times n$ identity matrix. This atomic proposition p is satisfied if the end effector position is contained within an axis-aligned rectangle in the workspace with extents described by $\theta^p = [\bar{x}, \bar{y}, \bar{z}, -\underline{x}, -\underline{y}, -\underline{z}]$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ denote the upper extents in the x-, y-, and z-dimensions, and $\underline{x}$, $\underline{y}$, and $\underline{z}$ denote the lower extents in the x-, y-, and z-dimensions. Finally, we can write an LTL formula $\Diamond_{[t_1,t_2]}p$ to enforce that all trajectories must satisfy this workspace constraint at some point between time t_1 and t_2.

We formalize the discussion above by defining the semantics, which describe the satisfaction of an LTL formula φ by a trajectory ξ_{xu}. Specifically, we denote the satisfaction of a formula φ on a finite-duration trajectory ξ_{xu} of total duration T, evaluated at time $t \in \{1, 2, \ldots, T\}$, as $(\xi_{xu}, t) \models \varphi$. Then, the formula satisfaction is defined recursively in the formal semantics (2):

$$\begin{aligned}
&(\xi_{xu}, t) \models p_i \Leftrightarrow \mathbf{g}_i(\eta_i(x_t), \theta_i^p) \leq \mathbf{0} \\
&(\xi_{xu}, t) \models \neg p_i \Leftrightarrow \neg((\xi_{xu}, t) \models p_i) \\
&(\xi_{xu}, t) \models \varphi_1 \vee \varphi_2 \Leftrightarrow (\xi_{xu}, t) \models \varphi_1 \vee (\xi_{xu}, t) \models \varphi_2 \\
&(\xi_{xu}, t) \models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\xi_{xu}, t) \models \varphi_1 \wedge (\xi_{xu}, t) \models \varphi_2
\end{aligned}$$

$$
\begin{aligned}
&(\xi_{xu}, t) \models \Box_{[t_1,t_2]}\varphi \Leftrightarrow (t + t_1 \leq T) \\
&\quad \wedge (\forall \tilde{t} \in [t + t_1, \min(t + t_2, T)], (\xi_{xu}, \tilde{t}) \models \varphi) \\
&(\xi_{xu}, t) \models \varphi_1 \mathcal{U}_{[t_1,t_2]} \varphi_2 \Leftrightarrow (t + t_1 \leq T) \\
&\quad \wedge (\exists \tilde{t} \in [t + t_1, \min(t + t_2, T)] \text{ s.t. } (\xi_{xu}, \tilde{t}) \models \varphi_2) \\
&\quad \wedge (\forall \check{t} \in [t, \tilde{t} - 1], (\xi_{xu}, \check{t}) \models \varphi_1) \\
&(\xi_{xu}, t) \models \Diamond_{[t_1,t_2]} \varphi \Leftrightarrow (t + t_1 \leq T) \\
&\quad \wedge (\exists \tilde{t} \in [t + t_1, \min(t + t_2, T)] \text{ s.t. } (\xi_{xu}, \tilde{t}) \models \varphi) \qquad (2)
\end{aligned}
$$

We will write $\varphi \models \xi_{xu}$ as shorthand for $(\xi_{xu}, 1) \models \varphi$. We emphasize that since we consider discrete-time trajectories, a time interval $[t_1, t_2]$ is evaluated only on integer time instants $\{t_1, t_1 + 1, \ldots, t_2\}$; this is made concrete in (2).

We consider tasks that involve optimizing a parametric cost function (encoding efficiency concerns, etc.), while satisfying an LTL formula $\varphi(\theta^s, \theta^p)$ (encoding constraints for task completion):

Problem 1 (**Demonstrator's forward problem**)

$$
\begin{aligned}
\underset{\xi_{xu}}{\text{minimize}} \quad & c(\xi_{xu}, \theta^c) \\
\text{subject to} \quad & \xi_{xu} \models \varphi(\theta^s, \theta^p) \\
& \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \mathcal{C}
\end{aligned}
$$

where $c(\cdot, \theta^c)$ is a potentially non-convex cost function, parameterized by $\theta^c \in \Theta^c$. Any *a priori* known constraints are encoded in $\bar{\mathcal{S}}$, where $\bar{\eta}(\cdot)$ is known. In this paper, we encode in $\bar{\mathcal{S}}$ the system dynamics, start state, and if needed, a goal state separate from the APs.

Next, to ease notation, we will define $G_i(\kappa, \theta_i^p) \doteq \max_{m \in \{1,\ldots,N_i^{\text{ineq}}\}} \left(g_{i,m}(\kappa, \theta_i^p)\right)$. Define the subset of $\mathcal{C}$ where p_i holds/does not hold, as

$$
\mathcal{S}_i(\theta_i^p) \doteq \{\kappa \mid G_i(\kappa, \theta_i^p) \leq 0\} \qquad (3)
$$

$$
\mathcal{A}_i(\theta_i^p) \doteq \text{cl}(\{\kappa \mid G_i(\kappa, \theta_i^p) > 0\}) = \text{cl}(\mathcal{S}_i(\theta_i^p)^c) \qquad (4)
$$

To ensure that Problem 1 admits an optimum, we have defined $\mathcal{A}_i(\theta_i^p)$ to be closed; that is, states on the boundary of an AP can be considered either inside or outside. For these boundary states, our learning algorithm can automatically detect if the demonstrator intended to visit or avoid the AP (cf. Sect. 4.2).

We are given N_s demonstrations $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$ of duration T_j, which approximately solve Problem 1, in that they are feasible (satisfy the LTL formula and known constraints) and achieve a possibly suboptimal cost. Note that Problem 1 can be modeled with continuous (ξ_{xu}) and boolean decision variables (referred to collectively as $\mathbf{Z}$) (Wolff et al. 2014); the boolean variables determine the high-level plan, constraining the trajectory to obey boolean decisions that satisfy $\varphi(\theta^s, \theta^p)$, while the continuous component synthesizes a low-level trajectory implementing the plan. We will use different assumptions of demonstrator optimality on the continuous/boolean parts of the problem, depending on if θ^p (Sect. 4), θ^s (Sect. 5), or θ^c (Sect. 6) are being learned, discuss extensions and variants of these methods (Sect. 7), and discuss how these different degrees of optimality can affect the learnability of LTL formulas (Sect. 8).

Our goal is to learn the unknown structure θ^s and AP parameters θ^p of the LTL formula $\varphi(\theta^s, \theta^p)$, as well as unknown cost function parameters θ^c, given demonstrations $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$ and the *a priori* known safe set $\bar{\mathcal{S}}$.

4 Learning atomic proposition parameters (θ^p)

We develop methods for learning unknown AP parameters θ^p when the cost function parameters θ^c and formula structure θ^s are known. We first review recent results (Chou et al. 2020c) on learning time-invariant constraints via the KKT conditions (Sect. 4.1). Then, we show how the framework can be extended to learn θ^p (Sect. 4.2), and develop a method for extracting states which are guaranteed to satisfy or to violate p_i (Sect. 4.3). In all of Sect. 4, we will assume that demonstrations are *locally-optimal for the continuous component* and *feasible for the discrete component*.

4.1 Learning time-invariant constraints via KKT

Consider a simplified variant of Problem 1 that only involves always satisfying a single AP; this reduces Problem 1 to a standard trajectory optimization problem:

$$
\begin{aligned}
\underset{\xi_{xu}}{\text{minimize}} \quad & c(\xi_{xu}) \\
\text{subject to} \quad & \mathbf{g}(\eta(x), \theta^p) \leq \mathbf{0}, \quad \forall x \in \xi_{xu} \\
& \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \mathcal{C}
\end{aligned} \qquad (5)
$$

To ease notation, θ^c is assumed known in Sects. 4–5 and reintroduced in Sect. 6. Suppose we rewrite the constraints of (5) as $\mathbf{h}^k(\eta(\xi_{xu})) = \mathbf{0}$, $\mathbf{g}^k(\eta(\xi_{xu})) \leq \mathbf{0}$, and $\mathbf{g}^{\neg k}(\eta(\xi_{xu}), \theta^p) \leq \mathbf{0}$, where k and $\neg k$ group together the known and unknown constraints, respectively. Then, with Lagrange multipliers λ and ν, the KKT conditions (first-order necessary conditions for local optimality (Boyd and Vandenberghe 2004)) of the jth demonstration ξ_j^{dem}, denoted $\text{KKT}(\xi_j^{\text{dem}})$ are as written in (6),

$\text{KKT}(\xi_j^{\text{dem}})$:

Primal feasibility:

$$
\mathbf{h}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \qquad (6a)
$$

$$
\mathbf{g}^k(\eta(x_t^j)) \leq \mathbf{0}, \quad t = 1, \ldots, T_j \qquad (6b)
$$

$$
\mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) \leq \mathbf{0}, \quad t = 1, \ldots, T_j \qquad (6c)
$$

Lagrange multiplier nonnegativity:

$$\boldsymbol{\lambda}_t^{j,k} \geq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6d}$$

$$\boldsymbol{\lambda}_t^{j,\neg k} \geq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6e}$$

Complementary slackness:

$$\boldsymbol{\lambda}_t^{j,k} \odot \mathbf{g}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6f}$$

$$\boldsymbol{\lambda}_t^{j,\neg k} \odot \mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6g}$$

Stationarity:

$$\begin{aligned} &\nabla_{x_t} c(\xi_j^{\text{dem}}) + \boldsymbol{\lambda}_t^{j,k\top} \nabla_{x_t} \mathbf{g}^k(\eta(x_t^j)) \\ &\quad + \boldsymbol{\lambda}_t^{j,\neg k\top} \nabla_{x_t} \mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) \\ &\quad + \boldsymbol{\nu}_t^{j,k\top} \nabla_{x_t} \mathbf{h}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \end{aligned} \tag{6h}$$

where $\odot$ denotes elementwise multiplication. Intuitively, primal feasibility ensures that the demonstrations satisfy the learned constraint, complementary slackness encodes that a Lagrange multiplier for some constraint can only be nonzero if that constraint is active, and stationarity encodes that the cost cannot be locally improved without violating a constraint.

We vectorize the multipliers $\boldsymbol{\lambda}_t^{j,k} \in \mathbb{R}^{N_k^{\text{ineq}}}$, $\boldsymbol{\lambda}_t^{j,\neg k} \in \mathbb{R}^{N_{\neg k}^{\text{ineq}}}$, and $\boldsymbol{\nu}_t^{j,k} \in \mathbb{R}^{N_k^{\text{ineq}}}$, i.e. $\boldsymbol{\lambda}_t^{j,k} = [\lambda_{t,1}^{j,k}, \ldots, \lambda_{t,N_{\text{ineq}}^k}^{j,k}]^\top$. We drop (6a)–(6b), as they involve no decision variables. Then, we can find a constraint which makes the N_s demonstrations locally-optimal by finding a θ^p that satisfies the KKT conditions for each demonstration:

Problem 2 (**Inverse KKT problem, exact**)

$$\begin{aligned} &\text{find} \quad \theta^p, \{\boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_t^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}\}_{t=1}^{T_j}, \quad j = 1, ..., N_s \\ &\text{subject to } \{\text{KKT}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \end{aligned}$$

If the demonstrations are only approximately locally-optimal, Problem 2 may become infeasible. In this case, we can relax stationarity and complementary slackness to cost penalties:

Problem 3 (**Inverse KKT problem, suboptimal**)

$$\begin{aligned} &\underset{\theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_t^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}}{\text{minimize}} \ \sum_{j=1}^{N_s} \left(\|\text{stat}(\xi_j^{\text{dem}})\|_1 + \|\text{comp}(\xi_j^{\text{dem}})\|_1 \right) \\ &\text{subject to} \quad (6c) - (6e), \ \forall \xi_j^{\text{dem}}, \ j = 1, \ldots, N_s \end{aligned}$$

where $\text{stat}(\xi_j^{\text{dem}})$ denotes the left hand side (LHS) of Eq. (6h) and $\text{comp}(\xi_j^{\text{dem}})$ denotes the concatenated LHSs of Eqs. (6f) and (6g). Please see Sect. 7.4 for more discussion on the effect of demonstration suboptimality on learning θ^p. Note that while we have written Problems 2–3 for general constraint parameterizations, not all parameterizations admit computationally-tractable inverse KKT problems. For some constraint parameterizations (e.g. unions of boxes or ellipsoids Chou et al. 2020c), Problems 2–3 are MILP-representable[1] and can be efficiently solved; we consider such parameterizations in further detail in Sect. 4.2. In the experiments of this paper, we focus on constraints which are parameterized as axis-aligned boxes in the constraint space $\mathcal{C} \subseteq \mathbb{R}^c$, i.e. $\mathbf{g}(\eta(x), \theta^p) \leq \mathbf{0} \Leftrightarrow A\eta(x) - \theta^p \leq \mathbf{0}$, where $A = [I_{c\times c}, -I_{c\times c}]^\top$ and $\theta^p = [\bar{x}_1, \ldots, \bar{x}_c, \underline{x}_1, \ldots, \underline{x}_c]^\top$ contains the upper extents $\bar{x}_1, \ldots, \bar{x}_c$ and lower extents $\underline{x}_1, \ldots, \underline{x}_c$ of the box in each coordinate.

4.2 Modifying KKT for multiple atomic propositions

Having built intuition with the single AP case, we return to Problem 1 and discuss how the KKT conditions change in the multiple-AP setting. We first adjust the primal feasibility condition (6c). Recall from Sect. 3 that we can solve Problem 1 by finding a continuous trajectory ξ_{xu} and a set of boolean variables $\mathbf{Z}$ enforcing that $\xi_{xu} \models \varphi(\theta^s, \theta^p)$. For each ξ_j^{dem}, let $\mathbf{Z}^j(\theta_i^p) \in \{0,1\}^{N_{\text{AP}} \times T_j}$, and let the (i,t)th index $Z_{i,t}^j(\theta_i^p)$ indicate if on ξ_j^{dem}, constraint state $\kappa_t \models p_i$ for parameters θ_i^p; that is,

$$\begin{aligned} &Z_{i,t}^j(\theta_i^p) = 1 \Leftrightarrow \kappa_t \in \mathcal{S}_i(\theta_i^p), \\ &Z_{i,t}^j(\theta_i^p) = 0 \Leftrightarrow \kappa_t \in \mathcal{A}_i(\theta_i^p). \end{aligned} \tag{7}$$

Since LTL operators have equivalent boolean encodings (Wolff et al. 2014), the truth value of $\varphi(\theta^s, \theta^p)$ can be evaluated as a function of $\mathbf{Z}^j, \theta^p$, and θ^s, denoted as $\Phi(\mathbf{Z}^j, \theta^p, \theta^s)$ (we suppress θ^s, as it is assumed known for now). For example, consider the LTL formula $\varphi(\theta^s, \theta^p) = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$, which enforces that the system must eventually satisfy p_1 and eventually satisfy p_2. Two trajectories which satisfy this formula are shown in Fig. 3. We can evaluate the truth value of $\varphi(\theta^s, \theta^p)$ on ξ_j^{dem} by calculating $\Phi(\mathbf{Z}^j, \theta^p) = (\bigvee_{t=1}^{T_j} Z_{1,t}^j(\theta_1^p)) \wedge (\bigvee_{t=1}^{T_j} Z_{2,t}^j(\theta_2^p))$ (cf. Fig. 2). Boolean encodings of common temporal and logical operators can be found in Biere et al. (2006). Enforcing that $Z_{i,t}^j(\theta_i^p)$ satisfies (7) can be done with a big-M formulation and binary variables $\mathbf{s}_{i,t}^j \in \{0,1\}^{N_i^{\text{ineq}}}$ (Bertsimas and Tsitsiklis 1997):

$$\begin{aligned} &\mathbf{g}_i(\kappa_t^j, \theta_i^p) \leq M(\mathbf{1}_{N_i^{\text{ineq}}} - \mathbf{s}_{i,t}^j) \\ &\mathbf{1}_{N_i^{\text{ineq}}}^\top \mathbf{s}_{i,t}^j - N_i^{\text{ineq}} \leq M Z_{i,t}^j - M_\epsilon \\ &\mathbf{g}_i(\kappa_t^j, \theta_i^p) \geq -M \mathbf{s}_{i,t}^j \\ &\mathbf{1}_{N_i^{\text{ineq}}}^\top \mathbf{s}_{i,t}^j - N_i^{\text{ineq}} \geq -M(1 - Z_{i,t}^j) \end{aligned} \tag{8}$$

[1] This problem can also be represented and solved with satisfiability modulo theories (SMT) solvers.

