Inverse Optimal Control with Regular Language Specifications

Ivan Papusha* Min Wen† Ufuk Topcu‡

***Abstract*— Given a description of system dynamics, input constraints, and a cost function, the problem of optimal control is to find a sequence of inputs and a state trajectory that minimizes the total cost. A related problem posed by Kalman in the 1960s, called the inverse problem of optimal control, is to determine the objective function given optimal inputs and state trajectories. In this work, we pose the inverse problem of optimal control under temporal behavior specifications. In our setting, we are given demonstrations of optimal trajectories of a finite transition system, which may come from an expert, simulation, or some other process. In addition to these demonstrations, we are also given extra side information that optimal trajectories must satisfy temporal behavior constraints, expressed as an automaton over the state labels. We explore the value of this temporal side information in imputing an approximately optimal policy from finitely many demonstrations, and give a gridworld example with an eye toward extending the framework to hybrid systems with continuous states.**

I. Introduction

In this work, we study the inverse problem of optimal control, sometimes called *inverse reinforcement learning*, with side specifications. In the ordinary, or *forward*, problem of optimal control, the goal is to find a control policy and an optimal state trajectory that minimize a cost index. In the inverse problem, one is given samples or *demonstrations* of optimal trajectories, or optimal state-action pairs, and the goal is to determine the cost. This inverse problem is an important object of study for several reasons.

First, the inverse setting is a natural way of modeling expert demonstrations. Often when a decision task is too difficult for a computer to solve, an implementation can benefit from external knowledge. If one can determine, or even just approximate, a cost function consistent with the expert demonstrations, then the forward problem can be solved to mimic the expert. In this context, the inverse problem has gained particular attention in the machine learning and robotics communities [1], [2], [3], although the problem has also been studied in the control community as far back as [4], [5, §10.6], and more recently in [6], [7].

Second, the cost function contains information about the high-level task by providing a link between the underlying low level dynamics and a higher planning layer. Accurate determination of the cost can provide insight into why a certain low-level behavior is observed. Some researchers have applied inverse optimality to develop cognitive models and theories of the mind [8], [9].

Even though several formulations have been proposed, the inverse problem of optimal control is ill-posed, because there can be many cost functions consistent with the optimality of the input demonstrations [10]. This drawback typically manifests in the existence of trivial costs, or costs that have no generalization power. Furthermore there is a desire to use inverse optimal control to forego the intricate modeling of complex systems in favor of generalizing from expert examples, however in this case the assumption that the examples are optimal with respect to a cost of a certain form, or *any* cost at all, is often violated. Here, the inverse problem is ill-posed because the expert (typically a human) is not infallible, and may even supply contradictory data.

As there are many cost functions consistent with the expert data, some optimality principle (like Occam's razor) or extra side information must be introduced to pick just one. In past formulations, this side information usually comes in through assumptions on the specific form of the cost or dynamics, such as a specific basis expansion. In this work, we explore a different modality for the introduction of side information—the fact that all trajectories (and not just those belonging to the demonstration set) must satisfy a known temporal behavior specification.

Specifically, we propose that in addition to demonstration trajectories, we are given a finite automaton that describes allowed behaviors in the forward problem. The automaton can come from a temporal logic specification known to be satisfied by the optimal state space trajectories (for example, co-safe LTL [11], [12]), a regular expression, or some other known behavioral encoding that tracks, through an internally forced sequence of discrete mode transitions, the completion of tasks along the way. For the problem of inverse optimal control on discrete transition systems, we ask the question: what is the role of temporal behavior side information in imputing a policy from expert demonstrations? We argue that its role is to impart a *memory* to the imputed policy.

The advantages of this bestowed memory are twofold. The first advantage is computational: a behavioral specification allows us to impute *task*- or *mode*-varying (rather than more generally *time*-varying) policies, reducing the number of free parameters in a principled way, and allowing for a more computationally attractive inverse algorithm. The second advantage is in policy generalization: by formally incorporating known behavioral specifications into the inverse problem, we are able to ascribe a natural task-level separation into the approximation of policies from expert examples.

We first review the (by now) classical inverse problem

*Institute for Computational Engineering and Sciences, University of Texas at Austin, TX. (ipapusha@utexas.edu)

†Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA. (wenm@seas.upenn.edu)

‡Department of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin, TX. (utopcu@utexas.edu)

This work was supported in part by NSF 1646522, ARO W911NF-15-1-0592, DARPA W911NF-16-1-0001, and AFRL FA8650-15-C-2546.

of optimal control, and reformulate it to take into account known temporal behavior specifications in the form of a specification automaton. We present an algorithm for solving this inverse problem by convex optimization, as an alternative to previous work [13], and give examples of how the algorithm would be applied. Specifically, we compare solving the inverse problem with, and without, taking the behavioral specifications into account. Finally, we speculate on how our framework and solution method can be extended to hybrid systems, and provide a critique on our general approach.

II. Inverse Optimal Control

A. Solving inverse problems

Consider the problem of finding a minimum of a function,

$$\text{minimize} \quad f(x,p),$$

with variable $x \in \mathcal{X} \subseteq \mathbf{R}^n$ and parameter $p \in \mathcal{P} \subseteq \mathbf{R}^m$, where $f : \mathcal{X} \times \mathcal{P} \to \mathbf{R}$ is a real-valued differentiable convex function of x for every p, and the set $\mathcal{X}$ is convex. This is the *forward* problem: given a parameter p, determine an optimal point that minimizes $f(\cdot, p)$, as well as the optimal objective value $f^\star$. A simple idea for solving the forward problem is to solve for $x^\star$ in the first-order stationarity condition,

$$\nabla_x f(x^\star, p) = 0, \tag{1}$$

provided $x^\star$ lies in the interior of $\mathcal{X}$.

