# Advice-Based Exploration in Model-Based Reinforcement Learning

Rodrigo Toro Icarte[1,2], Toryn Q. Klassen[1],
Richard Valenzano[1,3], and Sheila A. McIlraith[1]

[1] University of Toronto, Toronto, Canada
{rntoro,toryn,rvalenzano,sheila}@cs.toronto.edu
[2] Vector Institute, Toronto, Canada
[3] Element AI, Toronto, Canada

**Abstract.** Convergence to an optimal policy using model-based reinforcement learning can require significant exploration of the environment. In some settings such exploration is costly or even impossible, such as in cases where simulators are not available, or where there are prohibitively large state spaces. In this paper we examine the use of advice to guide the search for an optimal policy. To this end we propose a rich language for providing advice to a reinforcement learning agent. Unlike constraints which potentially eliminate optimal policies, advice offers guidance for the exploration, while preserving the guarantee of convergence to an optimal policy. Experimental results on deterministic grid worlds demonstrate the potential for good advice to reduce the amount of exploration required to learn a satisficing or optimal policy, while maintaining robustness in the face of incomplete or misleading advice.

**Keywords:** Markov Decision Process, Reinforcement Learning, Model-Based Learning, Linear Temporal Logic, Advice

## 1 Introduction

Reinforcement Learning (RL) methods can often be used to build intelligent agents that learn how to maximize long-term cumulative reward through interaction with the environment. Doing so generally requires extensive exploration of the environment, which can be infeasible in real-world environments where exploration can be unsafe or require costly resources. Even when there is access to a simulator and exploration is safe, the amount of interaction needed to find a reasonable policy may be prohibitively expensive.

In this paper we investigate the use of *advice* as a means of guiding exploration. Indeed, when humans try to master a new task, they certainly learn through exploration, but they also avail themselves of linguistically expressed advice from other humans. Here we take “advice” to be recommendations regarding behaviour that may describe suboptimal ways of doing things, may not be universally applicable, or may even contain errors. However, even in these cases people often extract value and we aim to have RL agents do likewise. We use advice to guide the exploration of an RL agent during its learning process so that it more quickly finds useful ways of acting.

We use the language of Linear Temporal Logic (LTL) [1] for providing advice. The advice vocabulary is drawn from state features, together with the linguistically inspired temporal modalities of LTL (e.g., *"Turn out the lights before you leave the office"* or *"Always avoid potholes in the road"*). We propose a variant of the standard model-based RL algorithm *R-MAX* [2] that adds the ability to use given advice to guide exploration. Experimental results on randomly generated (deterministic) grid worlds demonstrate that our approach can effectively use good advice to reduce the number of training steps needed to learn a strong policy, and also recover from misleading advice.

## 2 Preliminaries

**Example environment:** Consider a grid world in which the agent starts at some initial location. At various locations there are doors, keys, walls, and nails. The agent can move deterministically in the four cardinal directions, unless there is a wall or locked door in the way. The agent can only enter a location with a door when it has a key, after which point the door and key disappear (i.e., the door remains open). The agent automatically picks up a key whenever it visits a location with a key. The agent receives a reward of -1 for every action, unless it enters a location with nails (reward of -10) or reaches the red door with a key (reward of +1000, and the episode ends). Figure 1, which we use as a running example below, depicts an instance of this domain, in which there is a single door and it is red.

In this environment, we may wish to advise the agent to avoid the nails, or to get the key before going to the door. In order to provide advice to an arbitrary agent, we must have a vocabulary from which the advice is constructed. For this purpose, we define a *signature* as a tuple $\Sigma = \langle \Omega, C, \text{arity} \rangle$ where $\Omega$ is a finite set of predicate symbols, $C$ is a finite set of constant symbols, and $\text{arity} : \Omega \to \mathbb{N}$ assigns an arity to each predicate. For example, in the grid-world environment, we use a signature with only a single predicate called $\texttt{at}$ (i.e., $\Omega = \{\texttt{at}\}$ and $\text{arity}(\texttt{at}) = 1$), where $\texttt{at}(c)$ states that the agent is at the same location as $c$. Each object in the domain will be represented with a single constant in $C$ (i.e., key1, door1, ...). Intuitively, we use this signature to reference different elements about states in the environment when providing advice.