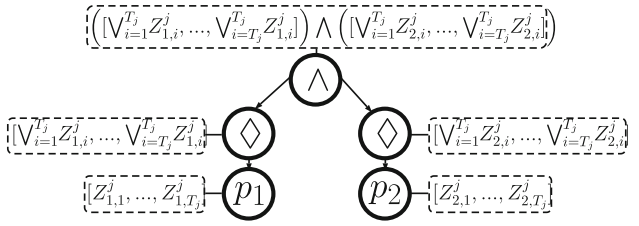

Fig. 2 A directed acyclic graph (DAG) model of the LTL formula $\varphi = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$ (eventually satisfy p_1 and eventually satisfy p_2). The DAG representation can be interpreted as a parse tree for φ (cf. Sect. 5.1). The T_j boolean values for each node represent the truth value of the formula associated with the DAG subtree when evaluated on ξ_j^{dem}, starting at times $t = 1, \ldots, T_j$, respectively. Each $\xi_j^{\text{dem}} \models \varphi$ iff the first entry at the root node, $(\bigvee_{i=1}^{T_j} Z^j_{1,i}) \bigwedge (\bigvee_{i=1}^{T_j} Z^j_{2,i})$, is true

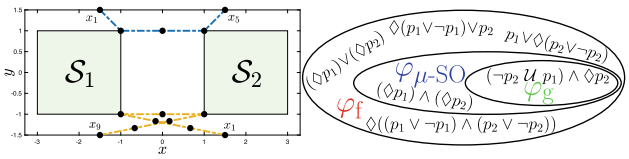

Fig. 3 **Left**: Two demonstrations which satisfy the LTL formula $\varphi = (\neg p_2 \, \mathcal{U}_{[0,T_j-1]} \, p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ (first satisfy p_1, then satisfy p_2). The demonstrations satisfy kinematic constraints and are minimizing path length while satisfying input constraints and start/goal constraints. The blue and yellow demonstrations begin at the corresponding x_1 states and end at x_5 and x_9, respectively. **Right**: Some example formulas that are consistent with φ, for various levels of discrete optimality (φ_f: discrete feasibility, φ_s: spec-optimality, φ_g: discrete global optimality)

where $\mathbf{1}_d$ is a d-dimensional vector of ones, M is a large positive number, and $M_\epsilon \in (0, 1)$. In practice, M and M_ϵ can be carefully chosen to improve the solver's performance. Note that $s^j_{i,m,t}$, the mth component of $\mathbf{s}^j_{i,t}$, encodes if κ^j_t satisfies a negated $g_{i,m}(\kappa^j_t, \theta^p_i)$, i.e. if $s^j_{i,m,t} = 1$ or 0, then κ^j_t satisfies $g_{i,m}(\kappa^j_t, \theta^p_i) \leq$ or ≥ 0. We can more compactly rewrite the constraint enforced on the demonstrations as $\mathbf{g}_i(\kappa^j_t, \theta^p_i) \odot (2\mathbf{s}^j_{i,t} - \mathbf{1}_{N_i^{\text{ineq}}}) \leq \mathbf{0}$ for each i, t; we use this form to adapt the remaining KKT conditions. While enforcing (8) is hard in general, if $\mathbf{g}_i(\kappa, \theta^p_i)$ is affine in θ^p_i for fixed κ, (8) is MILP-representable; henceforth, we assume $\mathbf{g}_i(\kappa, \theta^p_i)$ is of this form. Note that this can still describe non-convex regions in the constraint space, as the dependency on κ can be nonlinear.

As a concrete example, for the blue trajectory in Fig. 3, $\mathbf{Z}_1 = [0, 1, 0, 0, 0]$ and $\mathbf{Z}_2 = [0, 0, 0, 1, 0]$. Consider the first AP p_1. Here, since p_1 is a box in the state space, $g_{1,m}(\kappa_t, \theta^p_1) \leq 0$ can be written as $x_{t,m} - \theta^p_{1,m} \leq 0$, where $\theta^p_{1,m}$ defines the offset for the mth hyperplane that defines the boundary of the box for AP p_1. Then, $s_{1,m,t}$ determines if the polarity of halfspace constraint m is flipped at time t on the blue trajectory.

To modify complementary slackness (6g) for the multi-AP case, we note that the elementwise product in (6g) is MILP-representable:

$$\left[\boldsymbol{\lambda}^{j,\neg k}_{i,t}, \; -\mathbf{g}_i(\kappa^j_t, \theta^p_i) \odot (2\mathbf{s}^j_{i,t} - \mathbf{1}_{N_i^{\text{ineq}}})\right] \leq M\mathbf{Q}^j_{i,t}$$
$$\mathbf{Q}^j_{i,t}\mathbf{1}_2 \leq \mathbf{1}_{N_i^{\text{ineq}}} \tag{9}$$

where $\mathbf{Q}^j_{i,t} \in \{0, 1\}^{N_i^{\text{ineq}} \times 2}$. Intuitively, (9) enforces that either 1) the Lagrange multiplier is zero and the constraint is inactive, i.e. $g_{i,m}(\kappa, \theta^p_i) \in [-M, 0]$ if $s^j_{i,m,t} = 1$ or $g_{i,m}(\kappa, \theta^p_i) \in [0, M]$ if $s^j_{i,m,t} = 0, 2$) the Lagrange multiplier is nonzero and $g_{i,m}(\kappa_t, \theta^p_i) = 0$, or both; the value of $\mathbf{Q}$ toggles between these options. The stationarity condition (6h) must also be modified to consider whether a particular constraint is negated; this can be done by modifying the second line of (6h) to terms of the form $\left(\boldsymbol{\lambda}^{j,\neg k\top}_{i,t} \odot (2\mathbf{s}^j_{i,t} - 1)\right)\nabla_{x_t}\mathbf{g}^{\neg k}_i(\eta(x_t), \theta^p)$. The KKT conditions for the multi-AP case, denoted $\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$, then can be written as in (10).

$\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$:

Primal feasibility:

$$\text{Equations (6a)-(6b)}, \quad t = 1, \ldots, T_j \tag{10a}$$
$$\text{Equation (8)}, \quad i = 1, \ldots, N_{\text{AP}}, \; t = 1, \ldots, T_j \tag{10b}$$

Lagrange multiplier nonnegativity:

$$\text{Equation (6d)}, \quad t = 1, \ldots, T_j \tag{10c}$$
$$\boldsymbol{\lambda}^{j,\neg k}_{i,t} \geq \mathbf{0}, \quad i = 1, \ldots, N_{\text{AP}}, \; t = 1, \ldots, T_j \tag{10d}$$

Complementary slackness:

$$\text{Equation (6f)}, \quad t = 1, \ldots, T_j \tag{10e}$$
$$\text{Equation (9)}, \quad i = 1, \ldots, N_{\text{AP}}, \; t = 1, \ldots, T_j \tag{10f}$$

Stationarity:

$$\nabla_{x_t} c(\xi_j^{\text{dem}}) + \boldsymbol{\lambda}^{j,k\top}_t \nabla_{x_t}\mathbf{g}^k(\eta(x^j_t)) + \sum_{i=1}^{N_{\text{ineq}}} \left[\left(\boldsymbol{\lambda}^{j,\neg k\top}_{i,t} \odot (2\mathbf{s}^j_{i,t} - 1)\right)\nabla_{x_t}\mathbf{g}^{\neg k}_i(\eta(x^j_t), \theta^p_i)\right] + \boldsymbol{\nu}^{j,k\top}_t \nabla_{x_t}\mathbf{h}^k(\eta(x^j_t)) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{10g}$$

As mentioned in Sect. 3, if κ^j_t lies on the boundary of AP i, the KKT conditions will automatically determine if $\kappa^j_t \in \mathcal{S}_i(\theta^p_i)$ or $\kappa^j_t \in \mathcal{A}_i(\theta^p_i)$ based on whichever option enables $\mathbf{s}^j_{i,t}$ to take values that satisfy (10). To summarize, our approach is to (A) find $\mathbf{Z}^j$, which determines the feasibility of ξ_j^{dem} for $\varphi(\theta^s, \theta^p)$, (B) find $s^j_{i,m,t}$, which link the value of $\mathbf{Z}^j$ from the AP-containment level (i.e. $\kappa^j_t \in \mathcal{S}_i(\theta^p_i)$) to the single-constraint level (i.e. $g_{i,m}(\kappa^j_t, \theta^p_i) \leq 0$), and (C) enforce that ξ_j^{dem} satisfies the KKT conditions for the continuous optimization problem defined by θ^p and fixed values

of $\mathbf{s}_{i,t}^j$. Finally, we can write the problem of recovering θ^p for a fixed θ^s as:

Problem 4 (**Learning θ^p, for fixed template**)

$$\begin{aligned} \text{find} \quad & \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j, \ \forall i, j, t \\ \text{subject to} \quad & \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \end{aligned}$$

We can also encode prior knowledge in Problem 4, e.g. known AP labels or a prior on θ_i^p, which we discuss in Sect. 7.1.

4.3 Extraction of guaranteed learned AP

As with the constraint learning problem, the LTL learning problem is also ill-posed: there can be many θ^p which explain the demonstrations. Despite this, we can measure our confidence in the learned APs by checking if a constraint state κ is guaranteed to satisfy/not satisfy p_i for a given AP parameterization. This check is particularly useful when planning trajectories which satisfy the learned LTL formula, as we discuss shortly. Denote $\mathcal{F}_i$ as the feasible set of Problem 4, projected onto Θ_i^p (feasible set of θ_i^p). Then, we say κ is learned to be guaranteed contained in $\mathcal{S}_i(\theta_i^p)$ if for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa) \leq 0$ (i.e. $\kappa \models p_i$, for all feasible θ_i^p). Similarly, we say κ is learned to be guaranteed excluded from $\mathcal{S}_i(\theta_i^p)$ if for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa) \geq 0$. Denote by:

$$\mathcal{G}_s^i \doteq \bigcap_{\theta \in \mathcal{F}_i} \{\kappa \mid G_i(\kappa, \theta) \leq 0\} \tag{11}$$

$$\mathcal{G}_{\neg s}^i \doteq \bigcap_{\theta \in \mathcal{F}_i} \{\kappa \mid G_i(\kappa, \theta) \geq 0\} \tag{12}$$

as the sets of κ which are guaranteed to satisfy/not satisfy p_i. Having the ability to check if a constraint state lies within $\mathcal{G}_s^i$ or $\mathcal{G}_{\neg s}^i$ is useful when planning with the learned LTL formula, as we can design our plans to be robust to any uncertainty in the learned APs. For instance, if some constraint state κ on a candidate plan must satisfy/not satisfy AP i for the plan to satisfy the learned LTL formula, we can instead force κ to be contained in $\mathcal{G}_s^i$ or $\mathcal{G}_{\neg s}^i$, respectively. Then, plans generated in this fashion are guaranteed to satisfy the LTL formulas corresponding to any consistent θ^p.

Concretely, to query if κ is guaranteed to satisfy/not satisfy p_i, we can check the feasibility of the following problem:

Problem 5 (**Query containment of κ in/outside of $\mathcal{S}_i(\theta^p{}_i)$**)

$$\begin{aligned} \text{find} \quad & \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j, \ \forall i, j, t \\ \text{subject to} \quad & \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \\ & G_i(\kappa, \theta_i^p) \geq 0 \ \textbf{OR} \ G_i(\kappa, \theta_i^p) \leq 0 \end{aligned}$$

If forcing κ to (not) satisfy p_i renders Problem 5 infeasible, we can deduce that to be consistent with the KKT conditions, κ must (not) satisfy p_i. Similarly, continuous volumes of κ which must (not) satisfy p_i can be extracted by solving:

Problem 6 (**AP volume extraction**)

$$\begin{aligned} \underset{\substack{\varepsilon, \kappa_{\text{near}}, \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \\ \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j}}{\text{minimize}} \quad & \varepsilon \\ \text{subject to} \quad & \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \\ & \|\kappa_{\text{near}} - \kappa_{\text{query}}\|_\infty \leq \varepsilon \\ & G_i(\kappa_{\text{near}}, \theta_i^p) > 0 \ \textbf{OR} \ G_i(\kappa_{\text{near}}, \theta_i^p) \leq 0 \end{aligned}$$

Problem 6 searches for the largest box centered around κ_{query} contained in $\mathcal{G}_s^i/\mathcal{G}_{\neg s}^i$. An explicit approximation of $\mathcal{G}_s^i/\mathcal{G}_{\neg s}^i$ can then be obtained by solving Problem 6 for many different κ_{query}.

Finally, we note that another avenue to handle the ambiguity in the learned θ^p is to directly recover the set of all θ^p which are consistent with the demonstration, and planning to satisfy the LTL formulas associated with as many consistent θ^p as possible. This method is described in detail in Chou et al. (2020b) for time-invariant constraints, and a detailed investigation in applying this approach to temporal logic constraints is the subject of future work.

5 Learning temporal logic structure (θ^p, θ^s)

We will discuss how to frame the search over LTL structures θ^s (Sect. 5.1), the learnability of θ^s based on demonstration optimality (Sect. 5.2), and how we combine notions of discrete and continuous optimality to learn θ^s and θ^p (Sect. 5.3).

5.1 Representing LTL structure

We adapt (Neider and Gavran 2018) to search for a directed acyclic graph (DAG), $\mathcal{D}$, that encodes the structure of a parametric LTL formula and is equivalent to its parse tree, with identical subtrees merged. Hence, each node still has at most two children, but can have multiple parents. This framework enables both a complete search over length-bounded LTL formulas and encoding of specific formula templates through constraints on $\mathcal{D}$ (Neider and Gavran 2018).

Each node in $\mathcal{D}$ is labeled with an AP or operator from (1) and has at most two children; binary operators like $\wedge$ and $\vee$ have two, unary operators like $\Diamond_{[t_1,t_2]}$ have one, and APs have none (see Fig. 2). Formally, a DAG with N_{DAG} nodes, $\mathcal{D} = (\mathbf{X}, \mathbf{L}, \mathbf{R})$, can be represented as: $\mathbf{X} \in \{0,1\}^{N_{\text{DAG}} \times N_{\text{g}}}$, where $\mathbf{X}_{u,v} = 1$ if node u is labeled with element v of the grammar and 0 else, and $\mathbf{L}, \mathbf{R} \in \{0,1\}^{N_{\text{DAG}} \times N_{\text{DAG}}}$, where $\mathbf{L}_{u,v} = 1$ / $\mathbf{R}_{u,v} = 1$ if node v is the left/right child of node

u and 0 else. The DAG is enforced to be well-formed (i.e. there is one root node, no isolated nodes, etc.) with further constraints; see Neider and Gavran (2018) for more details. Since $\mathcal{D}$ defines a parametric LTL formula, we set $\theta^s = \mathcal{D}$.

As a concrete example, consider the DAG in Fig. 2. Let the grammar be $\varphi ::= p_1 \mid p_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi \mid \Diamond\varphi$, with DAG nodes labeled by $\{p_1, p_2, \vee, \wedge, \Box, \Diamond\}$. We refer to element 1 of the grammar as p_1, element 2 as p_2, element 3 as $\vee$, and so on. The DAG in Fig. 2, encoding $(\Diamond p_1) \wedge (\Diamond p_2)$, can be represented with:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where p_1, p_2, $\wedge$, the left $\Diamond$, and the right $\Diamond$, are labeled as nodes 1, 2, 3, 4, and 5 respectively. As convention, the unary operators are defined to have only left children.

To ensure that demonstration j satisfies the LTL formula encoded by $\mathcal{D}$, we introduce a satisfaction matrix $\mathbf{S}_j^{\text{dem}} \in \{0,1\}^{N_{\text{DAG}} \times T_j}$, where $\mathbf{S}^{\text{dem}}_{j,(u,t)}$ encodes the truth value of the subformula for the subgraph with root node u at time t (i.e., $\mathbf{S}^{\text{dem}}_{j,(u,t)} = 1$ iff the suffix of ξ_j^{dem} starting at time t satisfies the subformula). This can be encoded with constraints:

$$|\mathbf{S}^{\text{dem}}_{j,(u,t)} - \Phi^t_{uv}| \leq M(1 - \mathbf{X}_{u,v}) \tag{13}$$

where Φ^t_{uv} is the truth value of the subformula for the subgraph rooted at u if labeled with v, evaluated on the suffix of ξ_j^{dem} starting at time t. The truth values are recursively generated, and the leaf nodes, each labeled with some AP i, have truth values set to $\mathbf{Z}_i^j(\theta_i^p)$. Next, we can enforce that the demonstrations satisfy the formula encoded in $\mathcal{D}$ by enforcing:

$$\mathbf{S}^{\text{dem}}_{j,(\text{root},1)} = 1, \;\; j = 1, \ldots, N_s \tag{14}$$

Continuing our example, consider again the blue trajectory in Fig. 3, which satisfies the aforementioned LTL formula $(\Diamond p_1) \wedge (\Diamond p_2)$. For this trajectory, $\mathbf{S}^{\text{dem}}$ is:

$$\mathbf{S}^{\text{dem}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Note that $\mathbf{S}^{\text{dem}}_{(\text{root}=3,1)} = 1$, which reflects that the trajectory satisfies the formula. Furthermore, our method will also use synthetically-generated invalid trajectories $\{\xi^{\neg s}\}_{j=1}^{N_{\neg s}}$ (Sect. 5.3). To ensure $\{\xi^{\neg s}\}_{j=1}^{N_{\neg s}}$ do not satisfy the formula, we add more satisfaction matrices $\mathbf{S}_j^{\neg s}$ and enforce:

$$\mathbf{S}^{\neg s}_{j,(\text{root},1)} = 0, \;\; j = 1, \ldots, N_{\neg s}. \tag{15}$$

After discussing learnability, we will show how $\mathcal{D}$ can be integrated into the KKT-based learning framework in Sect. 5.3.

5.2 A detour on learnability

When learning only the AP parameters θ^p (Sect. 4), we assumed that the demonstrator chooses any *feasible* assignment of $\mathbf{Z}$ consistent with the specification, then finds a locally-optimal trajectory for those fixed $\mathbf{Z}$. Feasibility is enough if the structure θ^s of $\varphi(\theta^s, \theta^p)$ is known: to recover θ^p, we just need to find some $\mathbf{Z}$ which is feasible with respect to the known θ^s (i.e. $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$) and makes ξ_j^{dem} locally-optimal; that is, the demonstrator can choose an arbitrarily suboptimal high-level plan as long as its low-level plan is locally-optimal for the chosen high-level plan. However, if θ^s is also unknown, only using boolean feasibility is not enough to recover meaningful logical structure, as this makes any formula φ for which $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$ consistent with the demonstration, including trivially feasible formulas always evaluating to $\top$. Formally, we will refer to the set of formulas for which the demonstrations are feasible in the discrete variables and locally-optimal in the continuous variables as φ_f.