Now suppose the function f were not known in advance, and we were instead given a sample data set

$$\mathcal{D} = \left\{ (x^{(k)}, p^{(k)}) \mid k = 1, \ldots, N \right\},$$

where the point $x^{(k)}$ is optimal for the corresponding parameter $p^{(k)}$. The *inverse* problem is: given $\mathcal{D}$, can we determine f? The answer to the inverse problem is in general no, because there can be many functions f consistent with a given data set $\mathcal{D}$. However, if we had extra side information about the specific form of f, then we can say more.

For example, we could know that f is convex quadratic, in which case a data set $\mathcal{D}$ consisting of enough linearly independent examples could be used to fully determine f. Whether there is a satisfactory answer to the inverse problem therefore depends on both the provided data $\mathcal{D}$, and the side information known about f.

We can encode our knowledge about f by writing it as a linear combination of known functions,

$$f(x,p) = \sum_{i=1}^{M} \theta_i \phi(x,p),$$

where $\theta_i \in \mathbf{R}$ are real (unknown) coefficients, and $\phi_i : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ are (known) basis functions, for all $i = 1, \ldots, M$.

Since the first-order stationarity condition (1) must hold, we define the stationarity residual vector

$$r_{\text{stat}}^{(k)}(\theta) = \sum_{i=1}^{M} \theta_i \nabla_x \phi(x^{(k)}, p^{(k)}), \quad k = 1, \ldots, N,$$

for every data point $(x^{(k)}, p^{(k)}) \in \mathcal{D}$. Note that with the data set $\mathcal{D}$ fixed, the function $r_{\text{stat}}^{(k)}(\theta)$ is a linear function of θ. The condition that $(x^{(k)}, p^{(k)})$ is an optimal pair is equivalent to $r_{\text{stat}}^{(k)}(\theta) = 0$.

Thus to make a guess for θ, we can try to minimize a sum of penalties on the stationarity residuals,

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{k=1}^{N} \psi(r_{\text{stat}}^{(k)}(\theta)) \\ \text{subject to} & \theta \in \Theta, \end{array} \tag{2}$$

over the variable θ, where we can make the further restriction that θ must belong to some set $\Theta \subseteq \mathbf{R}^M$, and $\psi : \mathbf{R}^n \to \mathbf{R}$ is some convex penalty (e.g., a vector p-norm, $p \geq 1$). In certain applications like compressed sensing we can determine θ exactly, and therefore f exactly, by solving the optimization problem (2), provided ψ and Θ are chosen well.

B. Well-posedness and normalization

The objective in (2) can be trivially made zero by picking $\theta = 0$. This is the *trivial* solution, which is uninformative. One general technique for preventing its appearance, proposed in [7], is to fix a normalization for θ, e.g., by fixing a known weight component $\theta_1 = 1$. This is called a *normalization*.

C. Inverse problems on a discrete space

In the sequel, we assume that the function $f : \mathcal{X} \times \mathcal{P} \to \mathbf{R}$ is not a convex function of its first argument, specifically because the set $\mathcal{X}$ is discrete. In this case, the first-order stationarity condition $\nabla_x f(x^\star, p) = 0$ does not make sense, and we must instead use the optimality condition

$$f(x^\star, p) \leq f(x, p), \quad \text{for all } x \in \mathcal{X}. \tag{3}$$

Translating the optimality condition (3) to basis elements, we say that a weight $\theta \in \Theta$ is *consistent* with the optimality of a data point $(x^{(k)}, p^{(k)}) \in \mathcal{D}$ if

$$\sum_{i=1}^{M} \theta_i \left(\phi_i(x, p^{(k)}) - \phi_i(x^{(k)}, p^{(k)}) \right) \geq 0, \quad \text{for all } x \in \mathcal{X}. \tag{4}$$

This collection of $|\mathcal{X}|$ linear inequalities characterizes objective functions f from the family $\{\sum_{i=1}^{M} \theta_i \phi_i \mid \theta \in \Theta\}$ that are consistent with the optimality of the kth data point. The inverse problem on a discrete space amounts to finding a feasible weight $\theta \in \Theta$ that satisfies the consistency condition (4) for all k.

However, in real applications, it may be impossible to determine a unique consistent weight, because 1) the data $\mathcal{D}$ are typically noisy samples from an expert or oracle that may be only approximately optimal, and 2) in practice we are imputing an objective (see next section). In these cases, we relax the consistency requirement (4) by defining a measure of *in*consistency as follows: note that a weight $\theta \in \Theta$ is not consistent with the optimality of a data point $(x^{(k)}, p^{(k)})$ if there exists a variable $x \in \mathcal{X}$ satisfying

$$r_{\text{cons}}^{(k)}(x, \theta) = \sum_{i=1}^{M} \theta_i \left(\phi_i(x, p^{(k)}) - \phi_i(x^{(k)}, p^{(k)}) \right) < 0.$$

In other words, if the kth residual $r^{(k)}_{\text{cons}}(x,\theta)$ is negative for some $x \in \mathcal{X}$, then that value x certifies an inconsistency of the weight θ with the kth expert demonstration.