We define $GA(\Sigma) \overset{\text{def}}{=} \{P(c_1, \cdots, c_{\text{arity}(P)}) \mid P \in \Omega, c_i \in C\}$. That is, $GA(\Sigma)$ is the set of all *ground atoms* of the first order language with signature $\Sigma$. A *ground literal* is either a ground atom or the negation of a ground atom, so $\mathsf{lit}(\Sigma) \overset{\text{def}}{=} GA(\Sigma) \cup \{\neg p : p \in GA(\Sigma)\}$ is the set of ground literals. A *truth assignment* can be given by a set $\tau \subseteq \mathsf{lit}(\Sigma)$ such that for every $a \in GA(\Sigma)$, exactly one of $a$ and $\neg a$ is in $\tau$. Let $T(\Sigma)$ be the set of all truth assignments.

A *Markov Decision Process (MDP)* with an initial state and a signature is a tuple $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L \rangle$ where $S$ is a finite set of *states*, $s_0 \in S$ is the initial state, $A$ is a finite set of *actions*, $p$ is a function that specifies *transition probabilities* where $p(s', r|s, a)$ is the probability of transitioning to $s'$ and receiving reward $r \in \mathbb{R}$ if action $a \in A$ is taken in state $s$, $\gamma \in (0, 1]$ is the *discount factor*, $\Sigma = \langle \Omega, C, \text{arity} \rangle$ is a signature, and $L : S \to T(\Sigma)$ labels each state with a truth assignment. For example, in our grid-world, the labeling function $L(s)$ makes $\texttt{at}(c)$ true if and only if the location of the agent is equal to the location of $c$ in state $s$. Note that as $GA(\Sigma)$ is finite, we could equivalently consider a state label to be a vector of *binary features*, with the $i$th entry being 1 if the $i$th ground atom holds in that state, and 0 otherwise. Below, we assume that the agent does not know the transition probability function $p$ (as usual in RL).

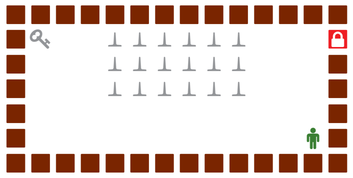
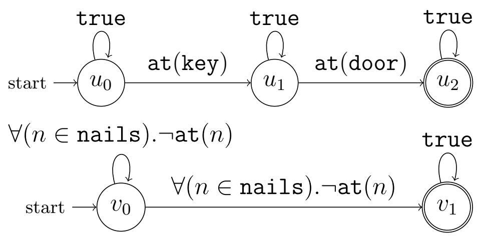


**Fig. 1.** The agent receives reward for going to the locked red door after having visited the key. It is penalized for stepping on nails.



true
true
true
start
$u_0$
$\texttt{at(key)}$
$u_1$
$\texttt{at(door)}$
$u_2$
$\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$
true
start
$v_0$
$\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$
$v_1$

**Fig. 2.** Advice example, with NFAs corresponding to the LTL formula $\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond\texttt{at(door)}) \wedge \Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$

## 3 Providing and Utilizing Advice While Learning

In this section, we describe our LTL advice language and how corresponding automata can be used to monitor satisfaction of such advice. We then describe our method for using the automata to guide exploration in a variant of R-MAX.

### 3.1 Linear Temporal Logic: A Language for Providing Advice

Providing advice to an agent requires a language for communicating that advice. Given that RL agents are typically engaged in an extended interaction with the environment, this language must allow us to suggest how the agent should behave over time (e.g. *"Get the key and then go to the locked door."*). To this end, we use Linear Temporal Logic (LTL), a modal temporal logic originally proposed for the verification of reactive systems [1], that has subsequently been used to represent temporally extended goals and preferences in planning [3]. Here, we use it as the basis for expressing advice.

Suppose that we have an MDP with a signature $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L \rangle$ for which we wish to provide advice. The language of LTL contains formulae consisting of propositional symbols, which we take to be the ground atoms of

$GA(\Sigma)$ (e.g. $\texttt{at}(\texttt{key2})$), and all formulae that can be constructed from other formulae using the standard set of connectives from propositional logic — namely *and* ($\wedge$), *or* ($\vee$), and *negation* ($\neg$) — and the temporal operators *next* ($\bigcirc$) and *until* ($\mathsf{U}$). From these temporal operators, we can also derive other useful operators such as *always* ($\square$) and *eventually* ($\lozenge$). We will also make use of *universal* ($\forall$) and *existential* ($\exists$) quantifiers in abbreviating conjunctions and disjunctions. If $T = \{t_1, \ldots, t_k\} \subseteq C$ is a set of constant symbols, then $\forall(x \in T).\varphi(x) \overset{\text{def}}{=} \varphi(t_1) \wedge \cdots \wedge \varphi(t_k)$ and $\exists(x \in T).\varphi(x) \overset{\text{def}}{=} \varphi(t_1) \vee \cdots \vee \varphi(t_k)$.