On the other end of the spectrum, we can assume the demonstrator is *globally-optimal* in solving Problem 1, i.e. there does not exist any trajectory with lower cost than the demonstration which satisfies both the specification and the known constraints. Let the set of all formulas which make the demonstrations globally-optimal be denoted φ_g. Assuming global optimality invalidates many structures in φ_f, as any formula which accepts a trajectory with a lower cost than the demonstration cannot belong in φ_g.

To make things concrete, consider again the example in Fig. 3. Assume for now that θ_1^p, θ_2^p are known. Assuming boolean feasibility, we cannot distinguish between formulas in φ_f, a subset of which are written in the Venn diagram in Fig. 3. φ_f contains trivial formulas like $\top$ or $\varphi = (\Diamond_{[0,T_j-1]} p_1) \vee (\Diamond_{[0,T_j-1]} p_2)$, Assuming global optimality, on the other hand, invalidates many structures in φ_f, i.e. the blue trajectory should not visit both $\mathcal{S}_1$ and $\mathcal{S}_2$ if $\varphi = (\Diamond_{[0,T_j-1]} p_1) \vee (\Diamond_{[0,T_j-1]} p_2)$; we achieve a lower cost by only visiting one. Using global optimality, we can distinguish between all but the formulas with globally-optimal

trajectories of equal cost (formulas in φ_g), i.e. we cannot learn the ordering constraint $(\neg p_2 \; \mathcal{U}_{[0,T_j-1]} \; p_1)$ from only the blue trajectory, as it coincides with the globally-optimal trajectory for $\varphi = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$; we need the yellow trajectory to distinguish the two.

From this discussion, we see that imposing global optimality of the demonstrations in the learning problem can be quite powerful for reducing the set of consistent LTL formulas (provided that the demonstrations are actually globally-optimal). Unfortunately, enforcing global optimality of the demonstrations in the learning problem is challenging, as it requires an exhaustive verification that there are no feasible trajectories with lower cost than the demonstrations. To overcome this challenge, we define an optimality condition that is more restrictive than feasibility and less restrictive than global optimality, and which crucially is easier to impose in learning:

Definition 1 (**Spec-optimality**) A demonstration ξ_j^{dem} is *μ-spec-optimal (μ-SO)*, where $\mu \in \mathbb{Z}_+$, if for every index set $\iota \doteq \{(i_1, t_1), ..., (i_\mu, t_\mu)\}$ in $\mathcal{I} \doteq \{\iota \mid i_m \in \{1, ..., N_{\text{AP}}\}, t_m \in \{1, ..., T_j\}, m = 1, ..., \mu\}$, at least one of the following holds:

- ξ_j^{dem} is locally-optimal after removing the constraints associated with p_{i_m} on $\kappa_{t_m}^j$, for all $(i_m, t_m) \in \iota$.
- For each index $(i_m, t_m) \in \iota$, the formula is not satisfied for a perturbed $\mathbf{Z}$, denoted $\hat{\mathbf{Z}}$, where $\hat{Z}_{i_m,t_m}(\theta_{i_m}^p) = \neg Z_{i_m,t_m}(\theta_{i_m}^p)$, for all $m = 1, \ldots, \mu$, and $\hat{Z}_{i',t'}(\theta_{i'}^p) = Z_{i',t'}(\theta_{i'}^p)$ for all $(i', t') \notin \iota$.
- ξ_j^{dem} is infeasible with respect to $\hat{\mathbf{Z}}$.

Spec-optimality enforces a level of logical optimality, evaluated *locally* around a demonstration: if a state κ_t^j on demonstration ξ_j^{dem} lies inside/outside of AP i (i.e. $G_i(\kappa_t^j, \theta_i^p) \leq 0 / \geq 0$), and the cost $c(\xi_j^{\text{dem}})$ can be lowered if that AP constraint is relaxed, then the constraint must hold to satisfy the specification. Intuitively, this means that the demonstrator does not visit/avoid APs which will needlessly increase the cost and are not needed to complete the task. Note that the conditions in Definition 1 are essentially checking how the local optimality of a demonstration changes as a result of local perturbations to the assignments of the discrete variables $\mathbf{Z}$. The three conditions in Definition 1 capture the three possibilities upon perturbing $\mathbf{Z}$: the demonstration could become infeasible if $\mathbf{Z}$ is perturbed (this is what the third condition checks), the demonstration could remain feasible but local optimality may not change (this is what the first condition checks), or the demonstration could remain feasible and no longer be locally-optimal (this is what the second condition checks). By enforcing that a demonstration is spec-optimal with respect to the formula being satisfied, we enforce that this last possibility (feasible but not locally-optimal) never occurs. We would want to enforce this, for instance, if the demonstration is assumed to be globally-optimal for the true LTL formula, because there should be no alternative assignment of $\mathbf{Z}$ which admits a feasible direction in which the demonstration cost can be improved.

Returning to the discussion on the example in Fig. 3, we will show how spec-optimality can be used to distinguish between $\varphi = (\neg p_2 \; \mathcal{U}_{[0,T_j-1]} \; p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ and $\hat{\varphi} = \Diamond_{[0,T_j-1]} p_1 \vee \Diamond_{[0,T_j-1]} p_2$ using only the blue demonstration. Specifically, we show the demonstration is 1-SO with respect to φ but not for $\hat{\varphi}$. For both formulas φ and $\hat{\varphi}$, we can see that $\mathcal{I} = \{(1, 1), \ldots, (1, 5), (2, 1), \ldots, (2, 5)\}$. Let's consider φ first. In this case, for values of $\iota \in \{(1, 1), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 5)\}$, the third condition in Definition 1 will hold, since for these time-AP pairs, the demonstration is not on the boundary of the paired AP. For $\iota \in \{(1, 2), (2, 4)\}$, the second condition in Definition 1 will hold, since perturbing $\mathbf{Z}$ at either of these time-AP pairs (from $Z_{1,2}(\theta_1^p) = 1$ to 0 or from $Z_{2,4}(\theta_2^p) = 1$ to 0) will cause φ to be not satisfied. Thus, the demonstration is spec-optimal with respect to φ. On the other hand, for $\hat{\varphi}$, again for values of $\iota \in \{(1, 1), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 5)\}$, the third condition in Definition 1 will hold. However, none of the three conditions will hold for $\iota \in \{(1, 2), (2, 4)\}$, since the demonstration will not be locally-optimal upon relaxing the constraints for either p_1 or p_2, and since $\hat{\varphi}$ only enforces that either one of $\mathcal{S}_1$ or $\mathcal{S}_2$ are visited, $\hat{\varphi}$ is still satisfied if either $Z_{1,2}(\theta_1^p)$ or $Z_{2,4}(\theta_2^p)$ is flipped to 0. Hence, the demonstration is not spec-optimal with respect to $\hat{\varphi}$.

In contrast, we can show that it is not possible to use spec-optimality to distinguish between the formulas $\varphi = (\neg p_2 \; \mathcal{U}_{[0,T_j-1]} \; p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ and $\hat{\varphi} = \Diamond_{[0,T_j-1]} p_1 \wedge \Diamond_{[0,T_j-1]} p_2$ using the yellow demonstration in Fig. 3. This follows from noting that perturbing any combination of $Z_{1,4}(\theta_1^p)$, $Z_{2,6}(\theta_2^p)$ from their values of 1 to 0 will cause both φ and $\hat{\varphi}$ to be not satisfied. Hence, the yellow demonstration is spec-optimal with respect to both φ and $\hat{\varphi}$; however, it is not globally-optimal for $\hat{\varphi}$, as the demonstrator can achieve a lower cost by first satisfying p_2 and then satisfying p_1.

We will conclude this subsection with some theoretical results which motivate how demonstration spec-optimality can be used to help the learning of LTL formulas. We first show that all globally-optimal demonstrations must also be μ-spec-optimal for the true specification, for any positive integer μ.

Lemma 1 *All globally-optimal trajectories are μ-SO.*

Proof We show that it is not possible for a demonstration ξ_j^{dem} to be globally-optimal while failing to satisfy (a), (b), and (c). If the constraints corresponding to p_{i_m} at $\kappa_{t_m}^j$ are relaxed, for some $\{(i_m, t_m)\}_{m=1}^{\mu}$, then ξ_j^{dem} can either remain locally-optimal (which means (a) is satisfied, and

happens if all the constraints are inactive or redundant) or become not locally-optimal. If ξ_j^{dem} becomes not locally-optimal for the relaxed problem (i.e. (a) is not satisfied), then at least one of the original constraints is active, implying $\bigvee_{m=1}^{\mu} \left(G_{i_m}(\kappa_{t_m}^j) = 0\right)$. In this case, one of the following holds: either (1) each $\kappa_{t_m}^j$ lies on its constraint boundary: $\bigwedge_{m=1}^{\mu} \left(G_{i_m}(\kappa_{t_m}^j) = 0\right)$, or (2) at least one κ_{t_m} does not lie on its constraint boundary. If (2) holds, then ξ_j^{dem} must be infeasible for $\hat{\mathbf{Z}}$, so (c) must be satisfied. If (1) holds, then ξ_j^{dem} is both feasible for $\hat{\mathbf{Z}}$ and not locally-optimal with respect to the relaxed constraints. Then, there exists some trajectory $\hat{\xi}_{xu}$ such that $c(\hat{\xi}_{xu}) < c(\xi_j^{\text{dem}})$, and for at least one m in $1, \ldots, \mu$, $G_{i_m}(\hat{\kappa}_{t_m}^j) > 0$, where $\hat{\kappa}_{t_m}^j$ is the constraint state at time t_m on $\hat{\xi}_{xu}$. $\hat{\xi}_{xu}$ cannot be feasible with respect to the true specification, since it makes ξ_j^{dem} not globally-optimal, so in this case (b) must hold. □

Given this result, we can use spec-optimality to vastly reduce the search space when searching for formulas which make the demonstrations globally-optimal (Sect. 5.3). To formalize this search space reduction, we prove that the set of consistent formulas shrinks as μ increases, approaching φ_f with lower values of μ and approaching φ_g with higher values of μ.

Theorem 1 (Distinguishability) *For the consistent formula sets defined in Sect. 5.2, we have* $\varphi_g \subseteq \varphi_{\tilde{\mu}\text{-SO}} \subseteq \varphi_{\hat{\mu}\text{-SO}} \subseteq \varphi_f$, *for* $\tilde{\mu} > \hat{\mu}$.

Proof $\varphi_g \subseteq \varphi_{\tilde{\mu}\text{-SO}}$, since per Lemma 1, all globally-optimal trajectories are $\tilde{\mu}$-SO. Thus, restricting Problem 8 to enforce global optimality requires more constraints than restricting Problem 8 to enforce $\tilde{\mu}$-SO. With more constraints, the feasible set of consistent formulas cannot be larger for global optimality. Similarly, as enforcing $\tilde{\mu}$-SO requires more constraints than enforcing $\hat{\mu}$-SO, the feasible set of consistent formulas cannot be larger for $\tilde{\mu}$-SO than for $\hat{\mu}$-SO. $\varphi_{\mu\text{-SO}} \subseteq \varphi_f$, since enforcing μ-SO also enforces feasibility. Thus, restricting Problem 8 to enforce μ-SO requires more constraints than the standard Problem 8. With more constraints, the feasible set of consistent formulas cannot be larger for μ-SO. □

5.3 Counterexample-guided framework

In this section, we will assume that the demonstrator returns a solution to Problem 1 which is *boundedly-suboptimal with respect to the globally optimal solution*, in that $c(\xi_j^{\text{dem}}) \leq (1+\delta)c(\xi_j^*)$, for a known suboptimality slack parameter δ, where $c(\xi_j^*)$ is the cost of the optimal solution. This is reasonable as the demonstration should be feasible (completes the task), but may be suboptimal in terms of cost (e.g. path length, etc.), and δ can be estimated from repeated demonstrations. We sketch one way δ can be estimated in Sect. 7.4.

Algorithm 1: Falsification

Input: $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \bar{\mathcal{S}}$, **Output**: $\hat{\theta}^s, \hat{\theta}^p$
$N_{\text{DAG}} \leftarrow 0, \{\xi^{\neg s}\} \leftarrow \{\}$
while $\neg$ *consistent* **do**
  $N_{\text{DAG}} \leftarrow N_{\text{DAG}} + 1$
  while *Problem 8 is feasible* **do**
    $\hat{\theta}^s, \hat{\theta}^p \leftarrow$ Problem 8$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}})$
    for $j = 1$ *to* N_s **do**
      $\xi_{xu}^j \leftarrow$ Problem 7(ξ_j^{dem})
      if Problem 7 *is feasible* **then** $\{\xi^{\neg s}\} \leftarrow \{\xi^{\neg s}\} \cup \xi_{xu}^j$
    if Problem 7 *infeasible, for all* $j = 1, \ldots, N_s$ **then**
      consistent $\leftarrow \top$; break

Under the bounded-suboptimality assumption, any trajectory ξ_{xu} satisfying the known constraints $\bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}}$ at a cost lower than the suboptimality bound, i.e. $c(\xi_{xu}) \leq c(\xi_j^{\text{dem}})/(1+\delta)$, must violate $\varphi(\theta^s, \theta^p)$ (Chou et al. 2018, 2019). We can use this to reject candidate structures $\hat{\theta}^s$ and parameters $\hat{\theta}^p$. If we can find a counterexample trajectory that satisfies the candidate LTL formula $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ at a lower cost by solving Problem 7,

Problem 7 (**Counterexample search**)

$$\begin{aligned} \text{find} \quad & \xi_{xu} \\ \text{subject to} \quad & \xi_{xu} \models \varphi(\hat{\theta}^s, \hat{\theta}^p) \\ & \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}}(\xi_j^{\text{dem}}) \subseteq \mathcal{C} \\ & c(\xi_{xu}) < c(\xi_j^{\text{dem}})/(1+\delta) \end{aligned}$$

then $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ cannot be consistent with the demonstration. Thus, we can search for a consistent $\hat{\theta}^s$ and $\hat{\theta}^p$ by iteratively proposing candidate $\hat{\theta}^s$ / $\hat{\theta}^p$ by solving Problem 8 (a modified version of Problem 4, which we will discuss shortly) and searching for counterexamples that can prove the parameters are invalid/valid; this is summarized in Algorithm 1. Heuristics on the falsification loop are discussed in Sect. 7.3.

We note that the structure of the falsification loop in Algorithm 1 is crucial for enforcing that the returned LTL formula makes the demonstrations globally-optimal (or boundedly-suboptimal), since as discussed in Sect. 5.2, it is challenging to encode global optimality directly. As a result, we will rely on encoding conditions that are weaker than global optimality but which can be efficiently enforced, proposing LTL formulas which make the demonstration feasible or spec-optimal (see Problem 8). Thus, the loop is needed to reject formulas which make the demonstrations feasible or spec-optimal but not globally-optimal, in order to ensure that the formula that is eventually returned makes the demonstrations globally-optimal. We now discuss in detail the core components of Algorithm 1: counterexample generation, addressed in Prob-

lem 7, and a combined search for θ^p and θ^s, addressed in Problem 8).

Counterexample generation: We propose different methods to solve Problem 7 based on the dynamics. For piecewise affine systems, Problem 7 can be solved directly as a MILP (Wolff et al. 2014). However, the LTL planning problem for general nonlinear systems is challenging (Li and Fu 2017; Fu et al. 2017). Probabilistically-complete sampling-based methods (Li and Fu 2017; Fu et al. 2017) or falsification tools (Annpureddy et al. 2011) can be applied, but can be slow on high-dimensional systems. For simplicity and speed, we solve Problem 7 by finding a trajectory $\hat{\xi}_{xu} \models \varphi(\hat{\theta}^s, \hat{\theta}^p)$ and boolean assignment $\mathbf{Z}$ for a kinematic approximation of the dynamics via solving a MILP, then warm-start the nonlinear optimizer with $\hat{\xi}_{xu}$ and constrain it to be consistent with $\mathbf{Z}$, returning some ξ_{xu}. We use IPOPT (Wächter and Biegler 2006) and TrajOpt (Schulman et al. 2014) to solve these nonlinear optimization problems for the simulation and hardware experiments, respectively. If $c(\xi_{xu}) < c(\xi_j^{\text{dem}})/(1+\delta)$, then we return, otherwise, we generate a new $\hat{\xi}_{xu}$. Whether this method returns a valid counterexample depends on if the nonlinear optimizer converges to a feasible solution; hence, this approach is not complete. However, we show that it works well in practice (see Sects. 9–10); moreover, the optimal sampling-based planning approaches (e.g. Li and Fu 2017) can always be used as a complete alternative, at the expense of higher computation time.