We thus define our relaxed measure as the size of the worst-case consistency residual. In the relaxed inverse problem we try to minimize the total inconsistency of the data,

$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{k=1}^{N} \max_{x\in\mathcal{X}_k} \left\{ \left(-r^{(k)}_{\text{cons}}(x,\theta)\right)_+ \right\} \\ \text{subject to} & \theta \in \Theta \end{array} \tag{5}$$

over the variable θ (cf. (2)), where we allow $\mathcal{X}_k \subseteq \mathcal{X}$ to potentially vary with the data points in $\mathcal{D}$. The notation $z_+ = \max\{0, z\}$ refers to the nonnegative (rectified) part of a real number z. Interestingly, similar regret-based measures of inconsistency appear in the rank aggregation literature [14], and as a hinge loss in statistics [15]. Note that the objective is convex in θ, because each term is the pointwise maximum of convex functions of θ. If the set $\Theta \subseteq \mathbf{R}^M$ is convex, then the optimization problem (5) is convex, see [16, §3.2.3].

D. Imputing an objective

We make a subtle but important distinction: if f is a sum of known basis elements, then determining the basis coefficients θ is the inverse problem. However, we can also *assume* that $f \approx \hat{f}$, where $\hat{f}$ is a sum of basis elements, but we make no assertions about the form of f itself. In this case, the approximation $\hat{f}$ is the *imputed* objective function [7].

III. Preliminaries

To make the problem setting concrete, we first review concepts from formal verification. For more extensive background, see [17], [18]. The optimization problem (5) forms the bedrock of our approach to inverse optimal control, thus our eventual goal in defining the role of temporal side information is to build up a useful definition of f, $\mathcal{X}_k$, and Θ. Experts can skip to the next section.

We consider system dynamics expressed as *transition systems* with transition *stage costs* (or *loss* increments), and side information that is revealed through a *deterministic finite automaton* (DFA) that accepts optimal runs of the transition system. Recall that a transition system is a tuple $TS = (S, Act, \to, S_0, AP, L)$, where

- S is a discrete set of states,
- Act is a discrete set of actions,
- $\to \subseteq S \times Act \times S$ is a transition relation,
- $S_0 \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is a state labeling function.

Intuitively, a transition system models system or decision process dynamics as a *path* through a directed graph. The execution starts at a state $s_0 \in S_0$, and evolves according to the transition relation. Every element $(s, \alpha, s') \in \to$ (written as $s \xrightarrow{\alpha} s'$) corresponds to an edge of the directed graph connecting the states s and s' with the action α. Associated with each transition is a stage cost $\ell(s, \alpha, s')$, which is a real number indicating the cost of making that transition. The transition system TS is *deterministic* if $|S_0| \leq 1$, and the set of α-successors of every state $s \in S$,

$$\text{Post}(s, \alpha) = \{s' \in S \mid s \xrightarrow{\alpha} s'\},$$

satisfies $|\text{Post}(s, \alpha)| \leq 1$ for all $s \in S$ and $\alpha \in Act$. The labeling function L maps a state s to the set of logical predicates in AP that are true.

The temporal specifications are provided by a deterministic finite-state automaton (DFA), which is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : Q \times \Sigma \to Q$ is a transition function,
- $Q_0 \subseteq Q$ is a set of start states, and
- $F \subseteq Q$ is a set of final, or *accept* states.

The link between executions of the transition system and runs of the automaton is provided by letting the automaton's alphabet be $\Sigma := 2^{AP}$. (Note that AP is a component of TS, while Σ is a component of A.) Specifically, every valid path of states and actions through a transition system TS

$$s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{T-1}} s_T \tag{6}$$

is associated with a corresponding *trace* (or *word*) of labels (or *letters*) through the automaton A,

$$L(s_0)L(s_1)L(s_2)\ldots L(s_T) \in \Sigma^*. \tag{7}$$

Here, the asterisk (*) denotes the Kleene star, with Σ^* denoting finite words with letters in Σ. Each label $L(s_t) \in \Sigma = 2^{AP}$ is a (possibly empty) subset of AP, indicating which of the atomic propositions in AP are true in the state s_t. For example, if $AP = \{\text{green}, \text{white}, \text{goal}\}$, and some state s_t satisfied the logical predicate $\neg\text{green} \wedge \text{white} \wedge \neg\text{goal}$, then the state's label would be

$$L(s_t) = \{\text{white}\}.$$

Finally, the allowed temporal behavior is encoded in the accept states F of the automaton. For the path (6), we say that its corresponding trace (7) is *accepted* by A if there exists $q_0 \in Q_0$, as well as a sequence of automaton states $q_0, q_1, \ldots, q_{T+1} \in Q$ such that

$$q_{t+1} = \delta(q_t, L(s_t)), \quad t = 0, \ldots, T,$$

and $q_{T+1} \in F$. Otherwise, the trace is not accepted. The set of all traces in Σ^* accepted by A is the *language* $\mathcal{L}(A)$ of the automaton A.

IV. Forward problem

A. Product formulation

Given a transition system $TS = (S, Act, \to, S_0, AP, L)$, a DFA $A = (Q, 2^{AP}, \delta, Q_0, F)$, a transition cost function $\ell : S \times Act \times S \to \mathbf{R}$, an initial state $s_0 \in S_0$, a final goal state $s_f \in S$, and a terminal time T, the forward problem is to find an optimal policy π that minimizes the total transition cost

$$J^\pi(s_0) = \sum_{t=0}^{T-1} \ell(s_t, \mu_t(s_t), s_{t+1}),$$

subject to the following two conditions:

1) *Dynamics*: $s_0 \xrightarrow{\mu_0(s_0)} s_1 \xrightarrow{\mu_1(s_1)} \cdots \xrightarrow{\mu_{T-1}(s_{T-1})} s_T = s_f$, is a path through the transition system TS, and
2) *Temporal behavior*: the sequence of labels visited by the path through TS is accepted by the automaton A,

$$L(s_0)L(s_1)\dots L(s_T) \in \mathcal{L}(A).$$

We treat the policy $\pi = \{\mu_0, \dots, \mu_{T-1}\}$ as the optimization variable, where each $\mu_t : S \to Act$ is a function that assigns an action to every state. We refer to an optimal policy as $\pi^\star = \{\mu_0^\star, \dots, \mu_{T-1}^\star\}$, which has optimal cost $J^\star(s_0)$. Note that the policy is *stationary* if all the μ_ts are the same for every t, i.e., $\pi = \{\mu, \mu, \dots, \mu\}$; otherwise the policy π is *time-varying*. The first condition links the (deterministic) evolution of the transition system state to the policy π. The second condition is equivalent to the existence of a finite run $q_0, q_1, \dots, q_{T+1}$ of the automaton that satisfies $q_{T+1} \in F$.