To provide some intuition about LTL, we consider some possible example advice formulae for the problem in Figure 1 which use the unary operators $\square$, $\lozenge$, and $\bigcirc$. The formula $\square\neg\texttt{at}(\text{nail1})$ literally means "always, it is not the case that the agent is at the location of nail1"; used as advice it can be taken to say that "at all times the agent *should* not be at nail1." The formula $\lozenge(\texttt{at}(\text{key}) \wedge \bigcirc(\lozenge\texttt{at}(\text{door})))$ can be understood as "the agent should eventually get to a state where it is at the key and then eventually get to a state where it is at the door." If $\texttt{nails} \subset C$ is the set of objects that are nails, we can use $\square\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$ to advise that at all times, the agent should not be at a location where there is a nail.

The truth value of an LTL formula $\varphi$ is determined relative to a sequence $\sigma = \langle s_0, \ldots, s_n\rangle$ of states from $\mathcal{M}$ (i.e., the states visited in an episode). The truth of a ground atom at time $t$ is determined by the label of $s_t$, and the truth values of more complicated formulae are built up according to the formal semantics of LTL (see De Giacomo et al. [4] for more details).

## 3.2 From LTL to Finite State Automata

Any LTL formula $\varphi$ can be converted into a *Nondeterministic Finite State Automaton (NFA)* such that a finite sequence of states $\sigma$ will be accepted by the NFA if and only if $\sigma$ satisfies $\varphi$ [4,5]. We can represent the NFA as a directed graph with edges labelled by formulae from $\mathcal{L}_\Sigma$, the subset of LTL formulae that does not include temporal operators. Each edge represents a set of NFA transitions, one for each truth assignment satisfying the edge label. Because the NFA is non-deterministic, it may be in multiple states at once. Intuitively, the state(s) that an NFA is in after visiting a sequence of MDP states represent the progress that has been made towards satisfying $\varphi$. Reaching an accepting NFA state means that the current trace satisfies the advice formula.

To translate from LTL to automata we use the system developed by Baier and McIlraith [5], which, for computational efficiency, constructs a set $\mathcal{N}$ of small NFAs rather than one potentially very large NFA. $\mathcal{N}$ is considered to accept $\sigma$ if every NFA in $\mathcal{N}$ accepts $\sigma$. For example, Figure 2 shows the two NFAs generated from $\lozenge(\texttt{at}(\texttt{key}) \wedge \bigcirc\lozenge\texttt{at}(\texttt{door})) \wedge \square\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$. This advice states that the agent should get the key and then go to the door, and also avoid nails. We now demonstrate how we track the NFA state set for the top NFA. At the beginning of an episode, the agent is in the initial state of the MDP and the NFA is in state $u_0$. Thus, the NFA state set is initialized to $\{u_0\}$. This NFA state set will remain constant until the agent reaches an MDP state $s'$ for which $\texttt{at}(\texttt{key}) \in L(s')$ (i.e., the agent gets the key). Since the transition to $u_1$ is now possible in the NFA, but it is also possible to remain in $u_0$ because $\texttt{true}$ holds in $s'$, the NFA state set is updated to $\{u_0, u_1\}$. The state set will then remain constant until the agent reaches the door.

We note that the above procedure may lead to an empty NFA state set on some NFAs and state sequences. For example, notice that in the bottom NFA in Figure 2, there is no transition to follow when the agent enters a state with a nail. This occurs because once the agent is at a nail, the episode can never satisfy the advice formula regardless of how the rest of the episode proceeds. We call such situations *NFA dead-ends*. Since the advice may still be useful even if it has been violated, we handle NFA dead-ends as follows: if an NFA state set becomes empty, we revert it to the previous set. The NFA in Figure 2 will therefore continue to suggest that the agent avoid nails even if the agent already failed to do so.

## 3.3 Background Knowledge Functions

Advice like "get the key" would not be very useful if the agent had no notion of what behaviour might lead to the key. To give advice, we must presuppose that the advised agent has some capacity to understand and apply the advice. To this end, we assume that the agent has a *background knowledge function* $h_B : S \times A \times \mathsf{lit}(\Sigma) \to \mathbb{N}$, where $h_B(s, a, \ell)$ is an estimate of the number of primitive actions needed to reach the first state $s'$ where the literal $\ell$ is true (i.e, where $\ell \in L(s')$) if we execute $a$ in $s$. Intuitively, $h_B$ represents the agent's prior knowledge — which may not be perfectly accurate — about how to make ground atomic formulae either true or false.