Unifying parameter and structure search: When both θ^p and θ^s are unknown, they must be jointly learned due to their interdependence: learning the structure involves finding an unknown boolean function of θ^p, parameterized by θ^s, while learning the AP parameters θ^p requires knowing which APs were selected or negated, determined by θ^s. This can be done by combining the KKT (10) and DAG constraints (13)–(15) into a single MILP, which can then be integrated into Algorithm 1:

Problem 8 (**Combined search for $\boldsymbol{\theta^p}$, $\boldsymbol{\theta^s}$**)

$$\text{find} \quad \mathcal{D}, \mathbf{S}_j^{\text{dem}}, \mathbf{S}_j^{\neg s}, \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j, \quad \forall i, j, t$$

$$\text{s.t.} \quad \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s}$$

topology constraints (except single root) for $\mathcal{D}$

Equation (13), $j = 1, \ldots, N_s$

Equation (14), $j = 1, \ldots, N_s$

Equation (15), $j = 1, \ldots, N_{\neg s}$

In Problem 8, since (1) the $\mathbf{Z}_i^j(\theta_i^p)$ at the leaf nodes of $\mathcal{D}$ are constrained via (8) to be consistent with θ^p and ξ_j^{dem} and (2) the formula defined by $\mathcal{D}$ is constrained to be satisfied for the $\mathbf{Z}$ via (13), the low-level demonstration ξ_j^{dem} must be feasible for the overall LTL formula defined by the DAG, i.e. $\varphi(\theta^s, \theta^p)$, where $\theta^s = \mathcal{D}$. $\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$ then chooses AP parameters θ^p to make ξ_j^{dem} locally-optimal for the continuous optimization induced by a fixed realization of boolean variables. Overall, Problem 8 finds a pair of θ^p and θ^s which makes ξ_j^{dem} locally-optimal for a fixed $\mathbf{Z}^j$ which is *feasible* for $\varphi(\theta^s, \theta^p)$, i.e. $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$, for all j. To also impose the spec-optimality conditions (Definition 1), we can add these constraints to Problem 8:

$$\mathbf{S}_{j,(\text{root},1)}^{\text{dem},\hat{Z}_n^j} \leq b_{nj}^1 \tag{16a}$$

$$\|\boldsymbol{\lambda}_{i_m,t_m}^{j,\neg k\,\top} \nabla_{x_t} \mathbf{g}_{i_m}^{\neg k}(\eta(x_t^j), \theta_{i_m}^p)\| \leq M(1 - b_{nj}^2), \quad m = 1, \ldots, \mu \tag{16b}$$

$$\mathbf{g}_{i_m}^{\neg k}(\eta(x_t^j), \theta_{i_m}^p) \geq -M(1 - \mathbf{e}_{nm}^j), \ m = 1, \ldots, \mu \tag{16c}$$

$$\mathbf{1}_{N_{\text{ineq}}^{i_m}}^{\top} \mathbf{e}_{nm}^j \geq \hat{Z}_{i_m t_m}^j(\theta_{i_m}^p) - b_{nj}^3, \ m = 1, \ldots, \mu \tag{16d}$$

$$\mathbf{g}_{i_m}^{\neg k}(\eta(x_t^j), \theta_{i_m}^p) \leq M(\hat{Z}_{i_m,t_m}^j + b_{nj}^3) \tag{16e}$$

$$b_{nj}^1 + b_{nj}^2 + b_{nj}^3 \leq 1, \quad \mathbf{b}_{nj} \in \{0,1\}^3, \quad \mathbf{e}_{nm}^j \in \{0,1\}^{N_{\text{ineq}}^{i_m}} \tag{16f}$$

for $n = 1, \ldots, |\mathcal{I}|$, where $\mathbf{S}_j^{\text{dem},\hat{Z}_n^j}$ is the satisfaction matrix for ξ_j^{dem} where the leaf nodes are perturbed to take the values of $\hat{Z}_n^j$, where n indexes an $\iota \in \mathcal{I}$. (16a) models the case when the formula is not satisfied, (16b) models when ξ_j^{dem} remains locally-optimal upon relaxing the constraint (zero stationarity contribution), and (16c)–(16e) model the infeasible case. Generally, without spec-optimality, the falsification loop in Algorithm 1 will need to eliminate more formulas on the way to finding a formula which makes the demonstrations globally-optimal. We conclude this section with some remarks on spec-optimality and the falsification loop:

Remark 1 If $\mu = 1$, the infeasibility constraints (16c)–(16e) can be ignored (since together with (16a), they are redundant), and we can modify (16f) to $b_{nj}^1 + b_{nj}^2 \leq 1$, $\mathbf{b}_{nj} \in \{0,1\}^2$.

Remark 2 It is only useful to enforce spec-optimality on index pairs $(i_1, t_1), \ldots, (i_\mu, t_\mu)$ where $G_{i_m}(\kappa_{i_m}^j, \theta_{i_m}^p) = 0$ for all $m = 1, \ldots, \mu$; otherwise the infeasibility case automatically holds. If θ^p is unknown, we won't know *a priori* when this holds, but if θ^p are (approximately) known, we can pre-process so that spec-optimality is only enforced for salient $\iota \in \mathcal{I}$.

Remark 3 We can interpret μ as a tuning knob for shifting the computation between the falsification loop and Problem 8;

imposing a larger μ can potentially rule out more formulas at the cost of adding additional constraints and decision variables to Problem 8.

Remark 4 Problem 8 with spec-optimality constraints (16) can be used to directly search for a $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which can be satisfied by visiting a set of APs in any order (e.g. surveillance-type tasks) without using the loop in Algorithm 1, since (16) directly enforces that any AP (1-SO) or a set of APs (μ-SO) which were visited and which prevent the trajectory cost from being lowered must be visited for any candidate
$\varphi(\hat{\theta}^s, \hat{\theta}^p)$.

6 Learning cost function parameters (θ^p, θ^s, θ^c)

If θ^c is unknown, it can be learned by modifying KKT_{LTL} to also consider θ^c in the stationarity condition: all terms containing $\nabla_{\xi_{xu}} c(\xi_j^{\text{dem}})$ should be modified to $\nabla_{\xi_{xu}} c(\xi_j^{\text{dem}}, \theta^c)$. When $c(\cdot,\cdot)$ is affine in θ^c for fixed ξ_j^{dem}, the stationarity condition is representable with a MILP constraint. However, the falsification loop in Algorithm 1 requires a fixed cost function in order to judge if a trajectory is a counterexample. Thus, one valid approach is to first solve Problem 8, searching also for θ^c, then fixing θ^c, and running Algorithm 1 for the fixed θ^c. Specifically, the approach is the same as Algorithm 1, apart from an additional outer while loop, where candidate θ^c are selected. We formally write this procedure in Algorithm 2, where we refer to the Problem 8 variant that searches over θ^c as Problem 8′, and to the Problem 7 variant that takes in θ^c as input as Problem 7′. Upon the failure of a θ^c to yield a consistent θ^p and θ^s, the θ^c is added into a set of cost parameters for Problem 8 to avoid, Θ^c_{av}. The avoidance condition can be implemented with integer constraints, i.e. $|\theta^c_i - \hat{\theta}^c_i| \geq \varepsilon_{\text{av}} - (1 - z^i_{\text{av}})$, $\sum_i z^i_{\text{av}} \geq 1$, for $i = 1, \ldots, |\theta_c|$ and for binary variables z^i_{av}. Here, ε_{av} is a hyperparameter that defines the size of an infinity-norm ball around $\hat{\theta}_i$ which should be avoided in future iterations. One can also achieve a similar effect without this hyperparameter by adding an objective function $\max_{\theta^c} \|\theta^c - \hat{\theta}^c_i\|_\infty$ to Problem 8′, which is MILP-representable.

Note that this procedure either eventually returns an LTL formula consistent with the fixed θ^c, or Algorithm 1 becomes infeasible, and a new θ^c must be generated and Algorithm 1 rerun. This is guaranteed to eventually return a set of θ^c, θ^s, and θ^p which make each ξ_j^{dem} globally-optimal with respect to $c(\xi_{xu}, \theta^c)$ under $\varphi(\theta^s, \theta^p)$. However, it may require iterating through an infinite number of candidate θ^c and hence is not guaranteed to terminate in finite time (Corollary 3). Nonetheless, we note that for a certain class of formulas (Remark 4), a consistent set of θ^c, θ^s, and θ^p can be recovered in one shot.

Algorithm 2: Falsification, unknown cost function

Input: $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$, $\bar{\mathcal{S}}$, **Output**: $\hat{\theta}^s, \hat{\theta}^p, \hat{\theta}^c$
$N_{\text{DAG}} \leftarrow 0$, $\{\xi^{\neg s}\} \leftarrow \{\}$, $\Theta^c_{\text{av}} \leftarrow \{\}$
while *true* **do**
  $\hat{\theta}^s, \hat{\theta}^p, \hat{\theta}^c \leftarrow$ Problem 8′$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}}, \Theta^c_{\text{av}})$
  while ¬ *consistent* **do**
    $N_{\text{DAG}} \leftarrow N_{\text{DAG}} + 1$
    while *Problem 8 is feasible* **do**
      $\hat{\theta}^s, \hat{\theta}^p \leftarrow$ Problem 8$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}}, \hat{\theta}^c)$
      for $j = 1$ *to* N_s **do**
        $\xi^j_{xu} \leftarrow$ Problem 7′$(\xi_j^{\text{dem}}, \hat{\theta}^c)$
        if *Problem 7′ is feasible* **then**
          $\{\xi^{\neg s}\} \leftarrow \{\xi^{\neg s}\} \cup \xi_{xu}$
      if *Problem 7′ infeasible, for all* $j = 1, \ldots, N_s$ **then**
        consistent $\leftarrow \top$; break
  if *consistent* **then** return;
  else $\Theta^c_{\text{av}} \leftarrow \Theta^c_{\text{av}} \cup \hat{\theta}^c$; break;

7 Method extensions, variants, and discussion

In this section, we discuss some extensions and variants of our approach which can improve learning (Sect. 7.1) and computational performance (Sects. 7.2, 7.3). Finally, we discuss the effect of suboptimality on the learning procedure and how the suboptimality slack parameter δ can be estimated (Sect. 7.4).

7.1 Encoding prior knowledge

In some situations, we may have some *a priori* knowledge on the atomic propositions, e.g. which labels correspond to which atomic proposition regions, or a rough estimate of the AP parameters θ^p. We describe how this knowledge can be integrated into our method.

Known labels: We have assumed that the demonstrations only include state/control trajectories and not the AP labels; this can lead to ambiguity as to which $\mathcal{S}$ should be assigned to which proposition p_i. For example, consider the example in Fig. 3 (left), where the aim is to recover $\varphi(\theta^p) = \Diamond\mathcal{S}_1(\theta^p_1) \vee \Diamond\mathcal{S}_2(\theta^p_2)$. The KKT conditions will imply that the demonstrator had to visit two boxes and their locations, but not whether the left box should be labeled $\mathcal{S}_1$ or $\mathcal{S}_2$. However, in some settings it may be reasonable that the labels for each AP are provided, e.g. for an AP which requires a robot arm to grasp an object, we might have sensor data determining if the object has been grasped. In this case, we can incorporate this by simply constraining $\mathbf{Z}^j_i(\theta^p_i)$ to be the labels; this then removes the ambiguity mentioned earlier.

Prior knowledge on θ^p: In some settings, we may have a rough idea of θ^p, e.g. as noisy bounding boxes from a vision system. We might then want to avoid deviating from these nominal parameters, denoted θ^p_{nom}, or restrict θ^p to some region around θ^p_{nom}, denoted $\Theta_{i,\text{nom}}$, subject to the KKT conditions holding. This can be done by adding $\sum_{j=1}^{N_{\text{AP}}} \|\theta^p_i - \theta^p_{i,\text{nom}}\|_1$ as an objective or $\theta^p_{i,\text{nom}} \in \Theta_{i,\text{nom}}$ as a constraint to Problem 4 instead of simply solving Problem 4 as a feasibility problem.

7.2 Faster reformulations for the falsification loop

A shortcoming of Algorithm 1 is that it can be computationally intensive. This is primarily due to Problem 8, which is a mixed-integer program that contains many binary decision variables, including the DAG structure variables $(\mathbf{X}, \mathbf{L}, \mathbf{R})$ and variables $\mathbf{Q}$ and $\mathbf{Z}$ which are needed to learn the continuous parameters θ^p. While Problem 8 can still be solved for examples of moderate size (see the results in Sect. 9), we observe that its computation time can become unrealistic for examples with very long LTL formulas (i.e. a large search space for $(\mathbf{X}, \mathbf{L}, \mathbf{R})$). Intuitively, increasing the dimensionality of $(\mathbf{X}, \mathbf{L}, \mathbf{R})$ combinatorially increases the number of possible assignments, which can cause the optimizer to struggle to find a feasible solution in a reasonable timeframe.

To address these computational challenges, we propose a reformulation for Problem 8 which is better suited for large-scale problems. Instead of fixing the number of nodes N_{DAG} in the DAG $\mathcal{D}$ and searching over grammar element types $\mathbf{X}_{uv}$ for which to populate the nodes, we can fix $\mathbf{X}$ to contain a number of instances of each grammar element, and relax the constraint that there is only one root node, and enforce the constraints of Problem 8 on the LTL formula defined by the subgraph of a particular root node; that is, we enforce the constraints on one tree in the forest of an expanded DAG where AP nodes with common labels are not merged. Additionally, instead of incrementing the *total* DAG size N_{DAG} in the outer loop of Algorithm 1, we should increment the number of instances of *each* grammar element by one. As a concrete example, instead of searching for a DAG with 5 nodes, where each node can be labeled with any element in the grammar $\{p_1, p_2, \wedge, \Diamond, \Box\}$, one possibility under this reformulation would be to fix $\mathbf{X}$ to contain 11 nodes, with one instance each of p_1 and p_2 and three instances each of $\wedge$, $\Diamond$, and $\Box$. The optimizer would then choose a subset of these nodes to include in the candidate LTL formula by choosing a root node and $(\mathbf{L}, \mathbf{R})$ accordingly.

More concretely, this reformulated problem can be written as a modification of Problem 8, where $\mathbf{X}$ is dropped as a decision variable and an additional binary vector $\mathbf{r} \in \{0, 1\}^{N_{\text{DAG}}}$ is added. The purpose of $\mathbf{r}$ is to encode that at least one node in the DAG is a root node, and that conditions (14), (15), and (16) hold for each root node; concretely, if $r_i = 1$, node i is a root node, and if $r_i = 0$, then node i is not a root node. This adjustment can be performed by taking constraints (14), (15), and (16) and relaxing them depending on the value of $\mathbf{r}$, using a big-M formulation. As a concrete example, (14) would be modified to

$$1 - \mathbf{S}^{\text{dem}}_{j,(r_i,1)} \leq M(1 - r_i),\ i = 1, \ldots, N_{\text{DAG}},\ j = 1, \ldots, N_s. \tag{17}$$

By holding $\mathbf{X}$ constant instead of considering it as a decision variable, we dramatically reduce the computational cost of solving Problem 8, by combinatorially reducing the search space size compared to searching over the entirety of $(\mathbf{X}, \mathbf{L}, \mathbf{R})$. Furthermore, this does not overly restrict the LTL formula search, since we can still represent different formulas by searching over $\mathbf{L}$ and $\mathbf{R}$, and by allowing for multiple root nodes, we can still find different formulas involving a different number of nodes (i.e. the method can return formula defined by a subtree containing only a subset of the nodes in $\mathbf{X}$). For instance, consider representing the formula $\varphi = \Diamond p_1 \wedge \Diamond p_2$ using either formulation. Using the original formulation, one can represent φ by searching for a DAG with 5 nodes, resulting in the structure in Fig. 2. Using the reformulation, we can represent φ even when selecting one instance each of p_1 and p_2 and three instances each of $\wedge$, $\Diamond$, and $\Box$, as long as some subgraph in the resulting DAG replicates the structure in Fig. 2, and the root of that subgraph (say node i) is marked as a root node ($r_i = 1$). However, these computational gains can come at the cost of easily finding the shortest LTL formula consistent with the demonstrations, as we discuss in Corollary 1 (see Remark 5 for more discussion). Thus, this formulation should be used for large-scale learning problems with many APs and LTL grammar elements, while it should be avoided when the primary priority is to return the simplest possible LTL formula.

7.3 Prioritized variants on the falsification loop

Depending on the desired application, it may be useful to impose an ordering in which candidate structures θ^s are returned in line 4 of Algorithm 1. For example, the user may want to return the most restrictive formulas first (i.e. formulas with the smallest language), since more restrictive formulas are less likely to admit counterexamples (and hence the falsification should terminate in fewer iterations). On the other hand, the user may want to return the least restrictive formulas first, generating many invalid formulas in order to explicitly know what formulas do not satisfy the demonstrator's wishes.

However, imposing an entailment-based ordering on the returned formulas is computationally challenging, as in general this will involve pairwise LTL entailment checks over a large set of possible LTL formulas, and each check is in

PSPACE (Demri and Schnoebelen 2002). Despite this, we can heuristically approximate this by assigning weights to each node type in the DAG based on their logical "strength", such that each DAG with the same set of nodes has an overall weight $w = \sum_{u=1}^{N_{\text{DAG}}} \sum_{v=1}^{N_{\text{g}}} w_{u,v} \mathbf{X}_{u,v}$. For example, $\vee$ should be assigned a lower weight than $\wedge$, since $\vee$s can never restrict language size, while $\wedge$ can never grow it. Then, stronger/weaker formulas can be returned first by adding constraint $w \geq w_{\text{thresh}}/w \leq w_{\text{thresh}}$, where w_{thresh} is reduced/increased until a consistent formula is found.

Note that multiple consistent formula structures can be also generated by adding a constraint for Problem 8 to not return the same formula structure and continuing the falsification loop after the first consistent formula is found.

7.4 Demonstration suboptimality

We conclude this section by describing a method for estimating the suboptimality slack parameter δ, which is crucial for maintaining the correctness of Algorithm 1, and by discussing how demonstrator suboptimality can affect the performance of our algorithm.