The forward problem of determining a policy that minimizes the total of all stage costs, subject to dynamics and temporal behavior specifications, is a deterministic shortest path problem on the product space $S \times Q$. This shortest path problem is efficiently solved by dynamic programming (see, e.g. [19], [20])

Specifically, we form a product transition system $TS \otimes A = (S \times Q, Act, \to', S_0', AP', L')$, with a new transition relation $\to'$ defined by the rule

$$(s \xrightarrow{\alpha} s') \wedge (q' = \delta(q, L(s))) \Longrightarrow (s,q) \xrightarrow{\alpha}{}' (s', q'),$$

and a new set of initial states

$$S_0' = \{(s_0, q) \mid s_0 \in S_0, \exists q_0 \in Q_0 \text{ with } q = \delta(q_0, L(q_0))\},$$

cf. [17, §4.2]; the components AP' and L' are irrelevant. We associate the cost $\ell(s, \alpha, s')$ with each new edge $(s,q) \xrightarrow{\alpha}{}' (s', q')$, and find the shortest path (in terms of total cost) from every initial state to (s_f, q_f) for all $q_f \in F$.

Note that for a fixed final horizon T, the *only* way for the temporal behavior specifications to be met is if there exists a path through TS with T transitions, and a corresponding accepting run of A with $T+1$ transitions. In general, the optimal policy is time-varying. If the optimal policy is encoded by a time-varying value function $V_t^\star(s)$, i.e.,

$$\mu_t^\star(s) \in \underset{\{\alpha | s \xrightarrow{\alpha} s'\}}{\operatorname{argmin}} \left\{\ell(s, \alpha, s') + V_{t+1}^\star(s')\right\},$$

and that policy satisfies both the dynamics and temporal behavior specifications, then given an initial state s_0, one can map the time indices

$$t = 0, 1, \dots, T$$

onto some modes of the automaton

$$q = q_1, q_2, \dots, q_{T+1}$$

such that the run $q_0, q_1, \dots, q_{T+1}$ is accepted. In other words, the automaton, in addition to tracking the completion of tasks by accepting state labels, is also *counting time*.

B. Time as an automaton

Let T be a positive integer and define the DFA A_{T+1}^{time} as the automaton that consumes exactly $T+1$ labels, regardless of their value, and stops. Its language is given by $T+1$ repetitions of any element in 2^{AP}, i.e., all subsets of AP,

$$\mathcal{L}(A_{T+1}^{\text{time}}) = (\text{true})^{T+1},$$

see Figure 1. Entrance into the states $q_1, q_2, \dots, q_{T+1}$ can be equated with the passage of time epochs $0, \dots, T$. Thus, an optimal path through a transition system TS subject to the temporal behavior specification given by A_{T+1}^{time} must have length T. In this setting, the optimal value functions $V_t^\star(\cdot)$

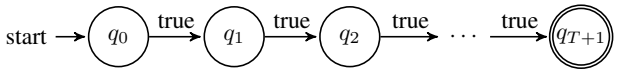


Fig. 1. Automaton A_{T+1}^{time} that consumes exactly $T+1$ symbols

can be re-parameterized as $V^\star(\cdot, q_{t+1})$, where q_{t+1} is the time parameter. The corresponding optimal (mode-varying) policy is given by

$$\mu^\star(s,q) \in \underset{\{\alpha | s \xrightarrow{\alpha} s', q' = \delta(q, L(s))\}}{\operatorname{argmin}} \{\ell(s, \alpha, s') + V^\star(s', q')\}.$$

The finite horizon shortest path problem (without temporal behavior specifications) is therefore a special case of the forward problem, where the temporal behavior is given by the inevitable deterministic progression of states in A_{T+1}^{time}.

C. Mode-varying decision process

Let $V^\star(s,q)$ be the minimum cost-to-go (possibly infinite) starting at any state $(s,q) \in S \times Q$ of the product transition system $TS \otimes A$, when using an optimal mode-varying policy,

$$V^\star(s,q) = \sum_{t=0}^{T-1} \ell(s_t, \mu^\star(s,q), s_{t+1}), \quad \text{where } s_0 = s, q_0 = q.$$

If the value function $V = V^\star$ were known (for example, by solving the forward problem), then the process of determining an optimal action $\alpha^\star = \mu^\star(s,q)$ can be written as a discrete optimization of the form

$$\begin{array}{ll} \text{minimize} & \ell(s, \alpha, s') + V(s', q') \\ \text{subject to} & s \xrightarrow{\alpha} s', \\ & q' = \delta(q, L(s)), \end{array} \tag{8}$$

where the variables are $x = (\alpha, s', q')$ and the parameters are $p = (s,q)$. The decision process (8) is simply a restatement of the dynamic programming principle in terms of the known value function V. To fix the cost-to-go at the accept states of A, we must have $V(s_f, q_f) = 0$ for all $q_f \in F$. Since the optimal action choice $\alpha^\star$ depends on both the transition system state s and the automaton mode q, the mode-varying process (8) can be viewed as time-invariant in the state space $S \times Q$ under the assumption that the optimal policy in the forward problem is time-invariant.

