Guiding Autonomous Agents to Better Behaviors through Human Advice

Gautam Kunapuli
Dept. of Biostat. & Med. Info.
University of Wisconsin-Madison
Madison, WI, USA
kunapg@biostat.wisc.edu

Phillip Odom
School of Info. & Computing
Indiana University
Bloomington, IN, USA
phodom@indiana.edu

Jude W. Shavlik
Dept. of Computer Science
University of Wisconsin-Madison
Madison, WI, USA
shavlik@cs.wisc.edu

Sriraam Natarajan
School of Info. & Computing
Indiana University
Bloomington, IN, USA
natarasr@indiana.edu

***Abstract*—Inverse Reinforcement Learning (IRL) is an approach for domain-reward discovery from demonstration, where an agent mines the reward function of a Markov decision process by observing an expert acting in the domain. In the standard setting, it is assumed that the expert acts (nearly) optimally, and a large number of trajectories, i.e., training examples are available for reward discovery (and consequently, learning domain behavior). These are not practical assumptions: trajectories are often noisy, and there can be a paucity of examples. Our novel approach incorporates advice-giving into the IRL framework to address these issues. Inspired by preference elicitation, a domain expert provides advice on states and actions (features) by stating preferences over them. We evaluate our approach on several domains and show that with small amounts of targeted preference advice, learning is possible from noisy demonstrations, and requires far fewer trajectories compared to simply learning from trajectories alone.**

I. Introduction

There has been a renewed surge in developing autonomous agents based on Reinforcement Learning (RL) [1]. One key attribute of RL agents is that they interact and learn from the environment by obtaining quantitative feedback (reward). In real-world settings, such learning requires a large amount of experience (that is, acting in the world and gathering feedback) before converging on optimal decision-making. This has led to the development of a popular learning paradigm called *learning from demonstration*, where the quantitative measure that influences agent behavior (*reward function*) is mined from trajectories. These trajectories are essentially training examples provided by a demonstrator, often a human domain expert, who acts according to some optimal reward function without ever explicitly articulating it to the learner.

Reinforcement learning is concerned with determining a policy, a conception of how an agent acts in an environment so that it can maximize some notion of reward. The problem is modeled within the Markov decision process (MDP) framework, and a wide variety of algorithms have been developed for finding optimal policies. In this setting, in addition to a description of states, actions and probability transitions, the reward is also specified. However, in several domains, *it is hard, if not impossible, to specify the reward explicitly*. Such cases have long been explored through diverse approaches including learning by observation [2], learning to act [3], programming by example [4], inverse reinforcement learning [5], behavioral cloning [6], imitation learning [7], learning from demonstrations [8], programming by demonstrations [9], and several others.

One such framework is *inverse reinforcement learning* (IRL), where an agent tries to explicitly learn the reward function by observing demonstrations. The observations include the demonstrator's behavior over time (actions), measurements of the demonstrator's sensory inputs, and the model of the environment. In this setting, IRL was studied by Ng and Russell [5], who developed algorithms based on linear programming (LP) for finite state spaces, and Monte Carlo simulation for infinite state spaces. Subsequent research extended their work in different directions including apprenticeship learning [10], Bayesian frameworks [11], parameter tuning of reward functions [12], multi-task settings [13], and partially-observable environments [14].

An important assumption in many of these approaches is *optimality of the training examples, the trajectories*, which are simply sequences of current state and action pairs. With suboptimal trajectories, it is usually assumed that additional knowledge in the form of priors, or a large number of trajectories themselves are available for learning. We take a different approach–we relax the optimality assumption: the demonstrator (who provides the training examples) can be suboptimal, *and* there is a domain expert who provides reasonable advice as *preferences in state/action/reward spaces*. This is inspired by the preference elicitation frameworks in RL [15] and IRL [16], [17] paradigms.

It should be noted that the distinction between a demonstrator (who provides trajectories) and an expert (who provides advice) is not always necessary – sometimes, these roles are can performed by a single teacher. In our setting, the teacher can make errors in demonstration, leading to noisy trajectories; for example, driving with distractions in a driving domain. The teacher also provides reasonably good (but still possibly noisy) advice. The goal is to learn a reward function from *both demonstration and instruction (advice)*. This setting is natural in domains where behaviors are learned from human experts: a single user demonstrating in real-time makes mistakes, but can provide advice carefully after taking various factors into consideration. Such advice can be an especially efficient way to transfer the teacher's accumulated experience about the domain to the learner.

Another motivation is to reduce reliance on demonstration, and a large number of trajectories. Expert advice can result

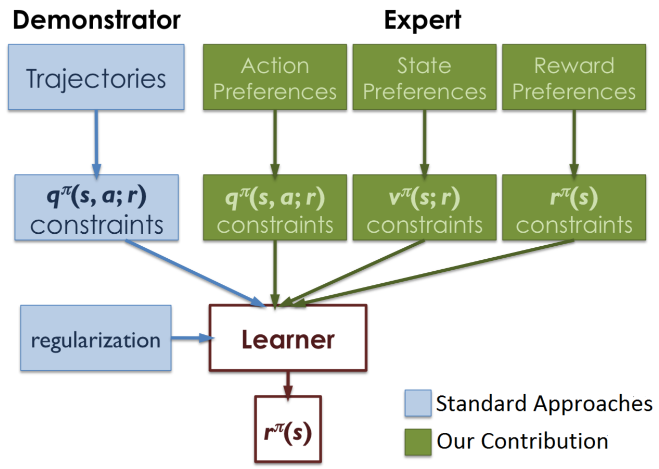

Fig. 1. Schematic of the proposed approach. We distinguish between Demonstrator and Expert to clarify our contributions: *preference advice*, provided by an expert. In practice, trajectories and advice can be provided by a single teacher.

in learning possibly better solutions, *with significantly less data*. This has been observed with other advice-taking mining algorithms such as knowledge-based support vector machines [18], [19] and advice-taking reinforcement learners [20]. Finally, learning from both demonstration, as well as instruction has been shown to be more beneficial than simply learning via only one or the other, e.g., in cognitive-science studies of motor-skill acquisition when learning to play sports [21].

We consider *preference advice*, which is natural to specify in many domains. This is because an expert can easily *tell the learner* about which states/actions are preferable, and which are avoidable. In different situations, one form of advice (i.e., state vs action) may be preferable to others. For instance, it is natural to specify that the agent avoid certain regions of a terrain, or that it prefer being closer to certain regions that are yet unexplored; in this situation *state preferences* are more effective. Alternately, while training car-driving agents in the US, it is natural to specify that turning right but not left is appropriate at a red light, through *action preferences*. We accommodate preference advice of these forms, and pose the advice-taking IRL problem of learning from trajectories *and* advice as a linear program (LP).

The key novelty is that with advice, the learner can *tradeoff between demonstrator and expert* based on whoever is more accurate. After presenting the basic IRL formulation in Section II, we describe our preference advice framework in Section III. We evaluate our methods on four different domains under different settings and show that our proposed approach is capable of learning reasonable rewards in Section IV. We also analyze the impact of advice on learning in the presence of noisy trajectories extensively, and aim to understand the usefulness of advice in several situations. The schematic of our approach is shown in Figure 1.