We first describe how δ can be estimated. Assume that the cost function parameters θ^c are fixed. Suppose that the demonstrator repeats task j R times, generating suboptimal demonstrations $\{\xi_{j,r}^{\text{dem}}\}_{r=1}^{R}$ with corresponding costs $\{c(\xi_{j,r}^{\text{dem}})\}_{r=1}^{R}$, where $c(\xi_{j,r})^{\text{dem}} \geq c(\xi_j^*)$, for all r, where $c(\xi_j^*)$ is the cost of a globally-optimal solution for task j, which we assume is finite. Using these repeated demonstrations, we would like to estimate the suboptimality bound δ. Assuming the demonstration costs are independent and identically distributed realizations of a random variable, we can estimate $c(\xi_j^*)$ using the location parameter of a Weibull distribution that is fit to the observed costs (Weng et al. 2018; De Haan and Ferreira 2007; Knuth et al. 2021). This follows from the Fisher–Tippett–Gnedenko Theorem from extreme value theory (De Haan and Ferreira 2007), which states that if the limiting distribution of the minimum of a set of realizations of a random variable converges to a finite value, the limit distribution is Weibull. Then, the location parameter of the Weibull distribution can be used to estimate the minimum $c(\xi_j^*)$; let this estimate be denoted $\hat{c}_j^*$. One can also compute a confidence interval around $\hat{c}_j^*$ (Knuth et al. 2021), which can be used to determine if the demonstration needs to be further repeated (i.e. if the confidence interval is large). Finally, we can recover an estimate of δ by taking the lowest-cost trajectory (which will be selected as the demonstration used in learning[2]) with cost $c(\xi_j^{\text{dem}}) \doteq \min_{1 \leq r \leq R} c(\xi_{j,r}^{\text{dem}})$ and setting

$$\delta = \frac{c(\xi_j^{\text{dem}}) - \hat{c}_j^*}{\hat{c}_j^*}. \tag{18}$$

We demonstrate this procedure on a simulated example in Sect. 9.2.

This δ estimation procedure can also be altered to work for the case of unknown θ^c. Since Algorithm 2 fixes a single consistent θ^c in an outer loop and then runs the falsification loop of Algorithm 1 for that fixed θ^c, we can estimate δ for each θ^c which comes up in the outer loop directly using the procedure described previously for a fixed θ^c. Thus, δ changes based on the current candidate θ^c.

We now discuss the overall effect of suboptimality on our method. Recall that our approach relies on a continuous notion of optimality to learn θ^p and θ^c (the KKT conditions) and discrete notions of optimality in a falsification loop to learn the LTL structure θ^s. We first discuss the effect of suboptimality on learning θ^p and θ^c; in these cases, any demonstrator suboptimality is reflected by the KKT conditions failing to hold exactly on the demonstrations (i.e. with an error in the stationarity or complementary slackness terms). This can be dealt with by solving Problem 3, which relaxes the KKT conditions to a penalty, so the optimization problem remains feasible despite the suboptimality. In essence, Problem 3 finds the cost function/AP parameters which make the demonstrations as close to satisfying the KKT conditions as possible. Unfortunately, these parameters may not reflect the true parameters if the demonstrations are extremely suboptimal; as a result, the accuracy of the recovered parameters can be sensitive to suboptimality. Quantifying uncertainty in the learned parameters as a function of the demonstrator's suboptimality may help mitigate any performance degradation, and is an interesting direction for future work.

Learning the LTL structure θ^s is in general less sensitive to suboptimality. To understand this, let us return to the two-AP setting of Fig. 3. In this setting, we first sort the possible LTL structures on a number line by the optimal trajectory cost that they admit (see Fig. 4 for a depiction of this idea). There are finitely many possible LTL structures θ^s, and many different θ^s may be semantically identical (for example, many θ^s have corresponding formulas which are just permutations of each other), thus admitting optimal trajectories of the same cost. Thus, while there may be exponentially many possible θ^s, there tends to only be a small number of groups of cost-distinguishable formulas (i.e. each such group contains formulas with equal optimal cost). Recall that in running Algorithm 1 using the given demonstrations to learn θ^s, the falsification loop terminates when the optimal cost of a trajectory satisfying the current candidate LTL formula and the known constraints exceeds the δ-adjusted demonstration cost $c(\xi_j^{\text{dem}})/(1+\delta)$. As an example, consider a suboptimal

[2] Provided that the remaining higher-cost demonstrations are feasible, they can still be used in the learning process; we can enforce that these demonstrations should still be feasible for any candidate LTL formula.

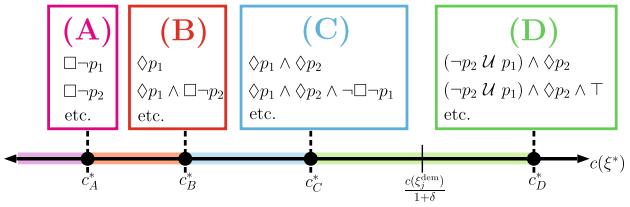

Fig. 4 Consider the two-AP setting first shown in Fig. 3. We visualize here sets of LTL formulas which can be distinguished based on cost. Formulas within group $(\cdot)$ have an optimal cost $c^*_\cdot$. The formulas listed in each group (A), (B), (C), and (D) are just a small subset of a much larger set of cost-indistinguishable formulas. For instance, if a demonstration has a δ-adjusted cost $c(\xi_j^{\text{dem}})/(1+\delta)$ falling in the green range, Algorithm 1 will return some LTL formula structure in group (D), each of which would have an optimal cost of c^*_D (Color figure online)

demonstration of $\varphi = (\neg p_2 \,\mathcal{U}\, p_1) \wedge \Diamond p_2$ which belongs to group (D) in Fig. 4 and has a δ-adjusted cost $c(\xi_j^{\text{dem}})/(1+\delta)$. As long as this adjusted cost lies anywhere within the green interval in Fig. 4, some formula from group (D) is returned, which will be a formula consistent with the bounded suboptimality of the demonstration. Note that the estimate of δ must be an overestimate of the true δ in order for the adjusted cost to lie in the green region in Fig. 4; this can be encouraged by setting the confidence interval described earlier in this section to be large, and selecting δ as the fit Weibull location parameter padded by the confidence interval. In Sect. 9.2, we show that we can obtain an overestimate of the true δ using this approach.

8 Theoretical analysis

In this section, we prove some theoretical guarantees of our method: that it is complete under some assumptions, without (Theorem 2) or with (Corollary 2) spec-optimality, that it returns the shortest LTL formula consistent with the demonstrations (Corollary 1), and that we can compute guaranteed conservative estimates of $\mathcal{S}_i/\mathcal{A}_i$ (Theorem 3).

Assumption 1 Problem 7 is solved with a complete planner.

Assumption 2 Each demonstration is locally-optimal (i.e. satisfies the KKT conditions) for fixed boolean variables.

Assumption 3 The true parameters θ^p, θ^s, and θ^c are in the hypothesis space of Problem 8: $\theta^p \in \Theta^p, \theta^s \in \Theta^s, \theta^c \in \Theta^c$.

We will use these assumptions to show that when the cost function parameters θ^p are known, our falsification loop in Algorithm 1 is guaranteed to return a consistent formula; that is, it makes the demonstrations globally-optimal.

Theorem 2 (Completeness and consistency, unknown θ^s, θ^p) *Under Assumptions 1–3, Algorithm 1 is guaranteed to return a formula $\varphi(\theta^s, \theta^p)$ such that (1) $\xi_j^{\text{dem}} \models \varphi(\theta^s, \theta^p)$ and (2) ξ_j^{dem} is globally-optimal under $\varphi(\theta^s, \theta^p)$, for all j, (3) if such a formula exists and is representable by the provided grammar.*

Proof To see the first point—that Algorithm 1 returns $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ such that $\xi_j^{\text{dem}} \models \varphi(\hat{\theta}^s, \hat{\theta}^p)$ for all j, note that in Problem 8, the constraints (13)–(15) on the satisfaction matrices $\mathbf{S}_j^{\text{dem}}$ encode that each demonstration is feasible for the choice of θ^p and θ^s; hence, the output of Problem 8 will return a feasible $\varphi(\hat{\theta}^s, \hat{\theta}^p)$. As Algorithm 1 will eventually return some $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which is an output of Problem 8, the $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ that is ultimately returned is feasible.

Next, to see the second point - that the ultimately returned $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ makes each ξ_j^{dem} globally-optimal. Note that at some iteration of the inner loop, if Problem 7 is feasible and its solution algorithm is complete (Assumption 1), it will return a trajectory which is lower-cost than the demonstration and satisfies $\varphi(\hat{\theta}^s, \hat{\theta}^p)$. Note that disregarding the lower-cost constraint, Problem 7 will always be feasible, since Problem 8 returns θ^p, θ^s for which the demonstration is feasible, and the feasible set of Problem 7 contains the demonstration. The falsification loop will continue until Problem 7 cannot produce a trajectory of strictly lower cost for each demonstration; this is equivalent to ensuring that each demonstration is globally optimal for the $\varphi(\hat{\theta}^s, \hat{\theta}^p)$.

To see the last point, we note that if there exists a formula $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which satisfies the demonstrations, it is among the feasible set of possible outputs of Algorithm 1; that is, the representation of LTL formulas, $\mathcal{D}$, is complete (cf. Lemma 1 in Neider and Gavran (2018)). □

We will further show that the formula returned by Algorithm 1 is the shortest formula which is consistent with the demonstrations; this is due to N_{DAG} only being incremented upon infeasibility of a smaller N_{DAG} to explain the demonstrations.

Corollary 1 (Shortest formula) *Let N^* be the size of a minimal DAG for which there exists (θ^p, θ^s) such that $\xi_j^{\text{dem}} \models \varphi(\theta^s, \theta^p)$ for all j. Under Assumptions 1–3, Algorithm 1 is guaranteed to return a DAG of size N^*.*

Proof The result follows since Algorithm 1 increases N_{DAG} incrementally (in the outer loop) until some $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ is returned which makes all of the demonstrations feasible and globally-optimal, and each inner iteration of Algorithm 1 is guaranteed to find a consistent $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ if one exists (cf. Theorem 2). □

Remark 5 A similar shortest formula guarantee can be obtained for the reformulation of Algorithm 1 described in Sect. 7.2 only if it is tractable to perform an exhaustive search over the number of nodes allocated to each grammar element, in order to find the shortest-length combination. This can be computationally intensive, and is in contrast to the simple

"line-search" over a single complexity variable, N_{DAG}, that the original Algorithm 1 enjoys.

Using Lemma 1, we can show that modifying Algorithm 1 to additionally impose the spec-optimality conditions in Problem 8 still enjoys the completeness properties discussed in Theorem 2, while also in general reducing the number of falsification iterations needed as a result of the reduced search space.

Corollary 2 (Algorithm 1 with spec-optimality) *By modifying Algorithm 1 so that Problem 8 uses constraints (16), Algorithm 1 still returns a consistent solution $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ if one exists, i.e. each ξ_j^{dem} is feasible and globally optimal for each $\varphi(\hat{\theta}^s, \hat{\theta}^p)$.*

Proof The result follows from completeness of Algorithm 1 (cf. Theorem 2) and Lemma 1: adding (16a)–(16c) enforces that ξ_j^{dem} are spec-optimal, and via Lemma 1, ξ_j^{dem}, which is a globally-optimal demonstration, must also be spec-optimal. Hence, imposing constraints (16a)–(16c) is consistent with the demonstration. □

Next, we show how the consistency properties extend to the case of unknown cost function, if Algorithm 2 returns a solution, which it is not guaranteed to do in finite time.

Corollary 3 (Consistency, unknown θ^c) *Under Assumptions 1–3, if Algorithm 2 terminates in finite time, it returns a formula $\varphi(\theta^s, \theta^p)$ such that (1) $\xi_j^{\text{dem}} \models \varphi(\theta^s, \theta^p)$ and (2) ξ_j^{dem} is globally-optimal with respect to θ^c under the constraints of $\varphi(\theta^s, \theta^p)$, for all j, (3) if such a formula exists and is representable by the provided grammar.*

Proof Note that Algorithm 2 is simply Algorithm 1 with an outer loop where potential cost parameters θ^c are chosen. From Theorem 2, we know that under Assumptions 1–2, for the true cost parameter θ^c, Algorithm 1 is guaranteed to return θ^p and θ^s which make the demonstrations globally-optimal under θ^c. From Assumption 3 and the fact that the true parameters θ^p, θ^s, and θ^c will make the demonstrations globally-optimal, we know there exists at least one consistent set of parameters (the true parameters). Then, Algorithm 2 will eventually find a consistent solution (possibly the true parameters), as it iteratively runs Algorithm 1 for all consistent θ^c. □

Finally, we show that for fixed LTL structure and cost function, querying and volume extraction (Problems 5 and 6) are guaranteed to return conservative estimates of the true $\mathcal{S}_i$ or $\mathcal{A}_i$.

Theorem 3 (Conservativeness for unknown θ^p) *Suppose that θ^s and θ^c are known, and θ^p is unknown. Then, extracting $\mathcal{G}_s^i$ and $\mathcal{G}_{\neg s}^i$, as defined in (11)–(12), from the feasible set of Problem 4 projected onto Θ_i^p (denoted $\mathcal{F}_i$), returns $\mathcal{G}_s^i \subseteq \mathcal{S}_i$ and $\mathcal{G}_{\neg s}^i \subseteq \mathcal{A}_i$, for all $i \in \{1, \ldots, N_{\text{AP}}\}$.*

Proof We first prove that $\mathcal{G}_{\neg s}^i \subseteq \mathcal{A}_i$. Suppose that there exists $\kappa \in \mathcal{G}_{\neg s}^i$ such that $\kappa \notin \mathcal{A}_i$. Then by definition, for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa, \theta_i^p) \geq 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter $\theta_i^{p,*}$; hence, $\theta_i^{p,*} \in \mathcal{F}$. Then, $x \in \mathcal{A}(\theta_i^{p,*})$. Contradiction. Similar logic holds for proving that $\mathcal{G}_s^i \subseteq \mathcal{S}_i$. Suppose that there exists $x \in \mathcal{G}_s^i$ such that $x \notin \mathcal{S}_i$. Then by definition, for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa, \theta_i^p) \leq 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter $\theta_i^{p,*}$; hence, $\theta_i^{p,*} \in \mathcal{F}_i$. Then, $\kappa \in \mathcal{S}_i(\theta_i^{p,*})$. Contradiction. □

9 Simulation experiments

We show that our algorithm outperforms a competing method (Sect. 9.1), can be robust to suboptimality in the demonstrations (Sect. 9.2), can learn shared task structure from demonstrations across environments (Sect. 9.3), and can learn LTL formulas θ^p, θ^s and uncertain cost functions θ^c on high-dimensional problems. Specifically, we demonstrate Algorithm 1 on a simulated manipulation example (Sect. 9.4) and the one-shot learning described in Remark 4 on a quadrotor surveillance task (Sect. 9.5). Please refer to the supplementary video for visualizations of the results.

9.1 Baseline comparison

Likely the closest method to ours is Jha et al. (2019), which learns a pSTL formula that is tightly satisfied by the demonstrations via solving a nonconvex problem to local optimality: $\arg\max_{\theta^p} \min_j \tau(\theta^p, \xi_j^{\text{dem}})$, where $\tau(\theta^p, \xi_j^{\text{dem}})$ measures how tightly ξ_j^{dem} fits the learned formula. We run the authors' code [26] on a toy problem (see Fig. 5), where the demonstrator has kinematic constraints, minimizes path length, and satisfies start/goal constraints and $\varphi = \Diamond_{[0,8]} p_1$, where $x \models p_1 \Leftrightarrow [I_{2\times 2}, -I_{2\times 2}]^\top x \leq [3, 2, -1, 2]^\top = [3, \theta_1^p]^\top$. We assume the structure θ^s is known, and we aim to learn θ^p to explain why the demonstrator deviated from an optimal straight-line path to the goal. Solving Problem 6 returns $\mathcal{G}_s^1 = \mathcal{S}_1$ (Fig. 5, right). On the other hand, we run TeLEx multiple times, converging to different local optima, each corresponding to a "tight" θ^p (Fig. 5, center): TeLEx cannot distinguish between multiple different "tight" θ^p, which makes sense, as the method tries to find *any* "tight" solution. This example suggests that if the demonstrations are goal-directed, a method that leverages their optimality is likely to better explain them.

