Interactive Q-Learning with Ordinal Rewards and Unreliable Tutor

Paul Weng[1], Robert Busa-Fekete[2], and Eyke Hüllermeier[2]

[1] Université Pierre et Marie Curie, LIP6, Paris
[2] Department of Mathematics and Computer Science, Marburg University, Germany

Abstract. Conventional reinforcement learning (RL) requires the specification of a numeric reward function, which is often a difficult task. In this paper, we extend the Q-learning approach toward the handling of ordinal rewards. The method we propose is interactive in the sense of allowing the agent to query a tutor for comparing sequences of ordinal rewards. More specifically, this method can be seen as an extension of a recently proposed interactive value iteration (IVI) algorithm for Markov Decision Processes to the setting of reinforcement learning; in contrast to the original IVI algorithm, our method is tolerant toward unreliable and inconsistent tutor feedback.

Keywords: reinforcement learning, ordinal rewards, preference learning

1 Introduction

Reinforcement learning (RL) has proved to be successful in many domains (game playing [1], robotics [2], finance [3], amongst others). Yet, the definition of a numeric reward function, as required in the standard RL setting, is often difficult in practice, especially in situations where rewards do not represent a physical or objective measure (such as duration, distance, monetary costs/gains, etc.). For example, when trying to teach an RL agent to perform a high level task (e.g., driving fast around a racing track [4], spoken dialog system [5]), it is not obvious how to specify numeric feedback signals locally so as to induce a policy accomplishing that task in a globally optimal manner.

To alleviate this problem, our idea is to make the RL setting amenable to *ordinal* feedback signals, i.e., rewards measured on an ordinal scale. Thus, our approach relies on the assumption that, from a modeling point of view, dealing with ordinal rewards might be easier or more convenient than dealing with numerical ones. While certainly not true in general, this assumptions is tenable at least for certain types of applications. When measuring the health state of a patient, for example, a medical doctor might be able to provide feedback of the form "the patient is in a critical condition" or "the patient is doing well", whereas she will hardly be willing to quantify the state in terms of a precise number.

In the setting considered in this paper, an RL agent is acting on behalf of a user, who occasionally interacts with the learning system by providing tutorial feedback. After performing an action in a state, the agent receives an immediate ordinal reward, i.e., a value from a categorical, completely ordered scale. By counting (with a discount factor γ) the number of times each ordinal reward was obtained while moving through the environment, the cumulative reward can be represented by a corresponding counting vector—thus, the problem is translated from a single-dimensional ordinal to a multi-dimensional numeric one. In the course of the learning process, the agent is allowed to query the user/tutor for helping her to compare sequences of ordinal rewards (i.e., the corresponding counting vectors). Although such queries are answered correctly only with a certain probability, they provide useful information about the user's preferences, which are a-priori not known to the RL agent. In this setting, we propose a variant of the Q-Learning algorithm for finding a good policy.

The case of an unknown or partially known reward function has been considered in several studies in the context of Markov decision processes [6–9] as well as in reinforcement learning [10–12]. Besides, our work is related to preference-based reinforcement learning [13, 14], learning from demonstration [15] and apprenticeship learning [16]. However, closest to our setting is the work of Weng and Zanuttini [17], who proposed an interactive value iteration (IVI) algorithm for solving Markov decision processes if only the order of the rewards is known. In fact, our work can be seen as an extension of this approach to the setting of reinforcement learning; besides, as mentioned above, we also allow the tutor to make mistakes, whereas the approach of Weng and Zanuttini assumes an error-free tutor.

In the next section, we introduce notation and detail the formal setup of our approach. In Section 3, we elaborate on the question of how the RL agent can learn the user's preferences; to this end, we first introduce a deterministic representation of the feasible preference models and then propose a probabilistic generalization thereof. Our interactive Q-learning algorithm is then introduced in Section 4 and analyzed experimentally in Section 5. The paper ends with a summary and an outlook on future work in Section 6.

2 Notation and formal setup

2.1 Markov decision processes

A *Markov Decision Process* (MDP) is defined as a quintuple $\mathcal{M} = (S, A, p, r, \gamma)$, with S a finite set of states, A a finite set of actions, $p\colon S \times A \to \mathcal{P}(S)$ a transition function mapping state/action pairs to probability distributions over states, $r\colon S \times A \to \mathbb{R}$ a reward function, and $\gamma \in [0, 1[$ a discount factor [18].