II. Background and Notation

Scalars are denoted in lowercase (r), vectors in lowercase bold ($\mathbf{r}$) and matrices in uppercase (P). A finite MDP is a tuple $(\mathcal{S}, \mathcal{A}, P_{sa}, \gamma, \mathbf{r})$. Here, $\mathcal{S}$ is a finite set of n states, $\mathcal{A}$ is a finite set of m actions, P_{sa} are the state transition probabilities upon taking action a in state s, $\gamma \in [0, 1)$ is the discount factor and $r : \mathcal{S} \to \mathbb{R}$ is a reward function. A policy is a mapping $\pi : \mathcal{S} \to \mathcal{A}$ that determines how the agent acts under the MDP. The *value function* $v^\pi : \mathcal{S} \to \mathbb{R}$ represents the long-term expected reward for each state s, when executing a policy, π. The values of a state s, and the values of a state-action pair (s, a) under π are given by the following Bellman equations:

$$\begin{aligned} v^\pi(s) &= r(s) + \gamma \textstyle\sum_a \pi(s,a) \displaystyle\sum_{s'} p_{sa}(s')\, v^\pi(s'),\\ q^\pi(s,a) &= r(s) + \gamma \sum_{s'} p_{sa}(s')\, v^\pi(s'). \end{aligned} \tag{1}$$

In finite state spaces, the Bellman equations can be written in vector form [5]:

$$\mathbf{v}^\pi = (I - \gamma P_{a_*})^{-1} \mathbf{r}, \tag{2}$$

$$\mathbf{q}^\pi = \mathbf{r} + \gamma P_a \mathbf{v}^\pi, \tag{3}$$

where P_a are the $n \times n$ *probability transition matrices* for actions $a \in \mathcal{A}$, P_{a_*} is the *expert probability transition matrix* (which is assumed to be optimal in classic IRL), and I is the $n \times n$ identity matrix. Ng and Russell [5, Thm 3] showed that a policy π is *Bellman optimal* if and only if, for all non-optimal actions $a \in \mathcal{A} \setminus a_*$, the reward $\mathbf{r}$ satisfies

$$(P_{a_*} - P_a)(I - \gamma P_{a_*})^{-1} \mathbf{r} \ \geq\ 0. \tag{4}$$

With this characterization IRL can be formulated as an LP. Many rewards $\mathbf{r}$ can satisfy (4); to address this degeneracy and choose between various solutions, we search for a reward that maximizes

$$\sum_{s \in \mathcal{S}} \Big(q^\pi(s, a_*) - \max_{a \in \mathcal{A} \setminus a_*} q^\pi(s, a) \Big). \tag{5}$$

The above equation seeks to maximize the difference between the value of executing the optimal action and the next best among all other actions. This leads to the following IRL formulation, where $B = (I - \gamma P_{a_*})^{-1}$:

$$\begin{aligned} \max_{\mathbf{r}, \xi_i} \quad & -\|\mathbf{r}\|_1 \ +\ \lambda_t \sum_{i=1}^{n} \xi_i \\ \text{s. t.} \quad & (P^i_{a_*} - P^i_a)\, B\, \mathbf{r} \ \geq\ \xi_i, \\ & \qquad\qquad \forall\, a \in \mathcal{A} \setminus a_*,\ i = 1, \ldots, n, \\ & \xi_i \geq 0,\ |r_i| \leq r_{\max}, \qquad \forall\, i = 1, \ldots, n. \end{aligned} \tag{6}$$

The slacks ξ_i measure the difference between the optimal Q-value at state i and all other suboptimal Q-values, $q(i, a_*) - q(i, a)$. The ℓ_1-regularization in the objective helps learn sparse rewards, the effect of which is controlled through $\lambda_t > 0$. Note that (6) is slightly different from the LP described by Ng and Russell [5], as it removes several redundant constraints.

III. Advice Giving for Inverse RL

To illustrate the effect of advice, we introduce a 10×10 pedagogical grid world (Figure 2). The start state is the bottom left of the grid, $(10, 1)$. The agent can execute three actions: `goNorth`, `goEast` and `goNorthEast`, and attempts to reach the goal, which is the top right of the grid, $(1, 10)$. In the advice-taking setting, we wish to learn how to act in a world by observing trajectories (demonstration) and by

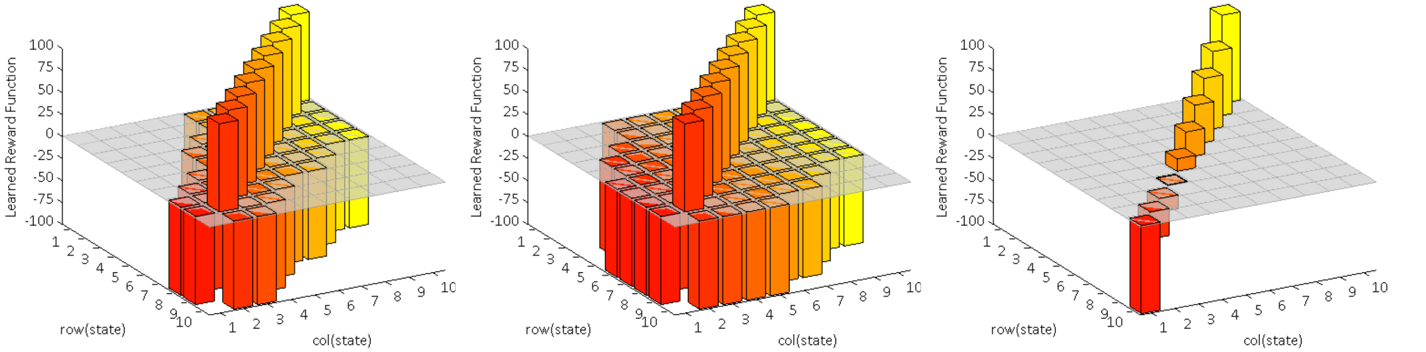

Fig. 2. Learned rewards using **our proposed approach** with various types of advice (introduced in Section III) on a 10×10 pedagogical domain. The start is $(10, 1)$ (bottom left) and goal is $(1, 10)$ (top right). (**left**) With action preferences, which specify that `goNorthEast` is preferable to other actions for states on the anti-diagonal from start to goal; (**center**) With state preferences, which specify for each anti-diagonal state, that neighboring states to the right and above are *less* preferable i.e., avoidable; (**right**) With reward preferences, which specify that rewards for anti-diagonal states nearer the goal should be higher. Our approach was provided with the trajectories of a suboptimal (in fact, random) demonstrator, and expert advice specified above. **Different advice types lead to different learned rewards, which reflect the nature of the given advice.** More specifically, small amounts of targeted advice that specifies preferences over states and actions can help overcome imperfect and noisy demonstrations effectively.

incorporating explicit advice (instruction). To further illustrate that preference advice can be very effective, we assume that the demonstrator acts randomly, that is, the demonstrator's action selection is uniformly random, and highly suboptimal. Without expert advice, in this extreme case of random trajectories, the formulation (6) (which assumes optimal P_{a_*}) can only learn $\mathbf{r} = 0$. Since the reward function is degenerate, the learner is unable to discriminate between states using trajectories alone, and learns a policy which involves it acting randomly in this world. This is not surprising since a randomly-acting demonstrator also does nothing to discriminate between states in their demonstration.

However, now consider that in addition to the demonstrator's trajectories, our learner is also provided with expert advice: preferences over a small set of actions, states, or rewards. We show that such natural advice can lead to rewards that effectively discriminate between states. Furthermore, advice can help overcome difficulties in learning from a demonstrator that is possibly suboptimal, and from significantly fewer trajectories. Figure 2, which previews the approach presented in the succeeding sections, shows the rewards learned from these random trajectories and each type of preference advice on this 10×10 domain.