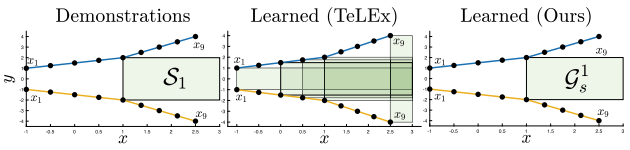

Fig. 5 Toy example for baseline comparison (Jha et al. 2019). The baseline is unable to disambiguate between possible APs as it does not consider the demonstrator's objective

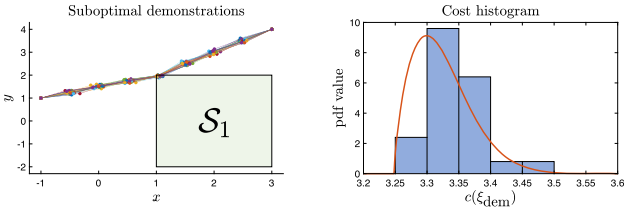

Fig. 6 **Left**: We are given 25 suboptimal demonstrations of the same task, with each demonstration starting at $[-1, 1]$, ending at $[3, 4]$, and satisfying $\Diamond_{[0,8]} p_1$. The globally-optimal cost is 3.25, while the best cost observed within the 25 demonstrations is 3.274. **Right**: We fit a Weibull distribution (orange) to the demonstration costs (right). The fitted location parameter, adjusted by its 95% confidence interval, is $3.248 < 3.25$, which leads to a valid overestimate of δ

9.2 δ-estimation for suboptimal demonstrations

In this example, we demonstrate the suboptimality estimation method described in Sect. 7.4. In this example, we consider the same problem setting as in Sect. 9.1, but instead use suboptimal versions of the blue demonstration in Fig. 5. We are given 25 such demonstrations (Fig. 6, left), and we are interested in estimating the suboptimality slack parameter δ. To do so, we follow the method in Sect. 7.4, fitting a Weibull distribution (Fig. 6, right, orange) to the demonstration costs (Fig. 6, right, blue histogram). The fitted Weibull distribution has a location parameter of 3.248 after being adjusted by its 95% confidence interval, which is smaller than the optimal cost of 3.25. Using the suboptimal demonstration with the lowest cost (in this case, 3.274), we can estimate $\delta = 0.008$ using (18), which overestimates the true $\delta = 0.007$. Per the discussion in Sect. 7.4, it is important to be able to obtain an estimate of δ which is a tight overestimate of the true δ, which this example achieves. Overall, this example suggests that our δ-estimation technique can effectively estimate the suboptimality bound, which is important for learning consistent LTL formulas in spite of suboptimality in the demonstrations.

9.3 Learning shared task structure

In this example, we show that our method can extract logical structure shared between demonstrations that complete the same high-level task, but in different environments (Fig. 7). A point robot must first go to the mug (p_1), then go to the coffee machine (p_2), and then go to goal (p_3) while avoiding obstacles (p_4, p_5). As the floor maps differ, θ^p also differ, and are assumed known. We add two relevant primitives to the grammar, sequence:

$$\varphi_1 \mathcal{Q} \varphi_2 \doteq \neg\varphi_2 \mathcal{U}_{[0,T_j-1]} \varphi_1,$$

enforcing that φ_2 cannot occur until after φ_1 has occurred for the first time, and avoid: $\mathcal{V}\varphi \doteq \Box_{[0,T_j-1]}\neg\varphi$, enforcing φ never holds over $[1, T_j]$. Then, the true formula is:

$$\varphi^* = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge (p_1 \mathcal{Q} p_2) \wedge (p_2 \mathcal{Q} p_3) \wedge \Diamond_{[0,T_j-1]} p_3.$$

Suppose first that we are given the blue demonstration in Environment 2. Running Algorithm 1 with 1-SO constraints (16) terminates in one iteration at $N_{\text{DAG}} = 14$ with

$$\varphi_0 = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3 \wedge (p_1 \mathcal{Q} p_2).$$

That is, always avoid obstacles 1 and 2, eventually reach coffee and goal, and visit mug before coffee. This formula is insufficient to complete the true task (the ordering constraint between coffee and goal is not learned). This is because the optimal trajectories satisfying φ_0 and φ^* are the same cost, i.e. both φ_0 and φ^* are consistent with the demonstration and could have been returned, and $\varphi_0, \varphi^* \in \varphi_g$ (cf. Sect. 8). Now, we also use the blue demonstration from Environment 1 (two examples total). Running Algorithm 1 terminates in two iterations at $N_{\text{DAG}} = 14$ with the formulas

$$\varphi_1 = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge \Diamond_{[0,T_j-1]} p_1 \wedge \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3$$

(which enforces that the mug, coffee, and goal must be eventually visited, but in any order, while avoiding obstacles) and $\varphi_2 = \varphi^*$. Since the demonstration in Environment 1 doubles back to the coffee before going to goal, increasing its cost over first going to goal and then to coffee, the ordering constraint between the two is learnable. We also plot the generated counterexample (Fig. 7, yellow), which achieves a lower cost, as φ_1 involves no ordering constraints. We can use the learned formula to plan a path completing the task in a new environment (with different AP parameters θ^p) in Fig. 8.

Overall, this example suggests we can use demonstrations from different environments to learn common task structure and disambiguate between potential explanations.

9.4 Multi-stage manipulation task

We consider the setup in Figs. 9 and 10 of teaching a 7-DOF Kuka iiwa robot arm to prepare a drink: first move the end effector to the button on the faucet (p_1), then grasp the cup (p_2), then move the cup to the customer (p_3), all while avoiding obstacles. After grasping the cup, an end-effector

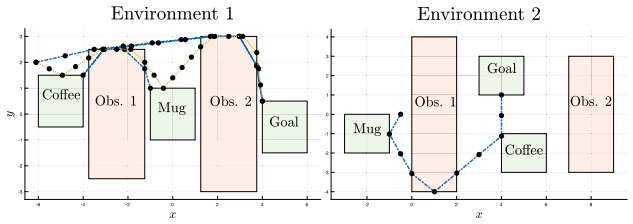

Fig. 7 We learn a common LTL formula from demonstrations in different environments (different θ^p) with shared task (same θ^s)

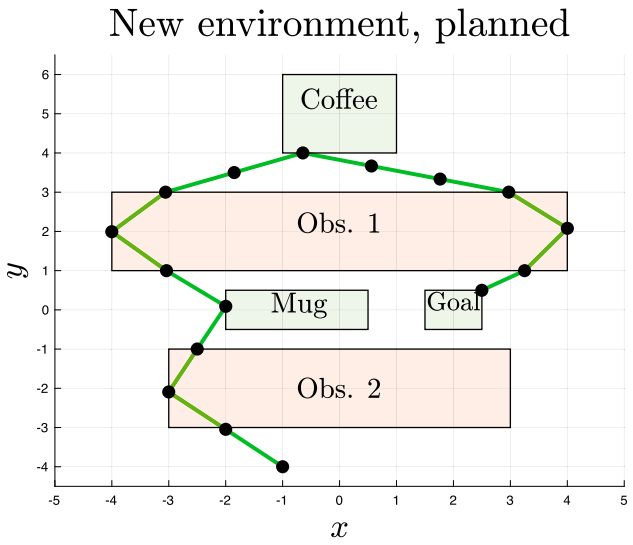

Fig. 8 Trajectory planned with the learned LTL formula on the environment-transfer example

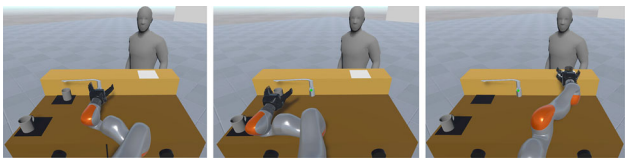

Fig. 9 Multi-stage simulated manipulation task: first fill the cup, then grasp it, and then deliver it. To avoid spills, a pose constraint is enforced after the cup is grasped

pose constraint $(\alpha, \beta, \gamma) \in \mathcal{S}_4(\theta_4^p)$ (p_4) must be obeyed. We add two "distractor" APs: a different cup (p_5) and a region (p_6) where the robot can hand off the cup. We also modify the grammar to include the sequence operator $\mathcal{Q}$, (defined as before), and add an "after" operator

$$\varphi_1 \,\mathcal{T}\, \varphi_2 \doteq \Box_{[0,T_j-1]}(\varphi_2 \rightarrow \Box_{[0,T_j-1]}\varphi_1),$$

that is, φ_1 must hold after and including the first timestep where φ_2 holds. The true formula is:

$$\varphi^* = (p_1 \,\mathcal{Q}\, p_2) \wedge (p_2 \,\mathcal{Q}\, p_3) \wedge \Diamond_{[0,T_j-1]} p_3 \wedge (p_4 \,\mathcal{T}\, p_2).$$

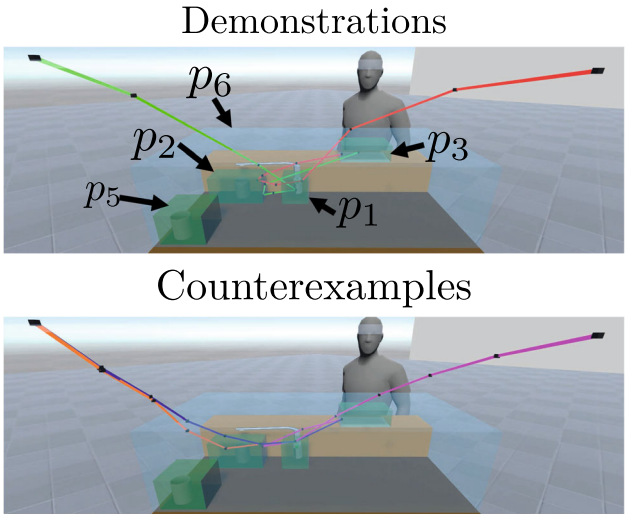

Fig. 10 Demonstrations and counterexamples for the simulated manipulation task

We use a kinematic arm model: $j_{t+1}^i = j_t^i + u_t^i, i = 1, \ldots, 7$, where $\|u_t\|_2^2 \leq 1$ for all t. Two suboptimal human demonstrations ($\delta = 0.7$) optimizing $c(\xi_{xu}) = \sum_{t=1}^{T-1} \|j_{t+1} - j_t\|_2^2$ are recorded in a Unity virtual reality (VR) environment. We assume we have nominal estimates of the AP regions $\mathcal{S}_i(\theta_{i,\text{nom}}^p)$ (e.g. from a vision system), and we want to learn the θ^s and θ^p of φ^*. We use IPOPT (Wächter and Biegler 2006) to solve the nonlinear optimization problems needed to compute counterexamples.

We run Algorithm 1 with the 1-SO constraints (16), and encode the nominal θ_i^p by enforcing that $\Theta_i^p = \{\theta_i^p \mid \|\theta_i^p - \theta_{i,\text{nom}}^p\|_1 \leq 0.05\}$. At $N_{\text{DAG}} = 11$, the inner loop runs for 3 iterations (each taking 30 minutes on an i7-7700K processor), returning candidates

$$\varphi_1 = (p_1 \mathcal{Q} p_3) \wedge (p_2 \mathcal{Q} p_3) \wedge (\Diamond_{[0,T_j-1]} p_3) \wedge (p_4 \mathcal{T} p_3),$$
$$\varphi_2 = (p_1 \mathcal{Q} p_3) \wedge (p_2 \mathcal{Q} p_3) \wedge (\Diamond_{[0,T_j-1]} p_3) \wedge (p_4 \mathcal{T} p_2),$$

and $\varphi_3 = \varphi^*$. φ_1 says that before going to the customer, the robot has to visit the button and cup in any order, and then must satisfy the pose constraint after visiting the cup. φ_2 has the meaning of φ^*, except the robot can go to the button or cup in any order. Note that φ_3 is a stronger formula than φ_2, and φ_2 than φ_1; this is a natural result of the falsification loop, which returns incomparable or stronger formulas with more iterations, as the counterexamples rule out weaker or equivalent formulas. Also note that the distractor APs don't feature in the learned formulas, even though both demonstrations pass through p_6. This happens for two reasons: we increase N_{DAG} incrementally and there was no room to include distractor objects in the formula (since spec-optimality may enforce that p_1-p_3 appear in the formula), and even if N_{DAG} were not minimal, p_6 would not be guar-

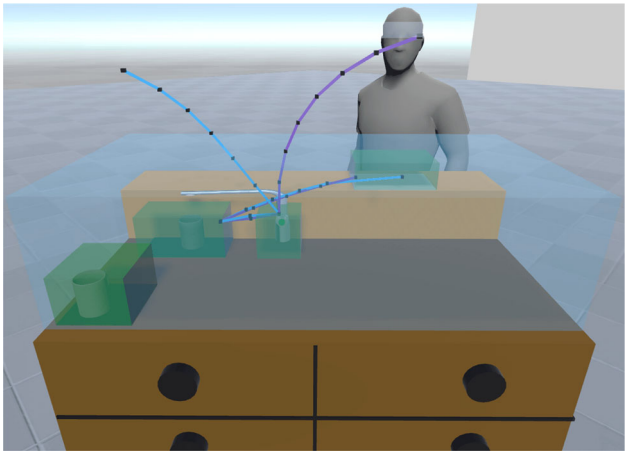

Fig. 11 Trajectories planned using the learned LTL formula, for the simulated 7-DOF arm

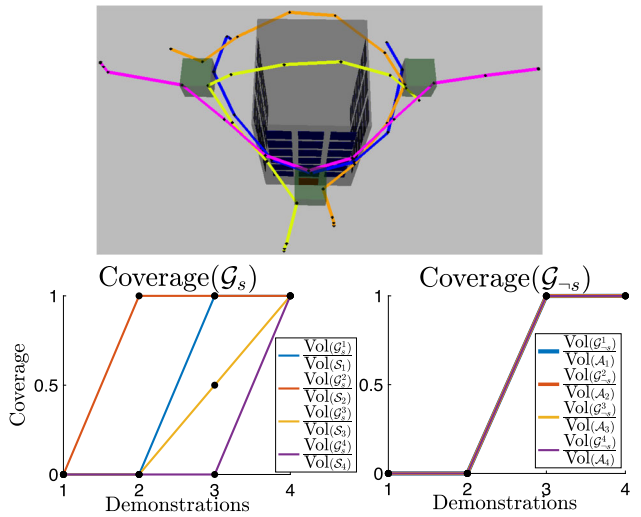

Fig. 12 Quadrotor surveillance demonstrations (top) and learning curves (bottom) (Color figure online)

anteed to show up, since visiting p_6 does not increase the trajectory cost.

We plot the counterexamples in Fig. 10: blue/purple are from iteration 1; orange is from iteration 2. They save cost by violating the ordering and pose constraints: from the left start state, the robot can save cost if it visits the cup before the button (blue, orange trajectories), and loosening the pose constraint can reduce joint space cost (orange, purple trajectories). The right demonstration produces no counterexample in iteration 2, as it is optimal for this formula (changing the order does not lower the optimal cost). For the learned θ^p, $\theta_i^p = \theta_{i,\text{nom}}^p$ except for p_2, p_3, where the box shrinks slightly from the nominal; this is because by tightening the box, a Lagrange multiplier can be increased to reduce the KKT residual. We use the learned θ^p and θ^s to plan trajectories which complete the task from new initial conditions in the environment (Fig. 11).

Overall, this example suggests that Algorithm 1 can recover θ^p and θ^s on a high-dimensional problem and ignore distractor APs, despite demonstration suboptimality.

9.5 Multi-stage quadrotor surveillance

We demonstrate that we can jointly learn θ^p, θ^s, and θ^c in one shot on a 12D nonlinear quadrotor system. The system dynamics for the quadrotor (Sabatino 2015) are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\gamma} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\beta}\frac{\sin(\gamma)}{\cos(\beta)} + \dot{\gamma}\frac{\cos(\gamma)}{\cos(\beta)} \\ \beta\cos(\gamma) - \dot{\gamma}\sin(\gamma) \\ \dot{\alpha} + \dot{\beta}\sin(\gamma)\tan(\beta) + \dot{\gamma}\cos(\gamma)\tan(\beta) \\ -\frac{1}{m}[\sin(\gamma)\sin(\alpha) + \cos(\gamma)\cos(\alpha)\sin(\beta)]u_1 \\ -\frac{1}{m}[\cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta)]u_1 \\ g - \frac{1}{m}[\cos(\gamma)\cos(\beta)]u_1 \\ \frac{I_y - I_z}{I_x}\dot{\beta}\dot{\gamma} + \frac{1}{I_x}u_2 \\ \frac{I_z - I_x}{I_y}\dot{\alpha}\dot{\gamma} + \frac{1}{I_y}u_3 \\ \frac{I_x - I_y}{I_z}\dot{\alpha}\dot{\beta} + \frac{1}{I_z}u_4 \end{bmatrix}, \tag{19}$$

with control constraints $\|u_t\|_2 \leq 10$. We time-discretize the dynamics by performing forward Euler integration with discretization time $\delta t = 1.2$ seconds. The 12D state is $x = [\chi, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}]^\top$, and the relevant constants are $g = -9.81\text{m/s}^2$, $m = 1\text{kg}$, $I_x = 0.5\text{kg}\cdot\text{m}^2$, $I_y = 0.1\text{kg}\cdot\text{m}^2$, and $I_z = 0.3\text{kg}\cdot\text{m}^2$.