Since $h_B$ only provides estimates with respect to individual ground literals, we believe that for many applications it should be relatively easy to manually define (or learn, e.g. [6]) a reasonable $h_B$. For example, in the grid world environment, we defined the background knowledge function so that

$$\begin{aligned} h_B(s, a, \texttt{at}(c)) = {} & |\texttt{pos}(agent, s).x - \texttt{pos}(c, s).x| + \\ & |\texttt{pos}(agent, s).y - \texttt{pos}(c, s).y| + \Delta \end{aligned}$$

where $\texttt{pos}(c, s)$ provides the coordinates of $c$ in state $s$ and $\Delta$ is equal to $-1$ if the action $a$ points toward $c$ from the agent's position and 1 if it does not. Hence, if the agent is three locations to the left of the key, $h_B(s, \text{right}, \texttt{at}(\text{key}))$ will return 2, even if there is a wall in the way. Furthermore, we defined $h_B(s, a, \neg\texttt{at}(c))$ to be equal to 1 if $h_B(s, a, \texttt{at}(c)) = 0$ and 0 otherwise.

Given any background knowledge function $h_B$, we can construct a function $h : S \times A \times \mathcal{L}_\Sigma \to \mathbb{N}$ that extends $h_B$ to provide an estimate of the number of primitive actions needed to satisfy an arbitrary formula $\varphi \in \mathcal{L}_\Sigma$. We recursively define $h(s, a, \varphi)$ as follows:

$$\begin{aligned} h(s,a,\ell) &= h_B(s,a,\ell) \text{ for } \ell \in \mathsf{lit}(\Sigma) \\ h(s,a,\psi \wedge \chi) &= \max\{h(s,a,\psi), h(s,a,\chi)\} \\ h(s,a,\psi \vee \chi) &= \min\{h(s,a,\psi), h(s,a,\chi)\} \end{aligned}$$

In the next sections, we describe how to drive the agent's exploration using $h$.

### 3.4 Advice-Based Action Selection

Suppose the agent is at a state $s$ in the MDP, with NFA state sets $q^{(0)}, \ldots, q^{(m)}$. Intuitively, following the advice in $s$ involves having the agent take actions that will move the agent through the edges of each NFA towards its accepting states. To this end, we begin by identifying *useful* NFA edges which may actually lead to progress towards the accepting states. Let us write $(q, \beta, q') \in \delta^{(i)}$ if there is an edge from $q$ to $q'$ that is labelled by the formula $\beta$ in the $i$th NFA. We say that an edge $(q, \beta, q') \in \delta^{(i)}$ is useful if $q$ is not an accepting state and there exists a path in the NFA from $q'$ to an accepting state that does not have $q$ along it. We then let *useful*$(q^{(i)})$ denote the set of all useful edges that are from NFA states in $q^{(i)}$. We now define the *advice guidance* formula $\hat{\varphi}$ as follows:

$$\hat{\varphi} \overset{\text{def}}{=} \bigvee_{i=0}^{m} \left[ \bigvee_{(q,\beta,q') \in \mathit{useful}(q^{(i)})} \texttt{to\_DNF}(\beta) \right]$$

where the function $\texttt{to\_DNF} : 2^{\mathcal{L}_\Sigma} \to \mathcal{L}_\Sigma$ converts the formula $\beta$ to disjunctive normal form. Notice that the formula $\hat{\varphi}$ will be satisfied by any action that achieves one of the formulas needed to transition over a useful edge. We define $\hat{h}(s, a) = h(s, a, \hat{\varphi})$ which can be used to rank how close each action is to making progress in satisfying the advice guidance formula $\hat{\varphi}$.

In addition to guidance, it is important to, if possible, avoid NFA dead-ends. To do so, we define the *advice warning* formula $\hat{\varphi}_w$ as follows:

$$\hat{\varphi}_w \overset{\text{def}}{=} \bigwedge_{i=0}^{m} \left[ \bigvee_{q \in q^{(i)} \text{ and } (q,\beta,q') \in \delta^{(i)}} \texttt{to\_DNF}(\beta) \right]$$

and use it to define the set $W(s) \overset{\text{def}}{=} \{a \in A(s) : h(s, a, \hat{\varphi}_w) \neq 0\}$. The idea is that $W$ contains those actions which the evaluation function predicts will lead to NFA dead-ends. In the next section, we describe how we recommend the agent to disfavor actions from $W$ while also being guided by $\hat{h}$.