V. Inverse problem

A. Statement

The goal of the inverse problem is to determine (or impute) the unknown objective in the decision process (8) from expert demonstrations. We now make the definitions promised in Section III. First, we make the basis approximation choice $\hat{V}(s,q) = \theta^T \phi(s,q)$, and model the decision process (8) instead as the approximate dynamic programming (ADP) process

$$\begin{array}{ll} \text{minimize} & \ell(s,\alpha,s') + \hat{V}(s',q') \\ \text{subject to} & s \xrightarrow{\alpha} s', \\ & q' = \delta(q, L(s)), \end{array} \tag{9}$$

where the variables are $x = (\alpha, s', q')$ and the parameters are $p = (s,q)$. We must be offered the following training data as examples:

$$\mathcal{D} = \left\{ \left((\alpha^{(k)}, s'^{(k)}, q'^{(k)}),\ (s^{(k)}, q^{(k)}) \right) \mid k = 1, \ldots, N \right\}.$$

The first and second components of the ordered pairs in $\mathcal{D}$ correspond to the variable $x \in \mathcal{X}_k$ and parameter $p \in \mathcal{P}$ in (5), respectively. Note that the side information is included in the training data. For every presented optimal state and action in TS, the training data also contain information about the appropriate mode transitions in A. This task *memory* about the expert examples must be provided for the bahvioral side information to have a benefit to the imputed policy.

B. Solution by convex optimization

The problem is solved by determining a weight $\theta \in \Theta$ for which the ADP process (9) mimics the expert decision process (8) as closely as possible. This is done by instantiating the convex optimization problem (5) with the specific choices

$$f(x,p) = \ell(s,\alpha,s') + \hat{V}(s',q'), \quad \hat{V}(s,q) = \sum_{i=1}^{M} \theta_i \phi_i(s,q),$$

which corresponds to picking a normalization where the first component of f, i.e., the stage cost function ℓ, is known.

C. Remarks

1) Freedom in choosing approximating process: For ease of presentation, we chose the approximating process (9) to have a known stage cost function and an unknown value function, a formulation typically ascribed to simpler ADP methods (such as projected value iteration [19], [20]).

However, we can equally well rewrite the approximating process (9) by parameterizing the stage cost function as well,

$$\begin{array}{ll} \text{minimize} & \hat{\ell}(s,\alpha,s') + \hat{V}(s',q') \\ \text{subject to} & s \xrightarrow{\alpha} s', \\ & q' = \delta(q, L(s)), \end{array}$$

with the same variables and training data $\mathcal{D}$. This can be interpreted as asserting an approximation

$$\hat{\ell}(s,\alpha,s') = \theta_\ell^T \phi_\ell(s,\alpha,s'),$$

and imputing a combined weight vector $(\theta_\ell, \theta) \in \Theta_\ell \times \Theta$ to make the approximation as close to the data set as possible. If the stage cost and value functions are nonnegative, the inverse problem is ill-posed; for example, if $(\hat{\ell}, \hat{V})$ is a valid pair of incremental loss and value function estimates, then so is $(\hat{\ell} + c_1, \hat{V} + c_2)$ for any constants c_1, c_2.

By picking a known stage cost function ℓ instead of trying to determine one consistent with the data, we ensure that the objective in the ADP process (9) is not homogeneous in the combined weights θ, thus attempting to guarantee a normalization for which the imputed objective is nontrivial.

A middle ground is possible: we partially specify the stage cost function as a sum of known and unknown parts,

$$\hat{\ell}(s,\alpha,s') = \ell_0(s,\alpha,s') + \theta_\ell^T \phi_\ell(s,\alpha,s'),$$

where the known part ℓ_0 serves as a normalization, and the unknown part coefficients are to be determined.

Even more generally, we can rewrite the value function at the landing state, $V(s',q')$ as a sum of known and unknown parts, as well. In this case, the expert data must also contain known components of landing cost-to-go values.

2) Data set derivability: The data set $\mathcal{D}$ can in practice be obtained from simpler data sets. For example, because both TS and A are deterministic, the landing states $s'^{(k)}$ and $q'^{(k)}$ in $\mathcal{D}$ are superfluous given the other components, thus we can derive $\mathcal{D}$ from the simpler data set

$$\mathcal{D}' = \left\{ \left(\alpha^{(k)},\ (s^{(k)}, q^{(k)}) \right) \mid k = 1, \ldots, N \right\}$$

by simulating the transition system. The component $q^{(k)}$ encodes the task memory. If we further know that the demonstration examples are entire trajectories (instead of scattered optimal state-action pairs), then the task memory is not needed from the expert, i.e., we can derive $\mathcal{D}$ from the data set of optimal state-actions

$$\mathcal{D}'' = \left\{ (\alpha^{(k)},\ s^{(k)}) \mid k = 1, \ldots, N \right\}$$

by simulating the expert trajectory in lockstep with the (known) automaton.

3) Guaranteed temporal behavior: To model the fact that the value function is fixed at the accept states of $TS \otimes A$, we restrict the allowed weights in the inverse problem (5) to the affine subspace

$$\Theta_0 = \Big\{ \theta \in \mathbf{R}^M \mid \sum_{i=1}^{M} \theta_i \phi_i(s_f, q_f) = 0, \quad \text{for all } (s_f, q_f) \in S \times F \Big\}. \tag{10}$$

This restriction guarantees that the imputed objective leads to behaviors in the product space $S \times Q$ that are accepted by the specification automaton. In general, we can restrict the allowed weights to any set $\Theta = \Theta_0 \cap \Theta_1$, as long as Θ_1 is another convex subset of $\mathbf{R}^M$.