A. General Form of Advice for IRL

Denote Pref as the set of some expert-specified preferred states *or* actions; similarly, denote Avoid as the set of some expert-specified avoidable states *or* actions, with $\mathsf{Pref} \cap \mathsf{Avoid} = \emptyset$. The general form of advice is as follows:

$$\phi(z; \mathbf{r}) - \psi(z'; \mathbf{r}) \geq \zeta + \delta, \; \forall\, z \in \mathsf{Pref}, z' \in \mathsf{Avoid}. \quad (7)$$

The variables z, z' can either be both states or both actions. The exact nature of this advice depends on ϕ, ψ, which can be the Q-values (for action preferences), state values (for state preferences), or directly, the rewards themselves (for reward preferences). It is evident from (2–2) that state values ($\mathbf{v}^\pi$) and Q-values ($\mathbf{q}^\pi$) can be expressed as functions of the rewards, $\mathbf{r}$. Thus, enforcing the constraints (7) enables the expert to naturally and *qualitatively* express preference/avoidance for specific states/actions. The constraints, through the utility functions, ϕ and ψ, can then *quantitatively* enforce these preferences when incorporated into the IRL framework (6).

The global parameters λ trade-off the influence of preference advice with the trajectories and sparsity-inducing regularization in the LP objective. In addition, in (7), the user can also set *advice-specific* parameters δ, which control the *hardness* of each preference individually. If *we set* $\delta > 0$, this enforces a *hard margin on the preferences*; a larger δ makes the constraint harder to violate. Alternately, if we set $\delta \leq 0$, the constraint becomes *softer advice*, which is easier to satisfy, or even violate. Thus, δ reflects how rigorously the expert prefers the advice to be used, with high values of δ representing greater emphasis on that rule.

In (7), $\zeta \geq 0$ are slack variables that measure the difference between preferred and avoidable states/actions/rewards. This variable is *maximized* in the objective to ensure the best possible discrimination within the learned rewards. The presence of slack variables also enables the formulation to handle conflicting advice, which can be common in larger domains. In our context, conflicting advice refers to situations in which $\mathsf{Pref} \cap \mathsf{Avoid} \neq \emptyset$. When we do have $\mathsf{Pref} \cap \mathsf{Avoid} = \emptyset$, it means that the expert has provided concrete advice and a violation of this condition is an example of noise in advice.

The slack variables in (7) handle this noisy case; for instance, consider in the j-th preference, $\mathsf{Pref}_j \cap \mathsf{Avoid}_j = \{s, s'\}$, and preference advice is given over states i.e., $\phi, \psi \equiv v^\pi$. In the optimal solution, we will have $v^\pi(s) - v^\pi(s') = \zeta_j = 0$, or both states are equally preferable, which is precisely what this advice specifies. If further discrimination is required, we can drop the constraints $\zeta \geq 0$. Note that if a certain piece of advice was designated soft advice by the user ($\delta < 0$), the optimal reward $\mathbf{r}$ *could also satisfy this advice constraint actively* that is, its $\zeta = 0$. This means that it would fail to discriminate between the rewards of the preferred and avoidable sets. The variables ζ function exactly like the slack variables in support vector machines (for example): they relax the hardness of the problem to make it feasible from an optimization perspective, and the extent of the relaxation is controlled by the regularization parameters, λ.

We now detail the three types of preferences that can be naturally provided by a domain expert within this framework.

B. Action Preferences

Action preferences are specified for the j-th state by partitioning the available actions into two groups: preferred actions, $a \in \mathsf{Pref}_j \subseteq \mathcal{A}(j)$ and avoidable actions $a' \in \mathsf{Avoid}_j \subseteq \mathcal{A}(j)$. Here, $\mathcal{A}(j)$ is the set of all available actions at state j. Similar to (5), we can set action preferences by ensuring that the *smallest* Q-value of the preferable actions at state i is better than the *largest* Q-value of the avoidable actions. This is enforced by adding the following constraint to (6)

$$\min_{a \in \mathsf{Pref}_j} q^\pi(j,a) - \max_{a' \in \mathsf{Avoid}_j} q^\pi(j,a') \;\geq\; \zeta_j + \delta_j, \qquad (8)$$

and maximizing the difference in Q-values for this constraint, ζ_i, in the objective. The piecewise-linear constraint (8) can be rewritten as the following set of constraints,

$$q^\pi(j,a) \,-\, q^\pi(j,a') \;\geq\; \zeta_j + \delta_j, \;\; \forall\, a \in \mathsf{Pref}_j,\, a' \in \mathsf{Avoid}_j. \qquad (9)$$

Here, each constraint is between a pair of actions for the state j, one preferred and the second avoidable; the constraints enumerate that each preferred action is better than each non-preferred action exhaustively. This may possibly lead to a combinatorial growth in the number of constraints added into the LP. However, as we are interested in providing *targeted preference advice over a small subset of states and actions*, the number of pieces of advice will typically be small, and the constraints added, very manageable for most standard LP solvers. Also note that the constraint (8) does not give any information about the relative ordering of the preferable actions (or avoidable actions); it only requires that the Q-values of all the preferable actions be at least as good, if not better than the Q-values of the avoidable actions. If a minimum margin (or separation) between the Q-values is desired, this can be enforced by setting $\delta_j > 0$, making this action-preference constraint *harder*. As discussed, previously, the constraint is softened by the slack variables $\zeta_j \geq 0$.

Again setting $B = (I - \gamma P_{a_*})^{-1}$ and using (3), we can write (9) as follows:

$$(P_a^j - P_{a'}^j)\, B\, \mathbf{r} \;\geq\; \zeta_j + \delta_j, \;\; \forall\, a \in \mathsf{Pref}_j,\, a' \in \mathsf{Avoid}_j. \quad (10)$$

In general, the expert can specify action preferences independently for a set of states $\mathcal{S}_e \subseteq \mathcal{S}$. Incorporating these constraints for each $j \in \mathcal{S}_e$ into (6) results in the following linear program:

$$\begin{array}{ll}\max\limits_{\mathbf{r},\xi_i,\zeta_j} & -\|\mathbf{r}\|_1 \,+\, \lambda_t \sum\limits_{i=1}^n \xi_i \,+\, \lambda_a \sum\limits_{j \in \mathcal{S}_e} \zeta_j \\ \text{s. t.} & (P_{a_*}^i - P_a^i)\, B\, \mathbf{r} \;\geq\; \xi_i, \\ & \qquad\qquad\qquad \forall\, a \in \mathcal{A} \setminus a_*,\; i = 1, \ldots, n, \\ & (P_a^j - P_{a'}^j)\, B\, \mathbf{r} \;\geq \zeta_j + \delta_j, \\ & \qquad\qquad \forall\, a \in \mathsf{Pref}_j,\, a' \in \mathsf{Avoid}_j,\, j \in \mathcal{S}_e, \\ & \zeta_j \geq 0, \qquad\qquad\qquad\qquad \forall\, j \in \mathcal{S}_e, \\ & |r_i| \leq r_{\mathsf{max}}, \qquad\qquad\qquad \forall\, i = 1, \ldots, n.\end{array} \qquad (11)$$

Note that we no longer impose $\xi_i \geq 0$ because we are relaxing the reward optimality condition (4) in (6) in order to ensure feasibility if the expert's action preferences possibly contradict the demonstrator's trajectories. The parameters λ_t and λ_a can be set by the user, and emphasize the relative importance of trajectories and action-preference advice respectively, and how they trade-off with each other, as well as regularization. The expert can also set the value of each δ_j and determine the hardness of each advice constraint.

For the grid world example domain described earlier, we specify 9 action preferences: one for each state on the anti-diagonal, excluding the goal $(1, 10)$. That is, for each state $(11 - k, k)_{k=1}^9$, we partition the actions as $\mathsf{Pref}_k = \{\texttt{goNorthEast}\}$, $\mathsf{Avoid}_k = \{\texttt{goNorth}, \texttt{goEast}\}$, preferring that the agent move north-east whenever on an anti-diagonal state. Thus, we only provide 9 pieces of action preference advice, or less than 10% of the states. We set these action preferences to be soft ($\delta_k = 0$). The learner was provided with both the random trajectories, as well as the expert advice. Figure 2 (left) shows that it is possible to learn a reasonable reward function (that reflects the nature of the provided advice accurately) from action-preference advice. More importantly, the approach is able to overcome the suboptimal effects of the random trajectories.