### 3.5 Incorporating Advice in R-MAX

Model-based Reinforcement Learning solves MDPs by learning the transition probabilities and rewards. The R-MAX family of algorithms [2] are model-based

```
Function Rmax_with_advice($S$, $A$, $L$, $h_B$, $\varphi_{advice}$, $C$, $N$)
    $T_{\text{unknown}} \leftarrow \emptyset$; $\hat{p} \leftarrow$ initialize_empty_model();
    for $s, a \in S \times A$ do
        $n(s,a) \leftarrow 0$;
        $T_{\text{unknown}} \leftarrow T_{\text{unknown}} \cup (s,a)$;
    $t \leftarrow 0$; $\pi \leftarrow \emptyset$; $s \leftarrow$ get_initial_state();
    while $t < N$ do
        $t \leftarrow t + 1$;
        if is_terminal_state($s$) or cannot_be_reached($s$, $\hat{p}$, $T_{unknown}$) then
            $s \leftarrow$ get_initial_state();
        if $s \notin \pi$ then
            $\pi \leftarrow$ policy_towards_min_heuristic($\hat{p}$, $\varphi_{advice}$, $T_{unknown}$, $L$, $h_B$);
        $a \leftarrow$ sample_action($\pi(s)$);
        $s', r \leftarrow$ execute_action($s$, $a$);
        $\hat{p} \leftarrow$ update_model($\hat{p}$, $s$, $a$, $s'$, $r$);
        $n(s,a) \leftarrow n(s,a) + 1$;
        if $n(s,a) = C$ then
            $T_{\text{unknown}} \leftarrow T_{\text{unknown}} \setminus (s,a)$;
        $s \leftarrow s'$;
    return compute_optimal_policy($\hat{p}$);
```

**Algorithm 1:** R-Max with Advice algorithm.

methods that explore the environment by assuming that unknown transitions give maximal reward $R_{max}$. In practice, if $R_{max}$ is big enough, this means that the agent plans towards reaching the closest unknown transition.

We propose a simple variant of an R-MAX algorithm that can take advantage of the advice. Instead of planing towards *the closest* unknown transition, we plan towards the closest unknown transition that is not in $W$ and has minimum $\hat{h}$-value. If every reachable unknown transition is in $W$, then we just go to the closest unknown transition with minimum $\hat{h}$-value. This planning step can be done using LAO* [7].

Algorithm 1 shows the pseudo-code of our R-MAX variant. Its inputs are the MDP states $S$, action set $A$, labelling function $L$, background knowledge function $h_B$, the advice formula $\varphi_{\text{advice}}$, the number $C \geq 1$ of times that a transition has to be tried before being labeled as *known*, and the number of actions $N$ that the agent can execute during training. The algorithm starts by marking every state-action as unknown. It also defines an auxiliary variable $n(s,a)$ that counts how many times the transition $(s,a)$ has been tried and an empty model of the estimated environment's *transition probabilities* $\hat{p}$. In every iteration of the main loop, the agent selects the next action $a$ to execute using a partial policy $\pi$, which encodes the optimal way to reach the closest unknown transition with minimum $\hat{h}$-value that is not in $W$ (as previously described). Then, the agent executes $a$ and receives a reward $r$ and the next state $s'$ from the environment. The probabilities in the estimated model $\hat{p}$ are updated to reflect this observation of the transition $(s,a,r,s')$. Whenever a transition $(s,a)$ is tried $C$ times, it is removed from $T_{\text{unknown}}$. If a terminal state is reached or it is not possible to reach unknown transitions from the current state, the environment is restarted. After executing $N$ actions, the algorithm determines the optimal policy with respect to $\hat{p}$ (using, for instance, value iteration) and returns it. For a deterministic MDP, whenever $N$ is sufficiently large, that policy will also be optimal for the MDP.

**Theorem 1.** *Given an MDP with a deterministic transition function, there exists a number $N_0$ so that Algorithm 1 converges to an optimal policy if $N \geq N_0$.*

*Proof (sketch).* Since the MDP is deterministic, after an action is attempted even one time in a state, the estimate of the probabilities (all 0 or 1) over the outcomes of that state-action pair will be exact. So there will be exact estimates for all transitions the algorithm considers *known* (even if $C = 1$). As each episode ends after a finite number of steps, eventually all transitions reachable from the initial state become known, since the agent plans to visit unknown transitions as long as there are any (regardless of the advice). By that point, the MDP can be solved exactly by the agent, and so the optimal policy can be found.