4) Penalizing data inconsistency: For a given imputed value function $\hat{V}$ in the baseline inverse problem, the existence of a better (in total cost) state-action pair $x = (\alpha, s') \in \mathcal{X}_k$ at a given state $s^{(k)}$ than the alleged optimal state-action pair $x^{(k)} = (\alpha^{(k)}, s'^{(k)})$ indicates a breakdown of the dynamic programming principle, because the stage cost plus cost-to-go can be made smaller by choosing x instead of $x^{(k)}$,

$$\ell(s, \alpha, s') + \hat{V}(s') < \ell(s, \alpha^{(k)}, s'^{(k)}) + \hat{V}(s'^{(k)}).$$

In this case, either the value function $\hat{V}$ (and hence θ) is wrong, or the "optimal" demonstration $x^{(k)}$ is a mistake. The objective in (5) remains agnostic to these alternatives by penalizing, through each term

$$\begin{aligned}&\max_{x \in \mathcal{X}_k} \left\{ \left(-r^{(k)}_{\text{cons}}(x, \theta) \right)_+ \right\} \\ &= \max_{\alpha, s'} \big\{ \max(0, \ell(s, \alpha^{(k)}, s'^{(k)}) + \hat{V}_\theta(s'^{(k)}) \\ &\qquad - \ell(s, \alpha, s') - \hat{V}_\theta(s')) \big\}\end{aligned}$$

the worst-case "regret" of a policy choosing a state-action (α, s') at $s^{(k)}$ instead of $(\alpha^{(k)}, s'^{(k)})$. Here, we write $\hat{V}_\theta$ to make explicit the dependence of the imputed value function on the weights θ to be determined.

We can interpret the objective in (5) as a game between a rational expert providing (through $\mathcal{D}$) optimal state-action pairs, and a student, who tries to approximate the expert's policy by looking at examples where the two differ on the choice of optimal action. Wherever they differ, the student tries to make the difference in imputed cost small.

Other measures of inconsistency between the expert and the student are possible depending on the broad dynamics setting. For example, if the dynamics are stochastic, we can minimize the expected regret, or maximize the likelihood of the observed data [6].

VI. Value of Side Information

A. Twofold comparison

Returning to our original motivation, we ask the question: can side information, provided as temporal behavior constraints that optimal runs of the transition system must satisfy, be used to our advantage when attempting to recover the stage cost function from expert training data? We argue that the answer is yes. The specific value of the temporal side information is in providing a task-level, instead of a time-based, memory.

To illustrate, we compare an implementation of baseline inverse optimal control to inverse optimal control with available temporal behavior side information. Note that in the baseline algorithm, we are looking for an optimal time-invariant strategy, ignoring any behavioral side information; this would be expected in a classical application of inverse optimal control or reward learning. We could impute a general time-varying policy, however the number of free parameters grows quickly with the number of timesteps, and in most practical applications the time horizon is either unknown or very large.

Algorithm: Baseline inverse optimal control
given: TS, $\Theta \subseteq \mathbf{R}^M$, $\ell : S \times Act \times S \to \mathbf{R}$, $s_f \in S$, $\{\phi_i : S \to \mathbf{R}\}_{i=1}^M$, and $\mathcal{D} = \left\{ \left((\alpha^{(k)}, s'^{(k)}), s^{(k)} \right) \right\}_{k=1}^N$
output: imputed value function $\hat{V}(s)$
begin
1. **for each** $k = 1, \ldots, N$:
   $x^{(k)} := (\alpha^{(k)}, s'^{(k)})$, $\quad p^{(k)} := s^{(k)}$,
   $\mathcal{X}_k := \{(\alpha, s') \mid s^{(k)} \xrightarrow{\alpha} s'\}$,
   define $f(x, p^{(k)}) := \ell(s^{(k)}, \alpha, s') + \hat{V}(s')$,
   for all $x = (\alpha, s') \in \mathcal{X}_k$
   end for
2. solve convex optimization problem (5)
   for an optimal weight $\theta^\star$
3. **return** $\hat{V}(s) = \sum_{i=1}^M \theta_i^\star \phi_i(s)$
end

Next, we present the algorithm that incorporates behavioral side information. Note that the differences from the baseline are in the format of the training data, the availability of the behavioral specification automaton A, and the "hybrid" (i.e., using the product state space $S \times Q$) nature of the imputed value function.

Algorithm: Inverse optimal control with side information
given: TS, A, $\Theta \subseteq \mathbf{R}^M$, $\ell : S \times Act \times S \to \mathbf{R}$, $s_f \in S$, $\{\phi_i : S \times Q \to \mathbf{R}\}_{i=1}^M$,
and $\mathcal{D} = \left\{ \left((\alpha^{(k)}, s'^{(k)}, q'^{(k)}), (s^{(k)}, q^{(k)}) \right) \right\}_{k=1}^N$
output: imputed value function $\hat{V}(s, q)$
begin
1. **for each** $k = 1, \ldots, N$:
   $x^{(k)} := (\alpha^{(k)}, s'^{(k)}, q'^{(k)})$, $\quad p^{(k)} := (s^{(k)}, q^{(k)})$,
   $\mathcal{X}_k := \{(\alpha, s', q') \mid s^{(k)} \xrightarrow{\alpha} s', q' \in \delta(q^{(k)}, L(s^{(k)}))\}$,
   define $f(x, p^{(k)}) := \ell(s^{(k)}, \alpha, s') + \hat{V}(s', q')$,
   for all $x = (\alpha, s', q') \in \mathcal{X}_k$
   end for
2. solve convex optimization problem (5)
   for an optimal weight $\theta^\star$
3. **return** $\hat{V}(s, q) = \sum_{i=1}^M \theta_i^\star \phi_i(s, q)$
end

B. Gridworld experiment

We use a 6×6 gridworld map for this experiment, as shown in Figure 2. We assume that an agent placed on each square has four actions (up, down, left, right), and all transitions are deterministic. If the agent is at a border square and tries to move towards the border (e.g., move upwards in the top row), the agent will not move. The blue square is the initial state s_0 and the yellow square is the goal state s_f.