We are given four demonstrations of a quadrotor surveilling a building (Fig. 12): it needs to visit three regions of interest (Fig. 12, green) while not colliding with the building. All visitation constraints can be learned directly with 1-SO (see Remark 4) and collision-avoidance can also be learned with 1-SO, with enough demonstrations. The true formula is

$$\varphi^* = \Diamond_{[0,T_j-1]}p_1 \wedge \Diamond_{[0,T_j-1]}p_2 \wedge \Diamond_{[0,T_j-1]}p_3 \wedge \Box_{[0,T_j-1]}\neg p_4,$$

where p_1-p_3 represent the regions of interest and p_4 is the building. We aim to learn θ_i^p for the parameteriza-

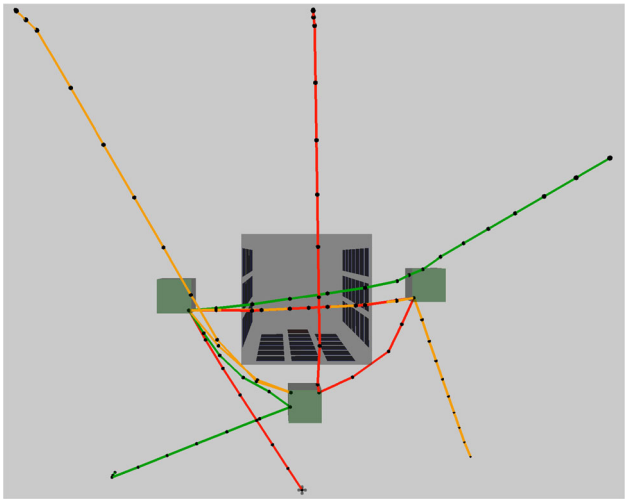

Fig. 13 Trajectories planned using the learned LTL formula, for the quadrotor system

tion $\mathcal{S}_i(\theta_i^p) = \{[I_{3\times3}, -I_{3\times3}]^\top [x, y, z]^\top \leq \theta_i^p\}$, assuming $\theta_{4,6}^p = 0$ (the building is not hovering). The demonstrations minimize $c(\xi_{xu}, \theta^c) = \sum_{r\in R} \sum_{t=1}^{T-1} \gamma_r (r_{t+1} - r_t)^2$, where $R = \{x, y, z, \dot{\alpha}, \dot{\beta}, \dot{\gamma}\}$ and $\gamma_r = 1$, i.e. equal penalties to path length and angular acceleration. We assume $\gamma_r \in [0.1, 1]$ and is unknown: we want to learn the cost weights for each state.

Solving Problem 8 with 1-SO conditions (at $N_{\text{DAG}} = 12$) takes 44 minutes and recovers θ^p, θ^s, and θ^c in one shot. To evaluate the learned θ^p, we show in Fig. 12 that the coverage of the $\mathcal{G}_s^i$ and $\mathcal{G}_{\neg s}^i$ for each p_i (computed by fixing the learned θ^s and running Problem 6) monotonically increases with more data. In terms of recovered θ^s, with one demonstration, we return

$$\varphi_1 = \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3 \wedge \Diamond_{[0,T_j-1]} p_4 \wedge \Box_{[0,T_j-1]} \neg p_1.$$

This highlights the fact that since we are not provided labels, there is an inherent ambiguity of how to label the regions of interest (i.e. p_i, $i = 1, \ldots, 3$ can be associated with any of the green boxes in Fig. 12 and be consistent). Also, one of the regions of interest in φ gets labeled as the obstacle (i.e. p_1 and p_4 are swapped), since one demonstration is not enough to disambiguate which of the four p_i should touch the ground. Note that this ambiguity can be eliminated if labels are provided (see Sect. 7.1) or if more demonstrations are provided: for two and more demonstrations, we learn $\varphi_i = \varphi^*$, $i = 2, \ldots, 4$. When using all four demonstrations, we recover the cost parameters θ^c and structure θ^s exactly, i.e. $\varphi(\hat{\theta}^s, \hat{\theta}^p) = \varphi^*$, and fixing the learned θ^s and running Problem 6 returns $\mathcal{G}_s^i = \mathcal{S}_i$ and $\mathcal{G}_{\neg s}^i = \mathcal{A}_i$, for all i. The learned θ^c, θ^s, and θ^p are used to plan trajectories that efficiently complete the task for different initial and goal states.

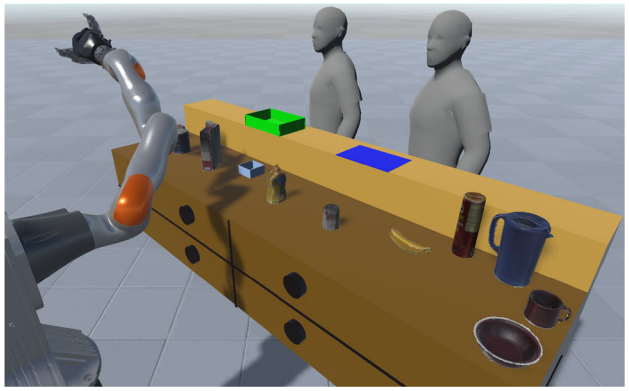

Fig. 14 We build a Unity virtual reality environment to collect demonstrations for the real-world object delivery manipulation task

Furthermore, assuming that the parameterization is correct, these plans are guaranteed to satisfy the true LTL formula; these trajectories are presented in Fig. 13.

Overall, this example suggests that our method can jointly recover a consistent set of θ^p, θ^s, and θ^c for high-dimensional systems.

10 Physical experiments

To demonstrate that our method can scale to handle the challenges of real hardware, we use our method to learn a real-world multi-stage manipulation task. A video of our physical experiment can be found in the supplementary material.

10.1 Environment and task description

Consider a tabletop manipulation task where the arm needs to retrieve several objects, put them in boxes, and deliver them in a particular order (see Fig. 14). Specifically, the task of interest is to first place a can of soup into a box (Fig. 15b, c), to then deliver that box to a blue delivery region (Fig. 15d). Next, the robot must move a Cheez-It box into a box located at a green delivery region (Fig. 15e, f). Finally, while the box containing the soup is grasped by the robot, the robot must keep its end effector upright so that the soup does not fall out of the box. The robot should also avoid colliding with the furniture as well as any other objects in the scene. There are a total of 11 objects in the scene, not including the delivery boxes or the furniture, which are taken from the YCB dataset (Çalli et al. 2017).

To describe the aforementioned task concisely in LTL, we define another new grammar element:

$$\varphi_1 \,\mathcal{M}\, \varphi_2 \doteq \Box_{[0,T_j-1]}((\varphi_2 \rightarrow \varphi_1)) \wedge \Diamond_{[0,T_j-1]} \varphi_2,$$

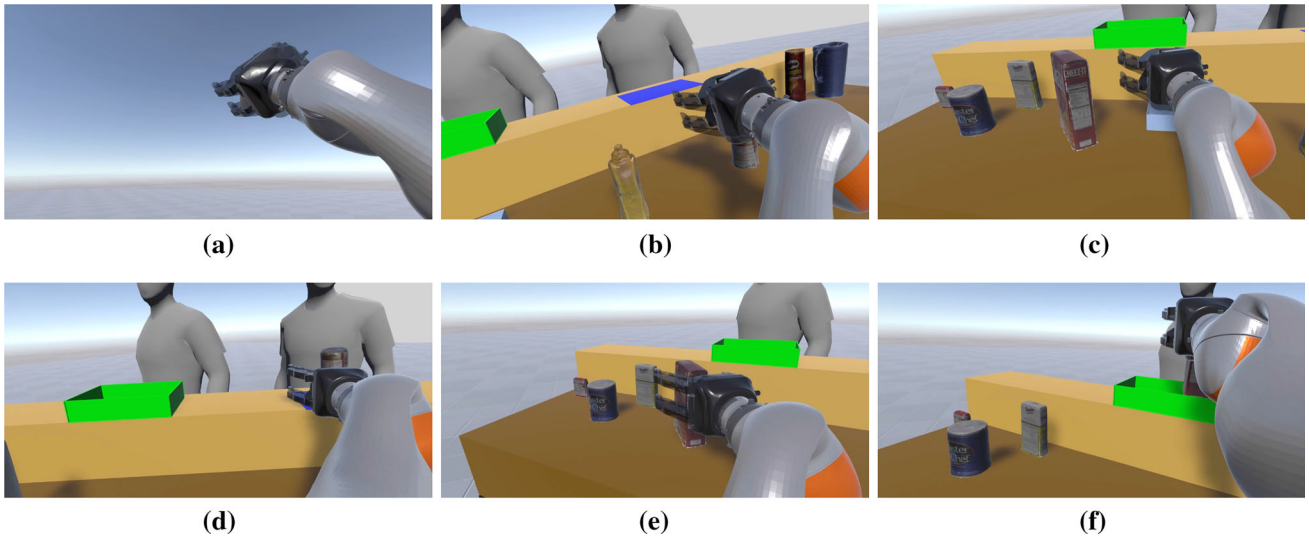

Fig. 15 One demonstration is recorded in the Unity virtual reality environment for the object delivery task, seen here from a first-person perspective. **a** Initial state. **b** First, grasp the soup. **c** Next, place the soup in the blue box, avoiding the mustard bottle which is in the way. **d** Place the box with the soup in the blue delivery region while satisfying a pose constraint. **e** Move to grasp the Cheez-It box. **f** Place the Cheez-It box in the green delivery box

i.e. if φ_2 holds, then φ_1 must also hold, and φ_2 must eventually hold. We define the following atomic propositions:

- p_S: The soup is grasped
- p_B: The movable box is grasped
- p_{G1}: The end effector is inside the blue delivery region
- p_C: The Cheez-It box is grasped
- p_{G2}: The end effector is inside the green delivery region
- p_P: The end effector is pointed upwards
- p_{D1}: End effector is within 0.05 distance of the gelatin
- p_{D2}: End effector is within 0.05 distance of the bowl
- p_{D3}: End effector is within 0.05 distance of the Master Chef coffee can
- p_{D4}: End effector is within 0.05 distance of the sugar
- p_{D5}: End effector is within 0.05 distance of the mustard bottle
- p_{D6}: End effector is within 0.05 distance of the banana
- p_{D7}: End effector is within 0.05 distance of the Pringles
- p_{D8}: End effector is within 0.05 distance of the pitcher
- p_{D9}: End effector is within 0.05 distance of the mug

We can then write an LTL formula which enforces the task as

$$\varphi^* = (p_S \,\mathcal{M}\, p_B) \wedge (p_B \,\mathcal{M}\, p_{G1}) \wedge (p_C \,\mathcal{M}\, p_{G2}) \wedge (p_{G1} \,\mathcal{Q}\, p_C) \wedge (p_P \,\mathcal{M}\, p_B).$$

The first through fourth clauses enforce that the soup, moving box, blue goal region, Cheez-It, and green delivery region are visited in the correct order, while the fifth clause enforces that the pose constraint is satisfied when the moving box is grasped. This is not overly restrictive, since per the first clause, it is not possible for the moving box to be grasped without the soup also being grasped. Note that we assume the demonstrator performs collision avoidance by avoiding contact with any object which is not the current grasp target.

10.2 LTL formula learning

For this experiment, we seek to learn the LTL formula structure θ^s while the AP parameters θ^p and cost function parameters γ are assumed known. This is reasonable for this example, since the APs detailed in Sect. 10.1 can be readily measured and the suboptimality parameter δ can be used to handle an imprecisely-known cost function. Specifically, we assume the cost function is

$$c(\xi, \gamma) = \sum_{t=1}^{T-1} \|j_{t+1} - j_t\|_2^2 + c_{\text{grasp}} \sum_{o \in \mathcal{O}} \sum_{t=1}^{T} z^o_{\text{grasp,t}},$$

where j_t denotes the arm joint values at time t, $z^o_{\text{grasp,t}} \in \{0, 1\}$ evaluates to 1 if object o is grasped at time t and 0 otherwise, $\mathcal{O}$ is the set of all manipulable objects, and $c_{\text{grasp}} = 0.01$ is a small penalty which discourages the unnecessary grasping of objects. Note that the learning is relatively robust to the specific value of c_{grasp}, as long as c_{grasp} is kept small enough such that the grasp cost term does not outweigh the path length term (in our experiments, this holds if $c_{\text{grasp}} \leq 0.115$). Mapping back to the notation of Problem 1, the state x_t contains the joint values j_t and the grasp status of each

object z_t^o, while the control input contains the joint velocities and a binary variable for each object to model grasping and releasing. The dynamics are constructed such that the grasp input for a given object is nullified if the end effector is far from that object.

We obtain one demonstration of this task which is recorded in a Unity VR environment (see Figs. 14 and 15). The demonstration consists of the state-control trajectory of the arm, as well as a binary trajectory for each object, evaluating to 0 or 1 at a given timestep depending on if that object is currently grasped. Furthermore, the initial configurations of all of the objects are given. Note that this information is sufficient to reconstruct the value of every atomic propositions. We also note that the VR environment does not simulate the grasp physics, and simply allows the demonstrator to attach an object to the grippers when it is close by. To learn θ^s, we run Algorithm 1, where Problem 8 uses the variant described in Sect. 7.2. We elect to use this variant instead of the original Problem 8 as in the simulated manipulation example (Sect. 9.4) since there are many more APs in this example (15 compared to 6 in Sect. 9.4), causing the original Problem 8 to be slow. We allocate one node for each AP, four "$\wedge$" nodes, four "$\mathcal{M}$" nodes, one "$\mathcal{Q}$" node, and one "$\Diamond$" node. We use a suboptimality parameter $\delta = 0.1$. Running Algorithm 1 generates 13 falsified candidate LTL formulas, including the following:

- $\varphi_2 = (p_C \ \mathcal{M} \ p_{G2}) \wedge (p_S \ \mathcal{M} \ p_P) \wedge (p_B \ \mathcal{Q} \ p_{G1}) \wedge (p_P \ \mathcal{M} \ p_B) \wedge (\Diamond p_{D5})$. This formula does not capture that the Cheez-Its should only be grasped after the soup has been grasped.
- $\varphi_3 = (p_B \ \mathcal{M} \ p_P) \wedge (p_P \ \mathcal{M} \ p_B) \wedge (p_{G1} \ \mathcal{Q} \ p_C) \wedge (p_C \ \mathcal{M} \ p_{G2}) \wedge (p_S \ \mathcal{M} \ p_{G1})$. This formula does not capture that the soup should be contained in the box upon delivery.
- $\varphi_8 = (p_C \ \mathcal{M} \ p_{G2}) \wedge (p_S \ \mathcal{M} \ p_P) \wedge (p_B \ \mathcal{M} \ p_{G1}) \wedge (p_P \ \mathcal{Q} \ p_C) \wedge (\Diamond \ p_{D5})$. This formula does not capture that the Cheez-Its should only be grasped after the movable box has been grasped.
- $\varphi_{13} = (p_P \ \mathcal{M} \ p_{G1}) \wedge (p_C \ \mathcal{M} \ p_{G2}) \wedge (p_B \ \mathcal{M} \ p_P) \wedge (p_S \ \mathcal{M} \ p_P) \wedge (p_{G1} \ \mathcal{Q} \ p_C)$. This formula does not enforce the pose constraint at the correct timesteps.

The candidate LTL formulas are falsified by the counterexample generation, for which we employ TrajOpt (Schulman et al. 2014) as the nonlinear trajectory optimizer (see Sect. 5.3). We visualize the counterexamples for φ_2, φ_3, φ_8, and φ_{13} in Fig. 16. One can observe that the missing constraints in these candidate LTL formulas accept lower-cost trajectories (achieved for example by not delivering the goods in the desired order, or by not picking up particular objects) which contradict the optimality of the demonstration. We emphasize that our method can ignore the large number of distractor objects. Limiting the expressibility of the DAG by limiting the number of nodes encourages the learned formula to be parsimonious, since the free nodes will be needed to explain demonstrator optimality rather than involving the distractor objects. In the 14th iteration, our method terminates after a total of 5 minutes, returning the true formula φ^*.

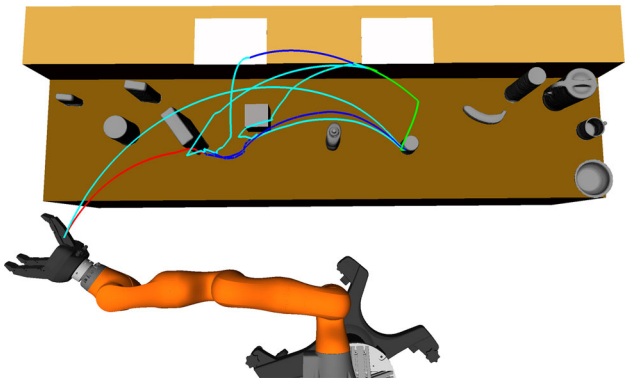

Fig. 16 Counterexample visualization on the object delivery task. The red, green blue, and cyan trajectories correspond to φ_2, φ_3, φ_8, and φ_{13}, respectively, as described in Sect. 10.2 (Color figure online)

Fig. 17 Setup of the object delivery task in the real world. The small brown box corresponds to the small blue box in the VR environment, while the large brown box corresponds to the green box in the VR environment (Color figure online)

10.3 Real-world planning and execution

Now that an LTL formula describing the desired task has been learned, we seek to use the learned formula to plan in the real world. We work with the real-world setup in Fig. 17. This setup has different furniture and object configurations compared to the VR demonstration environment. However, recall that since the learned LTL formula is parameterized by the APs, the learned LTL formula is not hardcoded to specific configurations and can handle changes in the object locations.

To reflect the realistic situation where the robot may be tasked to find and deliver a set of objects scattered across the workspace with *a priori* unknown locations, we assume that

(a) (b) (c)

Fig. 18 Object segmentation. **a** RGBD data provided by the Kinect sensor. **b** Segmented image. **c** Segmented point cloud, which is used to infer object poses

the locations of the delivery regions and the movable box are known, while the YCB objects have unknown location. The movable blue box in the VR environment corresponds to the small brown box on the left in Fig. 17, while the green box in the VR environment corresponds to the big brown box on the right in Fig. 17.