For a non-deterministic MDP, we expect that our approach would converge to a near-optimal policy in the same sense that R-MAX does [2], but further investigation is needed.

## 4 Evaluation and Discussion

We tested our approach using various pieces of advice on grid world problems of the sort defined in Section 2, using the signature and background knowledge functions described in Sections 2 and 3. In addition to the domain elements of walls, keys, and nails described earlier, some problems also had *cookies* and *holes*. When the agent reaches a location with a cookie, it gets a reward of +10 and the cookie disappears. For reaching a hole, the reward is -1000 and the episode ends. As a baseline, we reported the performance of standard R-MAX, which does not use any form of advice. To measure performance, we evaluated the agent's policy every 100 training steps. At training time, the agent explored the environment to learn a good model. At test time, we evaluated the best policy that the agent could compute using its current model (ignoring the unknown transitions). We used a discount factor of 1 in all experiments.

Figure 4a shows the median performance (over 20 independent trials) in a $25 \times 50$ version of the motivating example in Figure 1. Our approach allows the agent to quickly find a policy that follows the advice and then slowly converges to an optimal policy. This is the case even with the deliberately misleading advice `N`, which told the agent to go to every nail. As the quality of the advice decreases, the agent's initial performance also does. We note that R-MAX, without advice, has poor initial performance, but converges faster to an optimal policy than when using the more detailed advice of `K` or `KD` (see Table 1 for what these formulae are). As such, there is a trade-off between quickly learning the model (exploring nearby areas) and moving towards promising states suggested by advice.

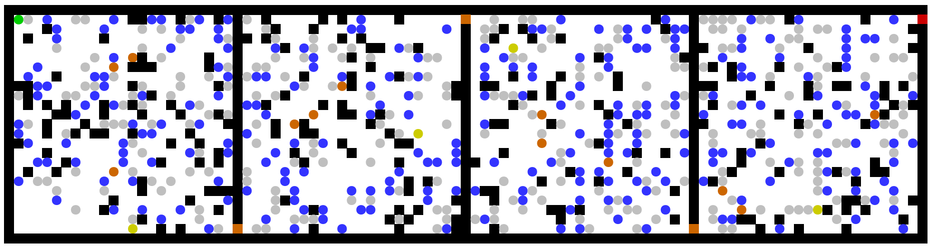
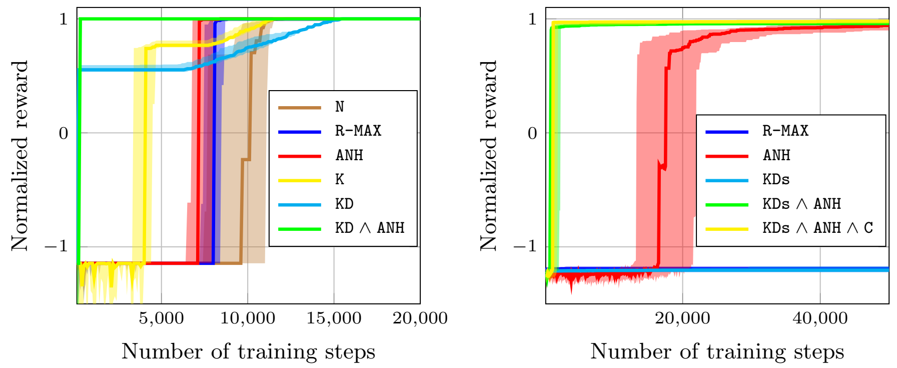


**Fig. 3.** An example randomly generated grid world map, containing an agent (●), walls (■), nails (●), holes (●), keys (●), cookies (●), doors (■), and a final door (■).



(a) Motivating example (25 × 50 version) (b) Random grid world maps

**Fig. 4.** In (a) and (b), reward is normalized so that 1 represents the best policy found on each map. The shaded areas represent the first and third quartiles. The formulae in the legends are explained by Table 1. Graphs are best viewed in colour.