1) Behavioral demonstrations: The agent must start from the blue square, pass the two green squares (in any order) while avoiding all red squares, and eventually go to the

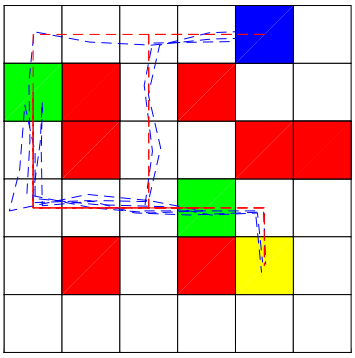


Fig. 2. Gridworld map and three expert demonstrations (red: obstacles, green: goals, blue: initial state, yellow: final state)

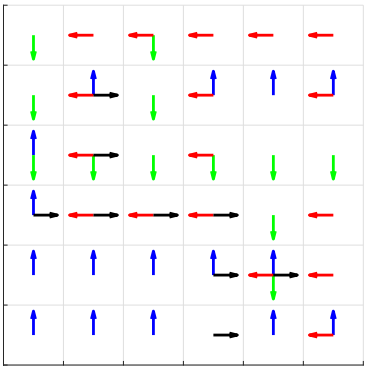


Fig. 3. Learned memoryless strategy without side information. The strategy can miss one green block before reaching the yellow block because of a lack of task memory.

yellow square. Once it reaches the yellow square, the task is complete. The transition system has 36 states, and the specification automaton has 4 states. Thus there are a total of 144 states in the product $TS \otimes A$, and at each state, there are 4 available (deterministic) actions. We show the agent 3 different expert demonstrations that satisfy the behavioral specification.

2) Value function model: We use $M = 5$ different features as basis functions of the action-value function, designed as in Table I.

TABLE I

DESIGN OF BASIS FUNCTIONS ϕ_i, $i = 1, \ldots, 5$.

Feature	Description
ϕ_1	Optimal value function corresponding to the stage cost that gives 0 only when the goal state (yellow block with final state q_f) is reached or at the final goal state, otherwise 1.
ϕ_2	Optimal value function corresponding to the reward that gives 0 only when a new intermediate state (green block) is reached or at the final goal state, otherwise 1.
ϕ_3	Optimal value function corresponding to the reward that gives 1 only when an obstacle (red block) is reached, otherwise 0.
ϕ_4	Optimal value function corresponding to the stage cost that gives 0 only at transitions in the demonstration, otherwise 1.
ϕ_5	Optimal value function corresponding to the stage cost that gives 1 to all transitions.

3) Regularizaton: The weights θ are constrained to the unit ball $\Theta_1 = \{\theta \in \mathbf{R}^M \mid \|\theta\|_2 \leq 1\}$ for the baseline test. With side information, the weights must be in $\Theta_0 \cap \Theta_1$, see eq. (10).

C. Simulation without side information

In this case, the feature ϕ_2 corresponds to the stage cost function, such that any transition that goes to a green block has loss 0, otherwise 1. Using the baseline algorithm, the learned weight vector is

$$\theta = (0, 0, 0.2562, 0.9528, 0),$$

and the learned optimal non-deterministic strategy is shown in Figure 3. Note that following the learned strategy an agent can miss one green block before reaching the yellow block, or even get stuck, depending on its initial off-policy location.

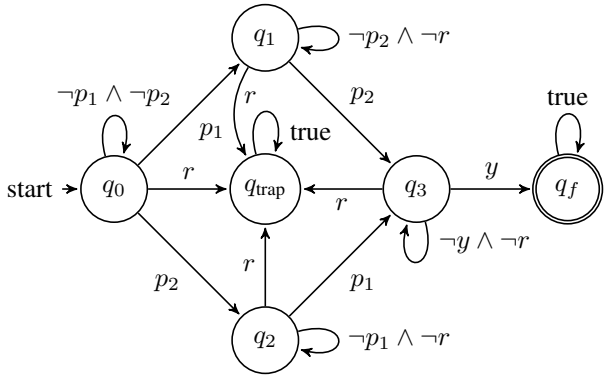


Fig. 4. Side information: a DFA constructed for the example. The atomic proposition p_1 is true whenever the agent is in the green block near the yellow block, and p_2 is true in the other green block; y or r is true if and only if the agent is visiting a yellow or a red block.