To apply our learned LTL formula, we first estimate the poses of the YCB objects using RGBD (image and point cloud) data provided by a Kinect sensor mounted above the base of the arm. We do so by leveraging the deep learning-based object segmentation framework in Zhou et al. (2019) and train it on the YCB object dataset. The trained network takes the Kinect RGBD data as input and returns a segmented point cloud (Fig. 18). We use the iterative closest point (ICP) algorithm (Rusu and Cousins 2011) with 1000 random initializations to estimate the object poses from the segmented point cloud by fitting them to the source point clouds. We visualize the objects at their estimated poses in an Openrave environment, which we also use for trajectory planning (Fig. 19). We note that due to occlusions and sensor noise present in the point cloud data, the poses recovered for the objects further from the Kinect can suffer from rotational inaccuracies (e.g. the mustard bottle is upside down and the pitcher is rotated around 90 degrees). While this degree of pose accuracy is sufficient to complete our task, we also note that more sophisticated methods can be employed (e.g. Deng et al. 2019, which provides good pose recovery on the YCB dataset in the presence of occlusions and object symmetry).

Now that the object poses have been determined (and thus so have the APs), we can employ the learned LTL formula to plan in the real environment. To do so, we solve Problem 1 for $\varphi(\theta^s, \hat{\theta}^p)$ using the approach detailed in Sect. 5.3. Specifically, we construct a high-level plan $\mathbf{Z}$ by solving a MILP, and then find a low-level joint trajectory which is consistent with $\mathbf{Z}$ with the trajectory optimization algorithm TrajOpt (Schulman et al. 2014). Like for the counterexample generation, we choose TrajOpt instead of IPOPT as it is better tuned for manipulation in cluttered environments. Snapshots

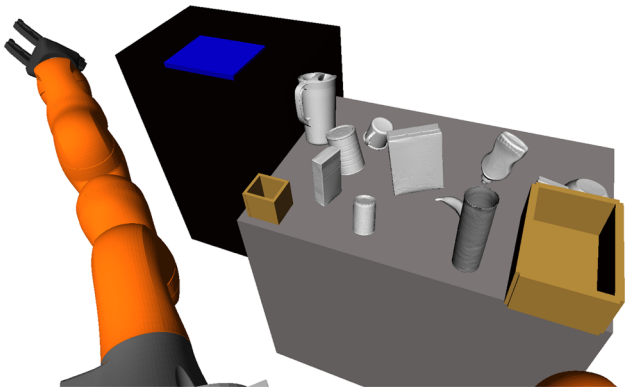

Fig. 19 Planning environment used. Object poses are recovered from the segmented depth cloud by running ICP

of the executed plan are presented in Fig. 20. Please see the supplementary video for a full visualization.

Overall, this experiment suggests that our learned LTL formulas can be used to transfer complex long-horizon task specifications across environments, and that the method is applicable to high-dimensional robotic systems acting in the real world.

11 Conclusion

This paper presents a method that learns LTL formulas with unknown atomic propositions and logical structure from only positive demonstrations, assuming the demonstrator is optimizing an uncertain cost function. We leverage both implicit and explicit optimality conditions on the demonstrations, namely the KKT conditions and algorithmically-generated lower-cost counterexample trajectories, respectively, in order to reduce the hypothesis space of LTL specifications consistent with the demonstrations. The generated lower-cost counterexample trajectories, together with the rejected candidate LTL formulas which admitted them, are concrete examples of the alternative behaviors and task specifications

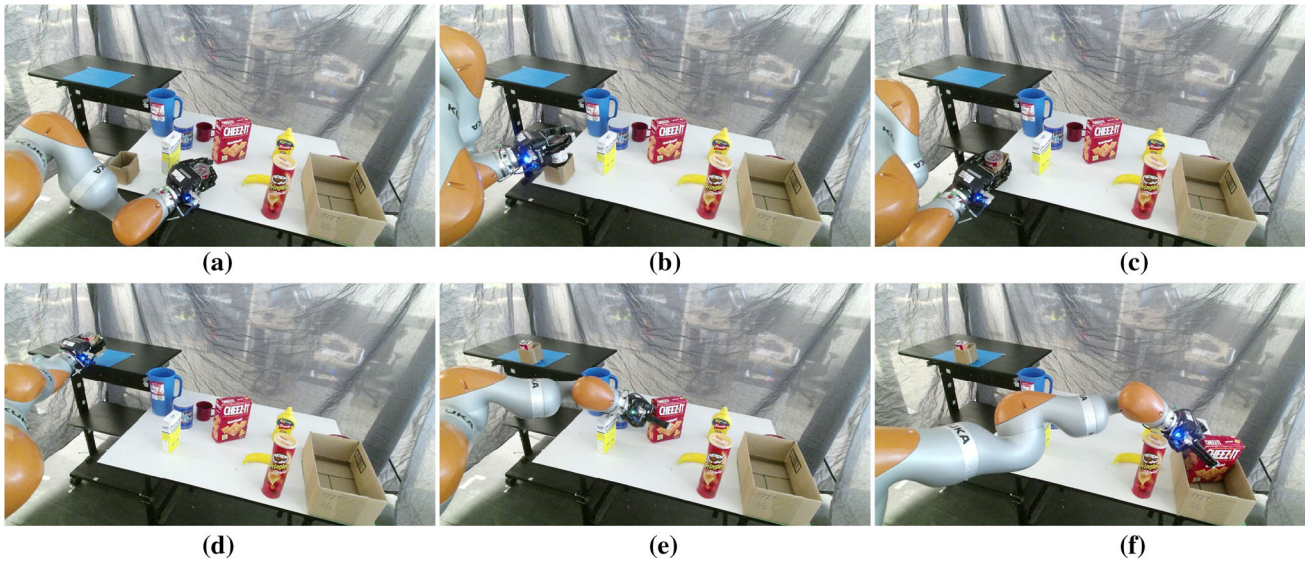

Fig. 20 Executed trajectory on the real robot. The robot first grasps the tomato soup **a**, moves to place it inside the movable box **b**, drops the soup into the box and grasps the loaded box **c**, and moves the loaded the box to the blue delivery region **d**. The robot then moves to grasp the Cheez-It box **e**, and finally places it in the box located at the green delivery region

rejected by our method, which can make our approach more explainable for an end user. We also derive theoretical guarantees for our method and demonstrate its applicability across a wide range of experiments in simulation and hardware. Specifically, we show that our method outperforms baseline approaches (Sect. 9.1), can learn abstract high-level task structure shared across demonstrations, which can transfer to tasks in different environments (Sects. 9.3 and 10), and scales to high-dimensional systems in simulation (Sects. 9.4 and 9.5) and in the real world (Sect. 10).

In future work, we aim to robustify our method to mislabeled demonstrations, explicitly consider demonstration suboptimality arising from risk, and reduce our method's computation time. We are also interested in integrating the methods presented in this paper with our recent results on uncertainty-aware constraint learning (Chou et al. 2020b) in order to plan with uncertain LTL formulas.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10514-021-10004-x.

Acknowledgements The authors thank Daniel Neider for insightful discussions and members of the Autonomous Robotic Manipulation (ARM) Lab for assistance and advice on the physical experiment. This research was supported in part by an NDSEG fellowship, NSF Grants IIS-1750489 and ECCS-1553873, and ONR Grants N00014-17-1-2050, N00014-18-1-2501, and N00014-21-1-2118.

References

Abbeel, P., & Ng, A.Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *International conference on machine learning (ICML)*.

Annpureddy, Y., Liu, C., Fainekos, G.E., & Sankaranarayanan, S. (2011). S-taliro: A tool for temporal logic falsification for hybrid systems. In *17th international conference on tools and algorithms for the construction and analysis of systems, TACAS*, pp. 254–257.

Araki, B., Vodrahalli, K., Leech, T., Vasile, C.I., Donahue, M., & Rus, D. (2019). Learning to plan with logical automata. In *Robotics: Science and systems XV*.

Argall, B., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*, 469–483.

Baier, C., & Katoen, J.-P. (2008). *Principles of model checking*. Cambridge: MIT Press.

Bakhirkin, A., Ferrère, T., & Maler, O. (2018). Efficient parametric identification for STL. In *Proceedings of the 21st international conference on hybrid systems: Computation and control*, pp. 177–186.

Bertsimas, D., & Tsitsiklis, J. (1997). *Introduction to linear optimization* (1st ed.). Belmont: Athena Scientific. ISBN 1886529191.

Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., & Schuppan, V. (2006). Linear encodings of bounded LTL model checking. Logical Methods in Computer Science, 2(5).

Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., & Belta, C. (2016). A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th international conference on hybrid systems: Computation and control, HSCC 2016*, pp. 1–10.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. New York: Cambridge University Press. ISBN 0521833787.

Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., & Bortolussi, L. (2014). Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Tech-*

niques and Applications—6th International Symposium, ISoLA 2014, pp. 391–403.

Calinon, S., & Billard, A. (2008). A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *International Conference on Intelligent Robots and Systems (IROS)*.

Çalli, B., Singh, A., Bruce, J., Walsman, A., Konolige, K., Srinivasa, S. S., et al. (2017). Yale-cmu-berkeley dataset for robotic manipulation research. *International Journal of Robotics Research*, *36*(3), 261–268.

Camacho, A., & McIlraith, S.A. (2019). Learning interpretable models expressed in linear temporal logic. In *Proceedings of the twenty-ninth international conference on automated planning and scheduling, ICAPS 2018*, pp. 621–630.

Chou, G., Berenson, D., & Ozay, N. (2018). Learning constraints from demonstrations. Workshop on the Algorithmic Foundations of Robotics (WAFR), arXiv:1812.07084.

Chou, G., Ozay, N., & Berenson, D. (2019). Learning parametric constraints in high dimensions from demonstrations. In *3rd Conference on Robot Learning (CoRL)*, arXiv:1910.03477.

Chou, G., Ozay, N., & Berenson, D. (2020a). Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations. In *Proceedings of robotics: Science and systems*, Corvalis, Oregon, USA.

Chou, G., Ozay, N., & Berenson, D. (2020b). Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations. In *4th Conference on Robot Learning (CoRL)*. arXiv:2011.04141.

Chou, G., Ozay, N., & Berenson, D.. (2020c). Learning constraints from locally-optimal demonstrations under cost function uncertainty. In *Robotics and Automation Letters (RA-L)*, arXiv:2001.09336.

De Haan, L., & Ferreira, A. (2007). *Extreme value theory: An introduction*. Berlin: Springer.

Demri, S., & Schnoebelen, P. (2002). The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, *174*(1), 84–103.

Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., & Fox, D. (2019). Poserbpf: A rao-blackwellized particle filter for 6d object pose estimation. In *Robotics: Science and Systems XV*.

Englert, P., Vien, N. A., & Toussaint, M. (2017). Inverse kkt: Learning cost functions of manipulation tasks from demonstrations. *International Journal of Robotics Research (IJRR)*, *36*(13–14), 1474–1488.

Fu, J., Papusha, I., & Topcu, U. (2017). Sampling-based approximate optimal control under temporal logic constraints. In *Proceedings of the 20th international conference on hybrid systems: Computation and control, HSCC 2017*, pp. 227–235.

Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., & Zuliani, P. (2009). A bayesian approach to model checking biological systems. In *7th International conference on computational methods in systems biology, CMSB 2009*, pp. 218–234.

Jha, S. (2017). susmitjha/telex. https://github.com/susmitjha/TeLEX.

Jha, S., Tiwari, A., Seshia, S. A., Sahai, T., & Shankar, N. (2019). Telex: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, *54*(3), 364–387.

Johnson, M., Aghasadeghi, N., & Bretl, T. (2013). Inverse optimal control for deterministic continuous-time nonlinear systems. In *IEEE Conference on Decision and Control (CDC)*.

Keshavarz, A., Wang, Y., & Boyd, S.P. (2011). Imputing a convex objective function. In *IEEE International Symposium on Intelligent Control (ISIC)*, pp. 613–619. IEEE

Knuth, C., Chou, G., Ozay, N., & Berenson, D. (2021). Planning with learned dynamics: Probabilistic guarantees on safety and reachability via lipschitz constants. IEEE Robotics and Automation Letters (RA-L).

Kong, Z., Jones, A., Ayala, A.M., Gol, E.A., & Belta, C. (2014). Temporal logic inference for classification and prediction from data. In *17th International conference on hybrid systems: Computation and control (part of CPS Week), HSCC'14*, pp. 273–282.

Kong, Z., Jones, A., & Belta, C. (2017). Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, *62*(3), 1210–1222.

Kress-Gazit, H., Fainekos, G. E., & Pappas, G. J. (2009). Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, *25*(6), 1370–1381.

Krishnan, S., Garg, A., Liaw, R., Thananjeyan, B., Miller, L., Pokorny, F. T., et al. (2019). SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. *International Journal of Robotics Research (IJRR)*, *38*(2–3), 126–145.

Leung, K., Aréchiga, N., & Pavone, M. (2019). Backpropagation for parametric STL. In *2019 IEEE Intelligent Vehicles Symposium, IV*, pp. 185–192.

Li, L., & Fu, J. (2017). Sampling-based approximate optimal temporal logic planning. In *2017 IEEE International Conference on Robotics and Automation, ICRA*, pp. 1328–1335.

Neider, D., & Gavran, I. (2018). Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018*, pp. 1–10.

Ng, A.Y., & Russell, S.J. (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 663–670, San Francisco, CA, USA.

Pais, A. L., Umezawa, K., Nakamura, Y., Billard, A. (2013). Learning robot skills through motion segmentation and constraints extraction. ACM/IEEE International Conference on Human-Robot Interaction (HRI).

Papusha, I., Wen, M., & Topcu, U. (2018). Inverse optimal control with regular language specifications. In *2018 Annual American Control Conference*. *ACC*, *2018*, 770–777.

Ranchod, P., Rosman, B., & Konidaris, G.D. (2015). Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015*, pp. 471–477.

Ratliff, N.D., Andrew Bagnell, J., & Zinkevich, M. (2006). Maximum margin planning. In *Proceedings of the twenty-third international conference on machine learning (ICML 2006)*, pp. 729–736.

Rusu, R.B., & Cousins, S. (2011). 3d is here: Point cloud library (PCL). In *IEEE international conference on robotics and automation, ICRA 2011*. IEEE.

Sabatino, F. (2015). Quadrotor control: modeling, nonlinearcontrol design, and simulation.

Sadigh, D., Dragan, A.D., Sastry, S., & Seshia, S.A. (2017). Active preference-based learning of reward functions. In *Robotics: Science and Systems XIII*.

Schulman, J., Duan, Y., Ho, J., Lee, A. X., Awwal, I., Bradlow, H., et al. (2014). Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research*, *33*(9), 1251–1270.

Shah, A., Kamath, P., Shah, J.A., & Li, S. (2018). Bayesian inference of temporal task specifications from demonstrations. In *Advances in Neural Information Processing Systems (NeurIPS) 2018*, pp. 3808–3817.

Vaidyanathan, P., Ivison, R., Bombara, G., DeLateur, N.A., Weiss, R., Densmore, D., & Belta, C. (2017). Grid-based temporal logic inference. In *56th IEEE Annual Conference on Decision and Control, CDC 2017*, pp. 5354–5359.

Vazquez-Chanlatte, M., Jha, S., Tiwari, A., Ho, M.K., & Seshia, S.A. (2018). Learning task specifications from demonstrations. In *Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 5372–5382.

Wächter, A., & Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57.

Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., & Daniel, L. (2018). Evaluating the robustness of neural networks: An extreme value theory approach. International Conference on Learning Representations (ICLR).

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

Xu, Z., Nettekoven, A.J., Agung Julius, A., & Topcu, U. (2019). Graph temporal logic inference for classification and identification. In *58th IEEE Conference on Decision and Control, CDC 2019*, pp. 4761–4768. IEEE.

Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., et al. (2019). Semantic understanding of scenes through the ADE20K dataset. *International Journal of Computer Vision*, *127*(3), 302–321.

Zhou, W., & Li, W. (2018). Safety-aware apprenticeship learning. In *30th International Conference on Computer Aided Verification, CAV 2018*, pp. 662–680.

Glen Chou received dual B.Sc. degrees in electrical engineering and computer science and mechanical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 2017. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Michigan, Ann Arbor, MI, USA. He is the recipient of the National Defense Science and Engineering Graduate (NDSEG) Fellowship. His current research interests include machine learning, optimization, motion planning, and control, with particular emphasis on safe, adaptive learning and control for robotic systems.

Necmiye Ozay received the B.Sc. degree from Bogazici University, Istanbul, Turkey, in 2004, the M.Sc. degree from the Pennsylvania State University, University Park, University Park, PA, USA, in 2006, and the Ph.D. degree from Northeastern University, Boston, PA, USA, in 2010, all in electrical engineering. She was a Postdoctoral Scholar with the California Institute of Technology, Pasadena, CA, USA, between 2010 and 2013. In 2013, she joined the University of Michigan, Ann Arbor, Ann Arbor, MI, USA, where she is currently an Associate Professor of Electrical Engineering and Computer Science and a core faculty with the Michigan Robotics Institute. Her research interests include hybrid dynamical systems, control, optimization and formal methods with applications in cyber-physical systems, system identification, verification and validation, autonomy, and dynamic data analysis. Dr. Ozay received several awards for her papers including a Nonlinear analysis: Hybrid Systems Prize Paper Award during 2014 to 2016. She was the recipient of five young investigator awards, including National Science Foundation (NSF) CAREER, and the 1938E Award and a Henry Russel Award from the University of Michigan for her contributions to teaching and research.

Dmitry Berenson received a B.Sc. in Electrical Engineering from Cornell University in 2005 and received his Ph.D. degree from the Robotics Institute at Carnegie Mellon University in 2011. He completed a post-doc at UC Berkeley in 2011 and was an Assistant Professor in Robotics Engineering and Computer Science at WPI 2012–2016. He is currently an Associate Professor in the EECS Department and Robotics Institute at the University of Michigan. He received the IEEE RAS Early Career Award and the NSF CAREER award. His current research focuses on motion planning and manipulation.