A (stationary, deterministic) *policy* $\pi : S \to A$ associates an action with each state. Such a policy is assessed by a *value function* $v^\pi : S \to \mathbb{R}$ defined as follows:

$$v^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s, \pi(s), s') v^\pi(s') \tag{1}$$

Then, a preference relation is defined over policies as follows:

$$\pi \succsim \pi' \Leftrightarrow \forall s \in S, v^{\pi}(s) \geq v^{\pi'}(s)$$

A solution to an MDP is a policy that ranks highest with respect to $\succsim$. Such a policy, called *optimal policy*, can be found by solving the *Bellman equations*:

$$v^*(s) = \max_{a \in A} r(s,a) + \gamma \sum_{s' \in S} p(s,a,s') v^*(s') \tag{2}$$

As can be seen, the preference relation $\succsim$ over policies is directly induced by the reward function r.

2.2 Ordinal reward MDP

As introduced in [19], an *Ordinal Reward MDP* (ORMDP) is defined as an MDP $(S, A, p, \hat{r}, \gamma)$ in which the reward function $\hat{r}\colon S \times A \to E$ takes values in a qualitative, totally ordered scale $E = \{e_1 < e_2 \ldots < e_k\}$. Such an ORMDP can be reformulated as a Vector Reward MDP (VMDP) $(S, A, p, \overline{r}, \gamma)$, where $\overline{r}(s,a)$ is the vector in $\mathbb{R}^k$ whose i-th component is 1 for $\hat{r}(s,a) = e_i$ and 0 in the other components. Like in standard MDPs, the value function $\overline{v}^{\pi}$ of a policy π in a VMDP can be defined as follows:

$$\overline{v}^{\pi}(s) = \overline{r}(s, \pi(s)) + \gamma \sum_{s' \in S} p(s, \pi(s), s') \overline{v}^{\pi}(s') \ , \tag{3}$$

where sums and products over vectors are componentwise. This equation amounts to counting the number of ordinal rewards obtained by applying a policy. Therefore, a value function in a state can be interpreted as a multiset or bag of elements of E, and comparing policies essentially means comparing vectors. In the following, counting vectors will be denoted $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$, and the preference relation over such vectors by $\succeq$.

Proposition 1 *If E were a numerical scale, then $v^{\pi}(s) = \sum_{i=1}^{k} \overline{v}_i^{\pi}(s) e_i$.*

A natural dominance relation $\succeq_D$ on counting vectors is given by

$$\boldsymbol{\beta} \succeq_D \boldsymbol{\beta}' \Leftrightarrow \forall i = 1, \ldots, k : \sum_{j=i}^{k} \beta_j \geq \sum_{j=i}^{k} \beta'_j \tag{4}$$

This relation states that for any reward e_i, the number of rewards as good as or better than e_i is higher in $\boldsymbol{\beta}$ than in $\boldsymbol{\beta}'$. This dominance essentially corresponds to a translation of first-order stochastic dominance [20] to our setting. It can also be viewed as Pareto dominance over transformed vectors

$$L(\boldsymbol{\beta}) = \left(\beta_k, \beta_{k-1} + \beta_k, \ldots, \sum_{j=1}^{k} \beta_j \right).$$

Although $\succeq_D$ is a partial order relation and may not be very discriminating, we will nevertheless use it for eliminating vectors corresponding to policies that are dominated for every reward function.

In [19], it is shown that, under natural assumptions about the preference relation $\succeq$ over counting vectors (formalized in terms of corresponding axioms), there always exists a numerical function $\rho : E \to \mathbb{R}$ representing $\succeq$:

$$\forall \boldsymbol{\beta}, \boldsymbol{\beta}' : \boldsymbol{\beta} \succeq \boldsymbol{\beta}' \Longleftrightarrow \sum_{i=1}^{k} \beta_i \rho(e_i) \geq \sum_{i=1}^{k} \beta'_i \rho(e_i). \tag{5}$$

The existence of such an embedding of E in the reals has an important implication: An RL agent that optimally acts in the sense of $\succsim$ is behaviorally equivalent to a standard RL agent that optimally acts according to the numerical reward function $r = \rho \circ \hat{r}$. This is to some extent comparable to classical expected utility theory: A decision maker who obeys certain rationality axioms behaves *as if she were an expected utility maximizer* [21]. Consequently, her behavior can be mimicked by maximizing expected utility with properly defined rewards on outcomes.