C. State Preferences

State preferences are specified between two subsets of states. For the j-th piece of advice, let $\mathsf{Pref}_j \subseteq \mathcal{S}$ be a subset of states designated as preferable by the expert, and $\mathsf{Avoid}_j \subseteq \mathcal{S}$ be another subset of states designated as avoidable. The expert can specify T such state preferences ($j = 1, \ldots, T$). Similar to (8), we can enforce these state preferences by ensuring that the *smallest* state value of the preferable states in Pref_j is better than the *largest* state value of the avoidable states in Avoid_j. This can be implemented by adding following constraint to the formulation (6)

$$\min_{s \in \mathsf{Pref}_j} v^\pi(s) - \max_{s' \in \mathsf{Avoid}_j} v^\pi(s') \;\geq\; \zeta_j + \delta_j, \qquad (12)$$

and maximizing the difference in state values for this constraint, ζ_i, in the objective. As before, setting $\delta_j > 0$ makes this state preference constraint harder. We denote B_s as the s-th row of $B = (I - \gamma P_{a_*})^{-1}$, corresponding to state s. The LP below takes into account the state preferences specified by the expert. :

$$\begin{array}{ll}\max\limits_{\mathbf{r},\xi_i,\zeta_j} & -\|\mathbf{r}\|_1 \,+\, \lambda_t \sum\limits_{i=1}^n \xi_i \,+\, \lambda_s \sum\limits_{j=1}^T \zeta_j \\ \text{s. t.} & (P_{a_*}^i - P_a^i)\, B\, \mathbf{r} \;\geq\; \xi_i, \\ & \qquad\qquad\qquad \forall\, a \in \mathcal{A} \setminus a_*,\; i = 1, \ldots, n, \\ & (B_s - B_{s'})\, \mathbf{r} \qquad \geq \zeta_j + \delta_j, \\ & \qquad\quad \forall\, s \in \mathsf{Pref}_j,\, s' \in \mathsf{Avoid}_j,\, j = 1, \ldots, T, \\ & \zeta_j \geq 0, \qquad\qquad\qquad\qquad \forall\, j = 1, \ldots, T, \\ & |r_i| \leq r_{\mathsf{max}}, \qquad\qquad\qquad \forall\, i = 1, \ldots, n.\end{array} \qquad (13)$$

The parameter λ_s controls the influence of state preferences and trades-off with trajectories (via λ_t) and regularization. For the example grid world domain, we specified $T = 9$ state preferences, one for each state on the anti-diagonal, excluding

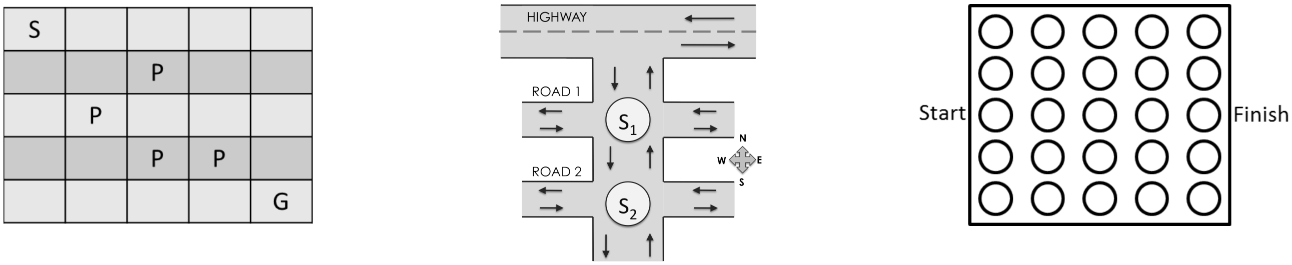

Fig. 3. Experimental Domains: (**left, Wumpus World**) The agent travels from the start (S) to the goal (G), while avoiding the pits (P). If the agent reaches any P instead of G, the current trajectory ends in failure; (**center, Traffic Signals**) Two agent-controlled intersections attempt to regulate traffic, with the signals S_1 near the highway. The state space of each agent is the discretized density of cars, which is higher on the highway, and consequently, at S_1. (**right, Sailing**) The agent navigates a sailboat from the start to finish, passing through the intermediate waypoints, taking into account the direction of the wind

the goal. Anti-diagonal states are preferred over states immediately to the right and above. This ensures that when the agent is in an anti-diagonal state, it will find the next higher anti-diagonal state (closer to the goal) more appealing than any of the surrounding states. Specifically, for $k = 1, \ldots, 9$, $\mathsf{Pref}_k = \{(11-k, k)\}$ and $\mathsf{Avoid}_k = \{(10-k, k),\, (11-k, k+1)\}$. We also set $\delta_k = 0$. Figure 2 (center) shows that state preferences can learn a more discriminative reward function than action preferences *for this domain*. The key takeaway is that we learn a reasonable reward function with accurate expert advice given over only a very small subset (10%) of the states.

D. Reward Preferences

For reward preferences, rather than specifying constraints over the state values, we specify constraints over the *immediate* rewards of these states directly. This advice expresses a direct preference for immediate rewards, $r(s)$, rather than long-term accumulated rewards as measured by $v^{\pi}(s)$. Similar to the state-preference advice case, let $\mathsf{Pref}_j \subseteq \mathcal{S}$ be a subset of states designated as preferable by the expert, and $\mathsf{Avoid}_j \subseteq \mathcal{S}$ be another subset of states designated as avoidable. We would like that learned rewards for states in Pref_j be higher than rewards for states in Avoid_j. This can be achieved by directly specifying constraints on the rewards:

$$\min_{s \in \mathsf{Pref}_j} r(s) - \max_{s' \in \mathsf{Avoid}_j} r(s') \geq \zeta_j + \delta_j. \tag{14}$$

Naturally, the expert can choose to provide T reward preferences. Again, these constraints can be incorporated into the formulation (6):

$$\begin{array}{ll}
\max\limits_{\mathbf{r}, \xi_i, \zeta_j} & -\|\mathbf{r}\|_1 + \lambda_t \sum\limits_{i=1}^{n} \xi_i + \lambda_r \sum\limits_{j=1}^{T} \zeta_j \\
\text{s. t.} & (P^i_{a_*} - P^i_a)\, B\, \mathbf{r} \geq \xi_i, \\
& \qquad\qquad\qquad \forall\, a \in \mathcal{A} \setminus a_*,\ i = 1, \ldots, n, \\
& r(s) - r(s') \geq \zeta_j + \delta_j, \\
& \qquad\quad \forall\, s \in \mathsf{Pref}_j,\, s' \in \mathsf{Avoid}_j,\, j = 1, \ldots, T, \\
& \zeta_j \geq 0, \qquad\qquad\qquad\quad \forall\, j = 1, \ldots, T, \\
& |r_i| \leq r_{\mathsf{max}}, \qquad\qquad\quad\ \forall\, i = 1, \ldots, n.
\end{array} \tag{15}$$

The reward-preference advice essentially enables the domain expert to specify a partial ranking ordering of states directly. The parameter λ_r controls the influence of reward preferences with respect to the trajectories and regularization. In the grid world domain, we enforce very simple reward preferences on the anti-diagonal states: $r(10,1) \leq r(9,2) \leq \ldots \leq r(1,10)$. We also set $\delta = 0.15 r_{\mathsf{max}}$, which requires that each of the states have an enforced separation of $0.15 r_{r\mathsf{max}}$. This makes these reward-preference constraints hard. Note here that the value of r_{max} should be set with care in order to *maintain feasibility of the LP*.