D. Simulation with side information

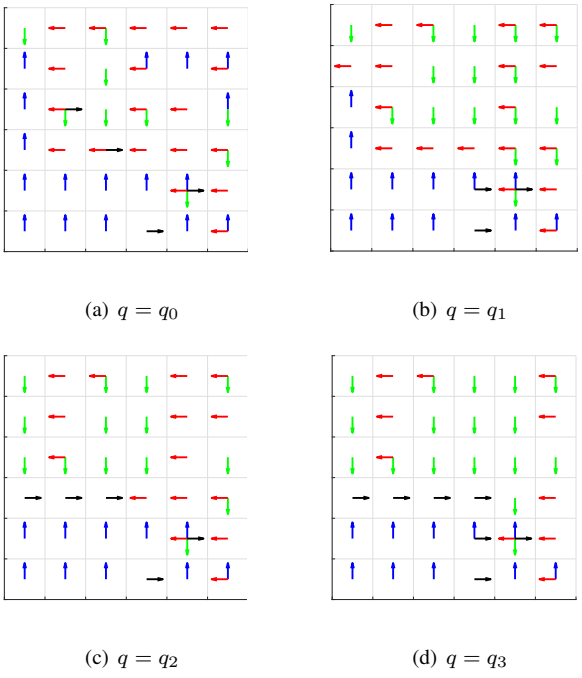
In this case, the side information is given as the DFA shown in Figure 4. Following the side information aware algorithm, we obtain the learned weight vector

$$\theta = (0.0067, 0.0720, 0.2169, 0.9586, -0.0787),$$

which imputes the mode-dependent strategy shown in Figure 5. The task is successfully implemented with this strategy, because the DFA is able to track which green squares have been visited in which order, and whether it is possible to move on to the yellow square.

VII. CONCLUSION

In this work, we presented a general framework for imposing behavioral specifications on policies learned from expert demonstrations. We showed that temporal side information in the form of a specification automaton can be used to effectively constrain the number of free parameters in an inverse optimal control problem, allowing imputed policies to have a task memory—rather then a time-based memory, or no memory at all. It remains to be seen whether this framework can be put to practical use in dynamically interesting applications. Although we have treated the problem entirely using



(a) $q = q_0$ (b) $q = q_1$

(c) $q = q_2$ (d) $q = q_3$

Fig. 5. Learned strategy with side information

deterministic finite-state transition systems and policies, we can envision several extensions. The most clear extension is to consider continuous dynamics, in which case the policy is a hybrid policy over a mixed continuous- (dynamics) discrete- (behavior automaton) space [21], [22], [23]. By choosing to present discrete transition system dynamics, we have sidestepped the important technicalities of hybrid systems. Our choice in this respect was deliberate—it allowed us to study the value of temporal side information without technical distractions. Another clear direction is to rewrite the dynamics as a Markov decision process, and allow for randomized (instead of deterministic) imputed policies. With this modeling choice, we can perhaps better interpret inconsistent demonstrations in Bayesian or maximum likelihood frameworks, e.g., [6], [13].

Key challenges still remain both in scalability and practicality. We showed a fairly small gridworld example; however it is well known that automaton size grows at least exponentially with the size of a regular expression or temporal logic formula that defines it. Furthermore, since the specification automaton must be deterministic, another exponential factor must be added to determinize useful specifications. To see what we mean, we invite the assiduous reader to compute the size of a gridworld specification automaton encoding the side information that the forward problem is a traveling salesman trajectory. Thus while it may be beneficial to use a task-level encoding in some cases, we might still be forced to look for a memoryless, or time-parameterized strategy in others.

REFERENCES

[1] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*. ACM, 2004, p. 1.

[2] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *International Conference on Machine learning (ICML)*. ACM, 2006, pp. 729–736.

[3] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, June 2010.

[4] R. E. Kalman, "When is a linear control system optimal?" *Journal of Basic Engineering*, vol. 86, no. 1, pp. 51–60, Mar. 1964.

[5] S. P. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, ser. Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, 1994, vol. 15.

[6] K. Dvijotham and E. Todorov, "Inverse optimal control with linearly-solvable MDPs," in *International Conference on Machine learning (ICML)*, 2010, pp. 335–342.

[7] A. Keshavarz, Y. Wang, and S. Boyd, "Imputing a convex objective function," in *IEEE International Symposium on Intelligent Control (ISIC)*, Sept. 2011, pp. 613–619, part of IEEE Multi-Conference on Systems and Control (MSC).

[8] C. L. Baker, J. B. Tenenbaum, and R. R. Saxe, "Goal inference as inverse planning," in *Conference of the Cognitive Science Society (CogSci)*, 2007.

[9] K. J. Arrow, "A difficulty in the concept of social welfare," *Journal of Political Economy*, vol. 58, no. 4, pp. 328–346, 1950.

[10] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2000, pp. 663–670.

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

[12] T. Latvala, "Efficient model checking of safety properties," in *International SPIN Workshop on Model Checking of Software*, ser. Lecture Notes in Computer Science, T. Ball and S. K. Rajamani, Eds. Springer, 2003, vol. 2648, pp. 74–88.

[13] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in *International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 2017, pp. 3055–3061.

[14] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank aggregation methods for the web," in *International Conference on World Wide Web (WWW)*. ACM, 2001, pp. 613–622.

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., ser. Springer Series in Statistics. Springer, 2009.

[16] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[17] C. Baier and J.-P. Katoen, *Principles of Model Checking*, ser. Representation and Mind. MIT Press, 2008.

[18] M. Sipser, *Introduction to the Theory of Computation*, 2nd ed. Thomson Course Technology, 2006.

[19] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2005, vol. I and II.

[20] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[21] I. Papusha, J. Fu, U. Topcu, and R. M. Murray, "Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints," in *IEEE Conference on Decision and Control (CDC)*, Dec. 2016, pp. 434–440.

[22] J. Fu, I. Papusha, and U. Topcu, "Sampling-based approximate optimal control under temporal logic constraints," in *ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, Apr. 2017, pp. 227–235.

[23] L. Li and J. Fu, "Sampling-based approximate optimal temporal logic planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1328–1335.

[24] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998.