Correspondingly, our idea is to reduce the learning of an optimal policy for our target ORMDP to the learning of an optimal policy for a standard MDP $\mathcal{M} = (S, A, p, r, \gamma)$ with numeric reward function $r = \rho \circ \hat{r}$. Of course, since this function is not known, our problem is actually more difficult and also involves learning the embedding ρ. This will be accomplished on the basis of feedback provided by the tutor, namely pairwise comparisons between counting vectors.

3 Learning the user's preferences

3.1 Reliable tutor

According to (5), the comparison of two counting vectors $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$ and $\boldsymbol{\beta}' = (\beta'_1, \ldots, \beta'_k)$ gives rise to a linear inequality of the form

$$\sum_{i=1}^{k} (\beta_i - \beta'_i) e_i \geq 0.$$

Correspondingly, provided the tutor's answers are always correct, the current knowledge about the numeric preference representation can be expressed as a polytope. Initially, this polytope merely encodes the order of the ordinal rewards and is specified by the inequalities:

$$\mathcal{K}_0 = \left\{ \begin{array}{l} e_2 - e_1 \geq \eta \\ e_3 - e_2 \geq \eta \\ \ldots \\ e_k - e_{k-1} \geq \eta \end{array} \right\}$$

where k is the (supposedly known) number of different ordinal rewards and η a small positive value, representing the smallest difference between any two

consecutive rewards. As shown in [22], two values can be set without loss of generality, e.g. $e_1 = 0$ and $e_k = 1$. As a side remark, we note that $\mathcal{K}_0$ could be enriched by more prior knowledge about reward values, provided this knowledge can be expressed in terms of linear inequalities.

Each answer to a query will reduce the size of the polytope, making the current knowledge more specific. Thus, starting from $\mathcal{K}_0$, the knowledge $\mathcal{K}_t$ at time step t is obtained from $\mathcal{K}_{t-1}$ by adding the inequality derived from the answer to the last query. Obviously, $\mathcal{K}_t$ can then also be used to compare two counting vectors $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$ and $\boldsymbol{\beta}' = (\beta'_1, \ldots, \beta'_k)$ that represent two sequences of ordinal rewards. To this end, one first solves the following linear program:

$$\begin{array}{ll} \text{min.} & \displaystyle\sum_{i=1}^{k} (\beta_i - \beta'_i) e_i \\ \text{s.t.} & \mathcal{K}_t \end{array}$$

If the optimal value of the objective function is non-negative, then $\boldsymbol{\beta} \succeq \boldsymbol{\beta}'$: The preferences of the tutor revealed so far imply that she must prefer $\boldsymbol{\beta}$ to $\boldsymbol{\beta}'$. Otherwise, the following program is solved:

$$\begin{array}{ll} \text{max.} & \displaystyle\sum_{i=1}^{k} (\beta_i - \beta'_i) e_i \\ \text{s.t.} & \mathcal{K}_t \end{array}$$

If the optimal value of the objective function is non-positive, then, for the same reason as above, $\boldsymbol{\beta}' \succeq \boldsymbol{\beta}$. Otherwise, the current knowledge $\mathcal{K}_t$ is not specific enough for comparing the two vectors. In this case, a definite answer can only be obtained by asking the tutor herself.