Figure 2 (right) shows the learned reward function under these settings. Note that, in the context of this example, the learned $\mathbf{r}$ is optimal to the LP, but sub-optimal from an RL perspective, that is, there exist reward functions that lead to better agent behavior. To see this, consider the states with negative learned rewards; an agent at this state will move off the anti-diagonal and take longer to eventually reach the goal state. While we could specify reward advice similar to the state advice, to train an optimally-acting agent, we do not do so as our intent is to demonstrate that we can effectively incorporate the expert's reward preferences. The learned rewards, in this example, reflect the expert advice very accurately. The start state has the lowest reward ($r(10,1) = -r_{\mathsf{max}}$), and the rewards on the anti-diagonal monotonically increase by the hardness margin δ, and the goal state has the highest reward ($r(1,10) = r_{\mathsf{max}}$).

IV. Experiments

We aim to answer the following questions through our experiments:

Q1: How does combining trajectories with each advice type compare with learning using only trajectories?
Q2: How do the behaviors of learners given these different forms of advice compare against each other?
Q3: How sensitive is the method to noisy trajectories?
Q4: How sensitive is the approach to noisy advice?
Q5: In which situations does advice help most?
Q6: How does the choice of parameters λ affect the learned reward functions?

We conducted experiments in four computer-simulated domains (Table I) and in each, the rewards were learned from noisy trajectories, as well as domain-specific advice given as action, state, and reward preferences. The LPs were solved using the publicly-available `GLPK`[1] solver. We use $\delta = 0$ for all experiments, and $\lambda_t = 10^3$ when learning from trajectories only. After the rewards were learned, value iteration [1] was

[1] `http://www.gnu.org/software/glpk`

performed, following which the agents acted greedily in their respective domains.

A. Domains and Experimental Setup

The performance of each advice type (with trajectories) was compared to a no-advice agent that learned a reward function using only the trajectories. The IRL problems were solved for increasing number of trajectories, and their performance was averaged over 20 simulations for Wumpus World and Sailing, and 10 simulations for the Traffic Signals and Driving. We compute P_a for Traffic Signals by simulating a random demonstrator in these domains, and applying a Laplacian correction to the transition function.

TABLE I. EXPERIMENTAL DOMAINS. $|\mathcal{S}|$ AND $|\mathcal{A}|$ ARE NUMBER OF STATES AND ACTIONS. $|AA|$, $|SA|$ AND $|RA|$ REFER TO NUMBER OF ACTION, STATE AND REWARD ADVICE PIECES RESPECTIVELY.

DOMAIN	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert AA\rvert$	$\lvert SA\rvert$	$\lvert RA\rvert$
`Wumpus World`	25	4	8	2	1
`Sailing`	51	4	23	3	3
`Traffic Signal`	256	16	45	2	-
`Driving`	400	3	37	2	2

Table I describes the domain sizes and the amount of advice provided. Each action advice piece represents action preferences for *one distinct* state, with the sets Pref and Avoid being small subsets of action space. In contrast, for state and reward advice, we specify subsets of preferable and avoidable states, and each Pref and Avoid pair is counted as a single piece of advice. This masks the fact that, for state and reward preferences, the sizes of Pref and Avoid can be large. We revisit this issue in the Traffic domain.

B. Wumpus World

This is a modification of a classical RL domain [22]; our Wumpus World consists of a 5×5 grid with 4 obstacles (pits); the agent must navigate from the start to the goal while avoiding the obstacles. The agent can execute 4 actions: move in each of the cardinal directions. If the agent enters a pit space, it dies. Action advice is specified for states adjacent to the pits: actions that take the agent into the pit are avoidable. The state (and respectively, reward) advice specifies that the goal should have the highest value (reward), and that the pit squares should have lower value (reward) than the other states. The regularization parameters were set to $\lambda_t = 1$ and $\lambda_a = \lambda_s = \lambda_r = 10$.

Performance is measured by the number of times (%) the agent following the greedy policy (specified by the learned rewards) is able to reach the goal. Figure 4 (top left) shows the behavior of agents under these different settings. When the agent learns rewards using trajectories only, its behavior is erratic, and it never achieves better than a 50% goal rate. In contrast, agents that were given advice are able to learn reasonably close to optimal behaviors, especially as the number of trajectories increases. Specifying state preferences produces a very strong performance, and the agent is able to learn to act optimally with little demonstration. This effect can also be observed in knowledge-based support vector machines [18], [19], which are capable of learning reasonable classifiers using advice and very little data, as long as the advice provided by the expert was reasonable.

C. Traffic Signals

This domain is an adaptation of the traffic simulator from Natarajan et al. [13]. It models *two* traffic intersections, each with four actions corresponding to the direction that have a green signal. Hence, the action space of the signals is to allow traffic in one direction to go straight, right or left in the signal. The two signals do not have uniformly identical behavior; this is because of the presence of a highway near the signal S_1, which means that it will have to deal with larger volumes of traffic. The state space of each signal is the (discretized) density of cars, $\{\texttt{low}, \texttt{high}\}$.

Action advice specifies that at S_1, traffic from the highway gets priority compared to the traffic from other directions. State advice prefers states in which there is only one direction at `high` for each signal. In addition, states that have two directions at `high` are preferred over all other states, *except* those that have only one direction at `high`. For state advice, while the set of preferable states is small, the set of avoidable states is much larger; this creates a large number of constraints in the LP, owing to the combinatorial nature of the advice (12). To overcome this issue, we *uniformly sample* avoidable states to maintain a computationally feasible LP. This is in keeping with our general advice-giving philosophy of providing small amounts of accurate advice. We set $\lambda_t = 10^2$ and $\lambda_a = 10^3$ for action advice, and $\lambda_t = 10$ and $\lambda_s = 10^3$ for state advice.

Performance is measured by the total wait time for cars (as measured in domain time steps) before getting a green, with lower times indicating better performance. In Figure 4 (top right), advice-taking agents outperform the no-advice agent. The performance of agents with state advice was more effective than with action advice. This shows that, *even when sampling preferences*, it is possible to learn effective reward functions with fewer trajectories.

D. Sailing

This domain is a modification of the one proposed by Vanderbei[2], in which, given a grid of waypoints connected by legs, the agent navigates a sailboat from one way point to the next to reach the finish in the shortest time possible. The key difference from the original domain is that the wind makes the result of the actions stochastic and the agent must take wind direction into account to choose one of 4 actions. The lake has distinct boundaries and if the agent sails out of these, it crashes into the shore. The action advice specifies action preferences around the edges of the lake to avoid this outcome. It also specifies actions that lead to selecting the shortest path across the lake, assuming that the wind has no effect on the action outcomes. The state/reward advice specifies that the middle of the lake should be preferred over the edges, and the finish state should be most preferred. For this domain, we used $\lambda_t = 10$, $\lambda_a = \lambda_s = \lambda_r = 10^2$.