We also randomly generated 10 grid world problems (Figure 3 shows one of them), each consisting of a sequence of four 25 × 25 rooms with doors between consecutive rooms. The agent always starts in the upper-left corner of the left-most room and the red (i.e., goal) door is on the rightmost wall. Each space within a room had a 1/9 probability of being each of a nail, hole, or wall. A key and three cookies were randomly placed in each room among the remaining empty spaces, such that each key, door, and cookie was reachable from the starting location. These grid world maps are challenging because there is sparse reward and it is fairly easy to die.

Figure 4b shows the performance over the 10 maps using four different pieces of advice. We ran 5 trials per piece of advice on each map; the graph reports the median performance across both trials and maps. Without advice, the agent was never able to reach the last door in 50 thousand training steps. Providing advice that the agent should get keys and go to doors (KDs) was also not enough, because the agent always fell in a hole before reaching its target. Stronger results were seen with safety-like advice to avoid holes and nails (ANH), and the best performance was seen when this safety-like advice was combined with advice that guides agent progress ($\texttt{KDs} \wedge \texttt{ANH}$ and $\texttt{KDs} \wedge \texttt{ANH} \wedge \texttt{C}$).

These results are encouraging. Firstly, they show the potential for advice to play a key role in scaling Reinforcement Learning. In particular, some problems are just too hard to expect RL to find good policies without the guidance provided by advice (e.g., Figure 4b). Secondly, they show that even well-intended advice could result in sub-optimal behaviour if it were treated as a hard constraint. For instance, the initial performance of KD in Figure 4a is a policy that satisfies the advice in an optimal number of steps, but gets sub-optimal reward (due to the nails). This further highlights the need for techniques like ours that are based on advice (guidance) instead of constraints (pruning). Still, our experiments are limited to a deterministic setting, and investigating the role of advice in non-deterministic domains is an important future work direction.

**Table 1.** Abbreviations for legends in Figure 4

| Abbreviation | Advice formula (and informal meaning) |
|---|---|
| R-MAX | *No advice*<br>Standard R-MAX. |
| C | $\forall(c \in \texttt{cookies}).\Diamond\texttt{at}(c)$<br>Get all the cookies. |
| ANH | $\Box(\forall(x \in \texttt{nails} \cup \texttt{holes}).\neg\texttt{at}(x))$<br>Avoid nails and holes. |
| K | $\Diamond\texttt{at}(\texttt{key})$<br>Get the key. |
| KD | $\Diamond(\texttt{at}(\texttt{key}) \wedge \bigcirc\Diamond(\texttt{at}(\texttt{door})))$<br>Get the key and then go to the door. |
| KDs | $\forall(k \in \texttt{keys}).\Diamond(\texttt{at}(k) \wedge \bigcirc\Diamond(\exists(d \in \texttt{doors}).\texttt{at}(d)))$<br>For every key in the map, get it and then go to a door. |
| N | $\forall(x \in \texttt{nails}).\Diamond\texttt{at}(x)$<br>Go to every nail. |

## 5 Related Work

The idea (and recognition of the importance) of constructing agents that can take advice of some sort dates back to John McCarthy's hypothetical *advice taker* system [8]. For MDPs, several works have proposed agents that accept *hard constraints* in the form of linear temporal logic safety constraints (e.g. [9,10]) or high level task specifications (e.g. [11,12]). Such constraints differ from the notion of advice studied in this paper. In particular, such hard constraints eliminate certain traces from consideration, potentially eliminating optimal policies. In contrast, advice only suggests behaviour and as such is more appealing when we want to provide guidance without pruning alternative behaviours.

In RL, the use of "advice" has been mainly focused on human *feedback* and *critique*. Human feedback allows an external observer to guide an agent (while it solves an MDP) by giving it positive and negative feedback (e.g. [13,14]), whereas critique proposes alternative actions to some states in a trace execution (e.g. [15,16]). In this work, we study a different modality of advice, which is given offline (prior to the agent's first interaction with the environment).

Maclin and Shavlik [17] were the first to propose using offline advice in RL. They defined a procedural programming language for encoding the advice. This language allows the user to recommend primitive actions using `If-Then` rules and loops. Since then, several extensions to this work have been done (e.g. [18,19]). Recently, Krening et al. [20] proposed a substantially different approach. They grounded natural language advice into a list of *(object,action)* recommendations (or warnings). The idea is to encourage (or discourage) the agent to perform a primitive action when it interacts with particular classes of objects. In contrast to those works, our advice language supports specification of temporally extended advice in terms of properties of states. For many simple pieces of LTL advice, such as *eventually get the key*, it is unclear how to express them as primitive action recommendations using the previous advice languages.