In this domain (Figure 4, bottom left), performance is measured by how often (%) the greedy agent reached the finish. Agents with each of the three advice types significantly outperform the no advice agent; the latter does eventually

[2] Sailing Strategies: An Application Involving Stochastics, Optimization, and Statistics (SOS); `http://orfe.princeton.edu/∼rvdb/sail/sail.html`

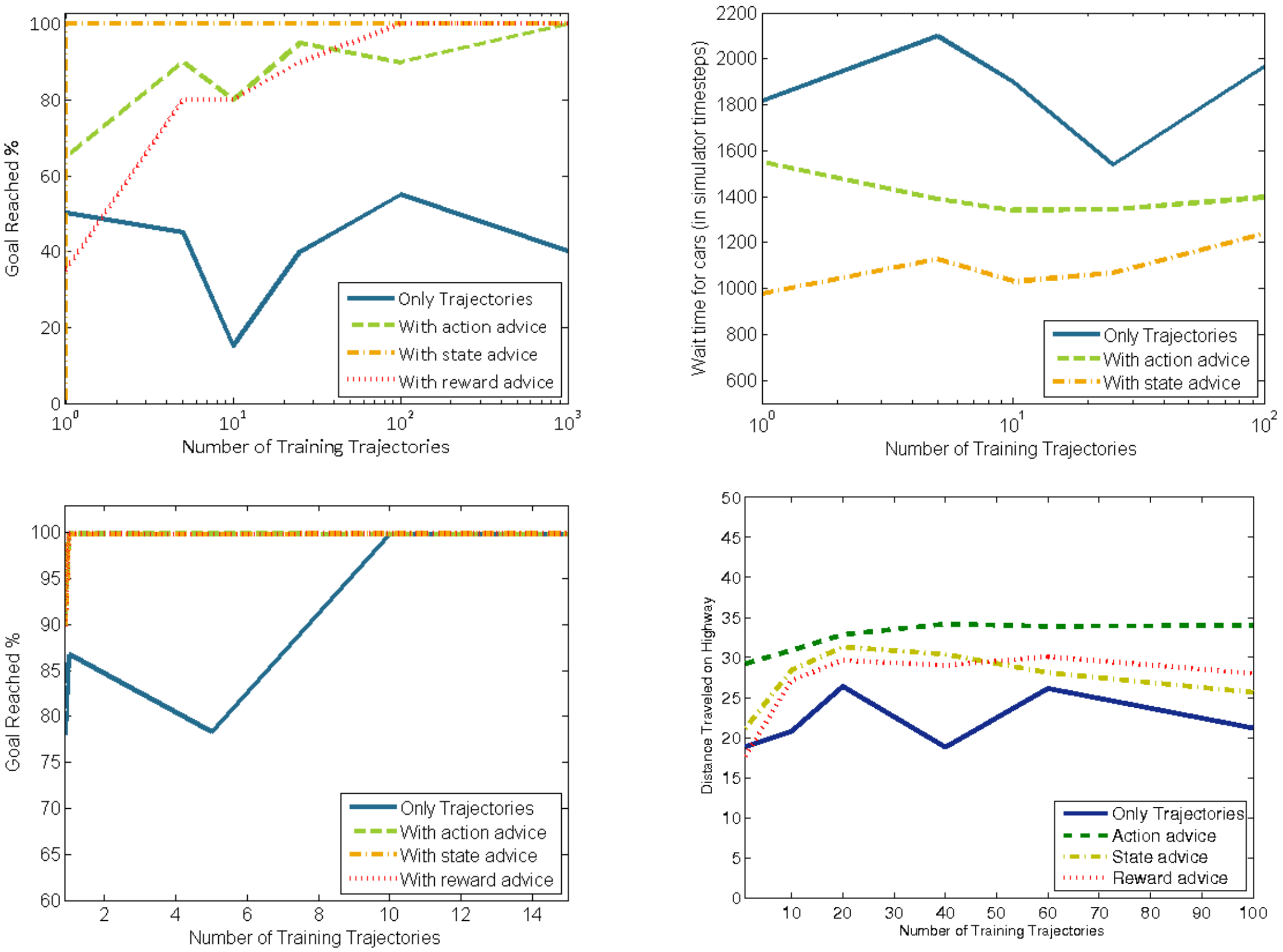

Fig. 4. Results comparing agents that learned from demonstration only, to those that learned from demonstration *and* preference advice. Performance is measured for (**top left, Wumpus World; bottom left, Sailing**) by how often (%) the agent reached the goal; (**top right, Traffic Signals**) by wait time at a signal, with *lower waiting times being better*; (**bottom right, Driving**) by distance driven on the highway, with *higher distances being better*. Various preference advice types uniformly improve performance over all domains.

learn optimal behavior, but requires more trajectories to do so, highlighting the benefit of giving small amounts of targeted advice to the learners.

E. Driving

This domain is a modification of the Driving simulator used by Abbeel and Ng [10], in which an agent must navigate a car on the highway. The agent's speed is constant and faster than all other cars on the highway, and therefore the agent must change lanes as it drives in order to avoid the other cars. The agent can occupy one of four lanes (right/left, off road right/left), and can see only the closest car in each lane (10 possible values). Cars in the right lane appear more often than in the left lane, and drive at slower speed. The agent must drive as far as possible on the highway while avoiding crashing into other cars, and driving off the road as much as possible. The advice specifies that it is generally better to be in the left lane as cars will be slower in the right lane. State and reward advice prefer the agent to come onto the highway if off-road, and the adjacent highway lane is free. We used $\lambda_t = 10^2$, $\lambda_a = \lambda_s = \lambda_r = 10$. In Figure 4 (bottom right) we see that advice-taking agents outperform the no-advice agent. More specifically, action preferences are more appropriate for this domain.

F. Noise-Free Trajectories and Advice

Now, **Q1** can be answered affirmatively: in all domains, the use of advice greatly helps in learning better rewards than only using trajectories. In three of four domains, giving state preferences, rather than action preferences is more useful, and results in learning better rewards. This is especially true when the number of available trajectories is very small. In such situations, it is certainly easier to specify state preferences since the amount of advice required is much smaller than specifying action preferences. Thus, in answer to **Q2**, state advice seems more natural and useful from our empirical evaluations, particularly when learning with a very small number of demostrations.

G. Noisy Trajectories and Advice

To answer **Q3**, **Q4** and **Q5**, we performed additional experiments in two domains – Wumpus World and Driving. First,

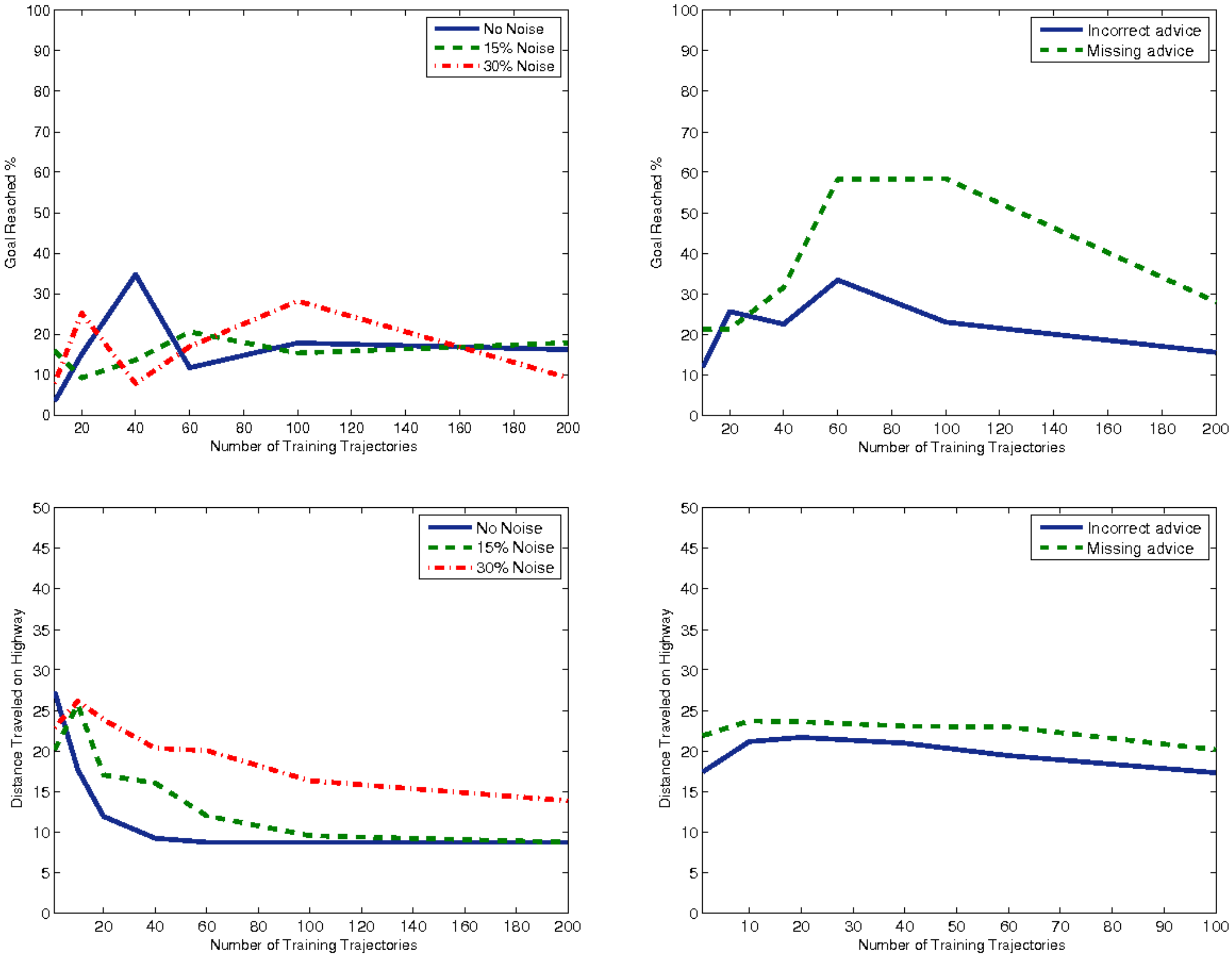

Fig. 5. Results showing the effect of noise in trajectories and advice on reward learning. We show the performance of agents that learned with (**left**) noisy trajectories and no advice and (**right**) optimal trajectories and noisy advice in two domains: Wumpus World (**top**) and Driving (**bottom**). For noisy trajectories and no advice (left) even a small amount of noise in the trajectories leads to inferior rewards being learned, and poorer domain behavior. This is because of the demonstrator optimality assumption in standard IRL [5]. Alternately, our approach is more robust to noise in advice (right), especially to missing advice.

we analyze Ng and Russell's original IRL formulation to see how well it handles noisy trajectories. The noisy trajectories in both domains are generated by choosing an action randomly (0%, 15%, and 30% of the time). For *noisy trajectories and no advice* (left column of Figure 5), performance is not high, even with a small amount of noise in the trajectories (cf. Figure 4). This answers **Q3**: standard IRL without advice does not handle noisy trajectories well, compared to with-advice cases. A significant reason for this is the original IRL assumption of demonstrator optimality; in the original formulation, the demonstrations are assumed to be the actions of an agent acting perfectly optimally in the given domain. We relax this assumption to varying degrees by adding randomness in the trajectories, and the resulting performance degenerates. This experiment also provides a glimpse of the answer to **Q5**: that advice is particularly useful for noisy trajectories.

The second column of Figure 5 shows performance with noisy *state* advice and optimal trajectories. Advice can be noisy due to two reasons: *missing* advice where some preferences are left out, and *incorrect* advice where some preferences are perturbed. For both Wumpus World and Driving, we dropped/perturbed 10% of states in the preferred or non-preferred sets. In Figure 5, we see that our approach is reasonably robust to missing advice as it can recover some of this advice from the (optimal) trajectories. When advice is incorrect, the method suffers more. To answer **Q4**, missing advice can be handled more effectively than incorrect advice in both domains. Overall, in many real-life situations, we can expect noise in both trajectories and advice, and our approach can incorporate both robustly to learn good reward functions.

In order to answer **Q5** explicitly, we demonstrate the importance of advice in a common real-life situation: unvisited states in domains. In many domains, when there are highly avoidable states or actions (i.e., those that lead to catastrophic consequences for the agent), the demonstrator will simply avoid those states and actions without providing an explanation

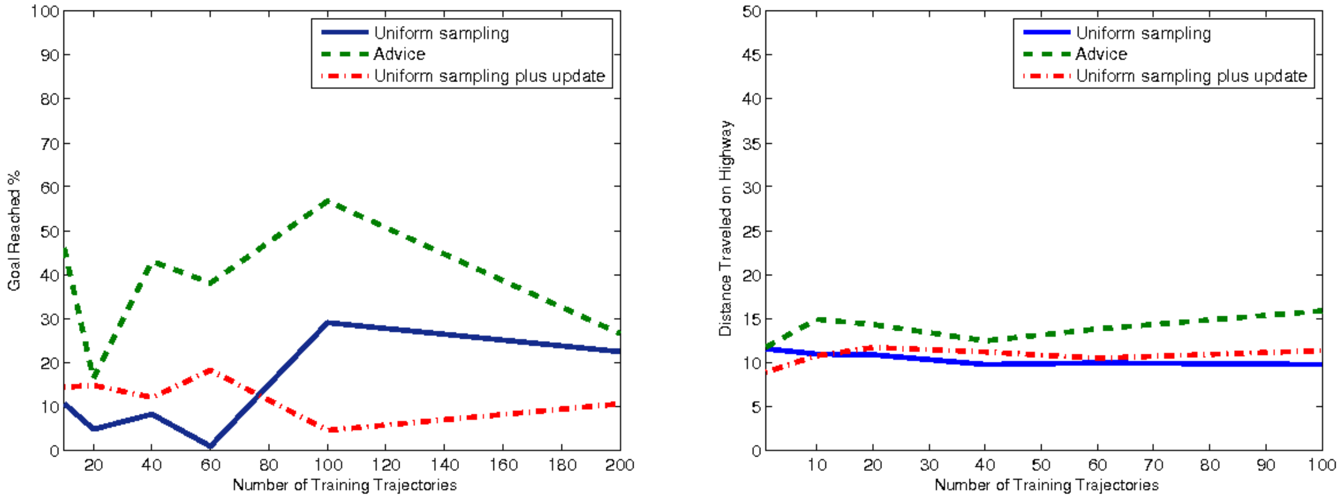

Fig. 6. Results showing the behavior three different agents in (**left**) Wumpus World, and (**right**) Driving, when certain states are not seen in the demonstrator trajectories. The first agent uniformly samples the actions space at a given state, when if it finds itself in a state not visited previously by the demonstrator. The second agent picks a random action uniformly once, and then its policy is skewed towards this random policy in future visits to the unseen state. The third agent, unlike the first two, has access to a domain expert's action-preference advice at the unseen states, which it incorporates in reward learning, and consequently in learning to act.

to the learner. It is very difficult for a learner to then reasonably infer the avoidability of such states. For these unseen states, it is far easier to specify the states and actions to be avoided through advice, rather than demonstrate the negative consequences. This is the scenario we consider in this experiment: we explicitly provide action advice at unseen states, i.e., for states that are never visited during the demonstration (shown as *advice* in Figure 6.

We compare this advice to two agents which have to select appropriate actions from the same demonstrations, but without advice. The first samples actions from a *uniform distribution*, and the second samples one action randomly, and then updates the demonstrator policy so that the distribution is skewed towards this action that is, we *simulate* an expert such that they are more likely to choose from this skewed distribution. We call this agent *uniform sampling plus update*. Put another way, the first agent employs a *uniform distribution* over the policy of unseen states, and the second agent skews the policy towards a *random policy*.

The performance of all three agents are shown in Figure 6. Giving advice for unseen states significantly improves learned behaviors, answering **Q5**; advice is most useful in unvisited or suboptimal states as observed from the demonstrator policy. This is another key reason why advice can be crucial – to avoid risk states [23] in unseen trajectories that are otherwise unavoidable without any advice.