In another recent work, Andreas et al. [21] proposed *policy sketches*. A policy sketch is a sequence of sub-goals given by the user to solve a task. In a multi-task setting, Andreas et al. show how to use the sketch to compose (previously learned) policies for each sub-task to solve a novel task. However, their advice language is quite constrained in comparison with LTL advice. In particular, advice such as *avoid nails* or *get cookies* cannot be expressed as a sequence of sub-goals (unless some unnecessary order is imposed to sequentialize the advice).

## 6 Concluding Remarks

This work studied how to exploit linguistically expressed advice in Reinforcement Learning. Advice has the distinctive feature of being a recommendation, but not a task specification. As such, techniques for exploiting advice should provide guidance without pruning alternative, and possibly optimal, behaviours. Three main challenges arise when doing so. First, we need a common vocabulary to communicate the advice to the agent. We handled this by defining a *signature* over the states that is understandable by both the agent and the user. Then, sophisticated pieces of advice were constructed over the signature using LTL. Second, the agent needs some sort of background knowledge to be able to *follow* the advice. In this work, we introduced a background knowledge function for that purpose. Finally, the agent needs a way to reason about how and when to use the advice. Our approach constantly looks to *as soon as possible* reach promising states for advancing towards satisfying the advice. However, getting obsessed with following the advice might result in slower convergence to an optimal policy (as discussed in Section 4). We encourage future works to pay special attention to these three dimensions when advising RL agents.

**Acknowledgement.** This research was supported by NSERC and CONICYT. A preliminary *non-archival* version of this work was presented at RLDM (2017).

## References

1. Pnueli, A.: The temporal logic of programs. In: FOCS. (1977) 46–57
2. Brafman, R., Tennenholtz, M.: R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research **3** (2002) 213–231
3. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116**(1-2) (2000) 123–191
4. De Giacomo, G., Masellis, R.D., Montali, M.: Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: AAAI. (2014) 1027–1033
5. Baier, J., McIlraith, S.: Planning with first-order temporally extended goals using heuristic search. In: AAAI. (2006) 788–795
6. Peng, B., MacGlashan, J., Loftin, R., Littman, M., Roberts, D., Taylor, M.: A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In: AAMAS. (2016) 957–965
7. Hansen, E., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence **129**(1-2) (2001) 35–62
8. McCarthy, J.: Programs with common sense. RLE and MIT Computation Center (1960)
9. Lacerda, B., Parker, D., Hawes, N.: Optimal and dynamic planning for markov decision processes with co-safe LTL specifications. In: IROS. (2014) 1511–1516
10. Wen, M., Topcu, U.: Probably approximately correct learning in stochastic games with temporal logic specifications. In: IJCAI. (2016) 3630–3636
11. Andre, D., Russell, S.J.: Programmable reinforcement learning agents. In: NIPS. (2000) 1019–1025
12. Shapiro, D., Langley, P., Shachter, R.: Using background knowledge to speed reinforcement learning in physical agents. In: AA. (2001) 254–261
13. Isbell, C., Shelton, C.R., Kearns, M., Singh, S., Stone, P.: A social reinforcement learning agent. In: AA. (2001) 377–384
14. Knox, W.B., Stone, P.: Tamer: Training an agent manually via evaluative reinforcement. In: ICDL. (2008) 292–297
15. Judah, K., Roy, S., Fern, A., Dietterich, T.G.: Reinforcement learning via practice and critique advice. In: AAAI. (2010) 481–486
16. Griffith, S., Subramanian, K., Scholz, J., Isbell, C., Thomaz, A.L.: Policy shaping: Integrating human feedback with reinforcement learning. In: NIPS. (2013)
17. Maclin, R., Shavlik, J.: Creating advice-taking reinforcement learners. Machine Learning **22**(1-3) (1996) 251–281
18. Maclin, R., Shavlik, J., Torrey, L., Walker, T., Wild, E.: Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In: AAAI. (2005) 819–824
19. Kunapuli, G., Odom, P., Shavlik, J.W., Natarajan, S.: Guiding autonomous agents to better behaviors through human advice. In: ICDM. (2013) 409–418
20. Krening, S., Harrison, B., Feigh, K., Isbell, C., Riedl, M., Thomaz, A.: Learning from explanations using sentiment and advice in RL. IEEE Transactions on Cognitive and Developmental Systems **9**(1) (2016) 44–55
21. Andreas, J., Klein, D., Levine, S.: Modular multitask reinforcement learning with policy sketches. In: ICML. (2017) 166–175