H. The Effect of Parameters λ_t, λ_a

Finally, to answer **Q6**, we investigate the effect of regularization parameters on learned rewards, and the consequent behavior of the agents. We return to the 10×10 pedagogical grid world, in which an agent moving from the start $(10,\ 1)$ to goal $(1, 10)$ optimally can do so in a trajectory of length 9. In this experiment, we consider a demonstrator that provides trajectories with 30% noise and action advice, as specified previously. Rewards were learned for uniformly sampled values of λ_t, $\lambda_a \in [2^{-5}, 2^{5}]$. Figure 7 summarizes these results.

We found that, except certain "poor values" of λ_t and λ_a, learned rewards (and resulting agent behavior) are similar, for "reasonable values" of these parameters. Poor parameter values are those that result in degenerate or non-informative rewards such as $\mathbf{r} = \mathbf{0}$ or $\mathbf{r} = \pm r_{\mathsf{max}}\mathbf{1}$. These solutions typically arise when λ values are very small, and are poor because they are unable to discriminate between various states. For reasonable values, usually in the range: $\lambda \in [2^{-1},\ 2^{3}]$, the *quality of behavior induced by the learned rewards did not change dramatically*. Similar behavior was observed in the other domains as well.

For these experiments, note that we set $\delta = 0$. Degenerate solutions may indicate that the constraints in the problem are not restrictive enough. In such situations, it would be helpful to set a value of $\delta > 0$, in order to harden the constraints and force discrimination between the learned rewards. However, these observations are very preliminary, and the subtle interaction between various parameters, including the hardness δ deserves deeper study, which is beyond the scope of this work.

V. Conclusions and Future Work

We propose a novel methodology for incorporating expert advice into the inverse reinforcement learning framework. This, our key contribution, arises from the relaxation of the assumption of demonstrator optimality, which is common in most IRL approaches to date. Our approach provides a framework within which preferences over states and actions, specified a non-AI domain expert can be incorporated into the IRL problem. Our approach is able to combine such natural advice with demonstrator trajectories to learn rewards, which can then be used to determine how to act in the domain. This approach enables learning in situations where the agent

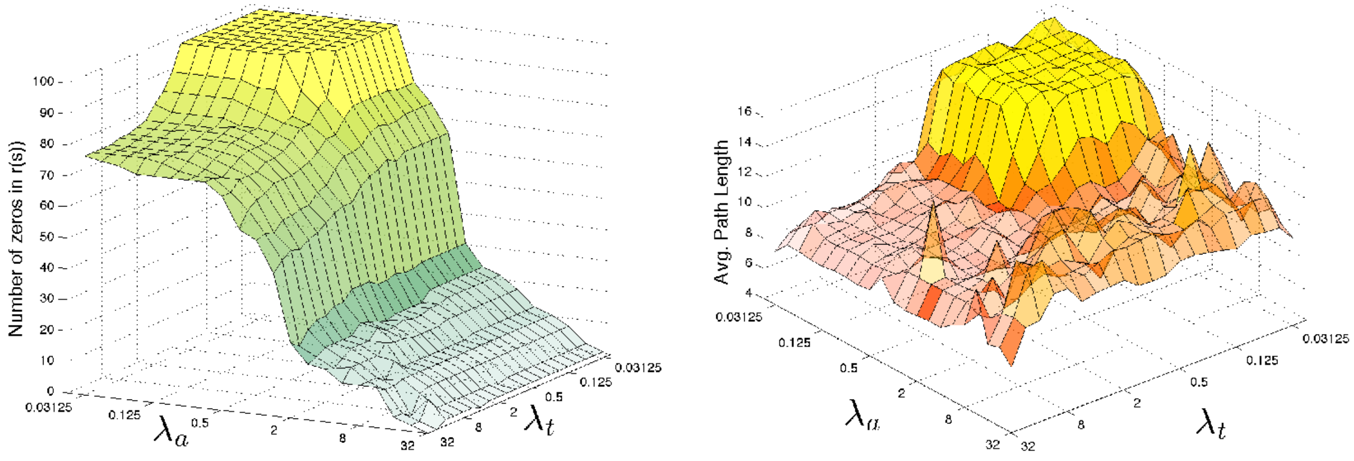

Fig. 7. Results showing the effect of the parameters λ_t and λ_a on (**left**) sparsity of the reward $\mathbf{r}$, and (**right**) quality of the learned rewards, when acting in the 10×10 grid domain (Figure 2), under a greedy policy as measured by the average path length to goal. Rewards that are too sparse or too dense are not very useful in discriminating states. Such degenerate or non-discriminative rewards are learned for "poor" choices of the regularization parameters, that is, very small values of λ. The behaviors of agents that learn to act using rewards generated with "reasonable" λ values are very similar.

observes a possibly noisy and suboptimal demonstrator, but can use expert advice (which is also possibly noisy) to learn good behavior. Our experiments show that it is possible to learn to act in such situations, and that advice can help learn with fewer trajectories. Furthermore, the incorporation of ℓ_1-regularization allows us to learn sparse and discriminative reward functions. These results serve as a proof-of-concept, and we propose to further investigate advice giving for more complex domains: in continuous, possibly infinite-dimensional state-action spaces.

Acknowledgements

SN and PO thank Army Research Office grant number W911NF-13-1-0432 under the Young Investigator Program. SN and JS gratefully acknowledge the support of the DARPA DEFT Program under the Air Force Research Laboratory (AFRL) prime contract no. FA8750-13-2-0039. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, or the US government.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Cambridge, MA: The MIT Press, 1998.

[2] A. Segre and G. DeJong, "Explanation-based manipulator learning: Acquisition of planning ability through observation," in *IEEE Conf. on Robotics and Automation*, 1985.

[3] R. Khardon, "Learning action strategies for planning domains," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 125–148, 1999.

[4] H. Lieberman, "Programming by example: Introduction," *Communications of the ACM*, vol. 43, no. 3, pp. 72–74, 2000.

[5] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000, pp. 663–670.

[6] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Imitation in animals and artifacts," K. Dautenhahn and C. L. Nehaniv, Eds., 2002, ch. Learning to fly, pp. 171–189.

[7] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, no. 1, 2009.

[8] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[9] S. Calinon, *Robot Programming by Demonstration - a Probabilistic Approach*. EPFL Press, 2009.

[10] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004.

[11] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *IJCAI*, 2007.

[12] G. Neu and C. Szepesvári, "Training parsers by inverse reinforcement learning," *Machine Learning*, vol. 77, no. 2-3, pp. 303–337, 2009.

[13] S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting, and J. Shavlik, "Multi-agent inverse reinforcement learning," in *ICMLA*, 2010, pp. 395–400.

[14] J. Choi and K.-E. Kim, "Inverse reinforcement learning in partially observable environments," *JMLR*, vol. 12, pp. 691–730, 2011.

[15] C. Boutilier, "A POMDP formulation of preference elicitation problems," in *AAAI*, 2002, pp. 239–246.

[16] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *EWRL*, 2011.

[17] C. Rothkopf and C. Dimitrakakis, "Preference elicitation and inverse reinforcement learning," in *ECML-PKDD*, 2011.

[18] G. Fung, O. L. Mangasarian, and J. W. Shavlik, "Knowledge-Based support vector machine classifiers," in *NIPS*, 2002, pp. 01–09.

[19] G. Kunapuli, K. P. Bennett, A. Shabbeer, R. Maclin, and J. W. Shavlik, "Online knowledge-based support vector machines," in *ECML*, 2010, pp. 145–161.

[20] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild, "Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression," in *AAAI*, 2005.

[21] N. A. Hodges and I. M. Franks, "Modelling coaching practice: the role of instruction and demonstration," *Journ. Sports Sci.*, vol. 20, pp. 793–811, 2002.

[22] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, 3rd ed. Pearson Education, 2010.

[23] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *JAIR*, vol. 24, no. 1, pp. 81–108, 2005.