3.2 Unreliable tutor

The above approach only works under the assumption of an error-free tutor. Otherwise, if the tutor may report incorrect (reversed) preferences, the polytope $\mathcal{K}_t$ may easily collapse. Therefore, we opt for a more flexible approach that allows for handling "noisy" preferences. To this end, we model the feedback of the tutor in a *probabilistic* way. More specifically, we make use of a *mixed logistic model* or Bradley-Terry model [23]:

$$\begin{aligned} \mathbb{P}\big[\, (\beta_1, \ldots, \beta_k) \succeq (\beta'_1, \ldots, \beta'_k) \,|\, e_1, \ldots, e_k, c \,\big] &= \\ &= \frac{\exp\left(c \sum_{i=1}^{k} \beta_i e_i\right)}{\exp\left(c \sum_{i=1}^{k} \beta_i e_i\right) + \exp\left(c \sum_{i=1}^{k} \beta'_i e_i\right)} \\ &= \frac{1}{1 + \exp\left(-c \sum_{i=1}^{k} (\beta_i - \beta'_i) e_i\right)} \end{aligned} \tag{6}$$

where $c \geq 0$ is a real-valued parameter modeling the precision of the tutor (for $c \to \infty$, the model converges toward the error-free case, whereas for $c = 0$, she provides answers purely at random).

By using the maximum likelihood principle, one can fit the Bradley-Terry model to the answers of the unreliable tutor. Assume that the set of queries asked so far is given in the form $\mathcal{B} = \big\{(\boldsymbol{\beta}_1, \boldsymbol{\beta}'_1), \ldots, (\boldsymbol{\beta}_n, \boldsymbol{\beta}'_n)\big\}$, where $\boldsymbol{\beta}_i = (\beta_{i,1}, \ldots, \beta_{i,k})$ and $\boldsymbol{\beta}'_i = (\beta'_{i,1}, \ldots, \beta'_{i,k})$ are the vectors that were compared in the i-th query. Without loss of generality, we can assume that the answer of the tutor was $\boldsymbol{\beta}_i \succeq \boldsymbol{\beta}'_i$ for all $1 \leq i \leq n$. Then, the log-likelihood function can be written as follows:

$$\mathcal{L}(e_1, \ldots, e_k, c \,|\, \mathcal{B}) = \sum_{\ell=1}^{N} \log \mathbb{P}\big[\, \boldsymbol{\beta}_\ell \succeq \boldsymbol{\beta}'_\ell \,|\, e_1, \ldots, e_k, c \,\big]. \tag{7}$$

Note that the scale parameter c should not be tuned, but assumed to be constant. In this way, the parameters $e_1, \ldots, e_k$ found are not scaled, nevertheless, after the optimization one can rescale them so as $e_k = 1$.

In order to obtain estimates and confidence intervals for the model parameters, we apply a basic form of non-parametric bootstrap [24]. This method can be summarized as follows: A *bootstrap sample* $\bar{\mathcal{B}}$ is obtained by randomly sampling n times from $\mathcal{B}$ with replacement. We repeat this resampling m times to obtain a set of independent bootstrap samples $\{\bar{\mathcal{B}}_1, \ldots, \bar{\mathcal{B}}_m\}$. Then, by optimizing the log-likelihood function (7) for each of these m samples, m different parameter estimates can be obtained. Finally, confidence intervals for the model parameters can be calculated based on the empirical quantiles from the bootstrap distribution. This approach is called *percentile interval* (for more details, see Section 13 in [24]).

Based on the confidence intervals of the model parameters, it is possible to estimate the uncertainty of a new prediction obtained by the model. To this end, denote the confidence interval of e_i by $[\ell_i, u_i]$, $1 \leq i \leq k$[3].Then, obeying the constraints imposed by these intervals, the highest possible score for two vectors $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ is given by

$$s_{\max}(\boldsymbol{\beta}, \boldsymbol{\beta}') = \max_{e'_i \in [\ell_i, u_i]} \mathbb{P}\big[\, \boldsymbol{\beta} \succeq \boldsymbol{\beta}' \,|\, e'_1, \ldots, e'_k, c \,\big].$$

Likewise, the lowest score is

$$s_{\min}(\boldsymbol{\beta}, \boldsymbol{\beta}') = \min_{e'_i \in [\ell_i, u_i]} \mathbb{P}\big[\, \boldsymbol{\beta} \succeq \boldsymbol{\beta}' \,|\, e'_1, \ldots, e'_k, c \,\big].$$

For the model given in (6), the interval $[s_{\min}(\boldsymbol{\beta}, \boldsymbol{\beta}'), s_{\max}(\boldsymbol{\beta}, \boldsymbol{\beta}')]$ is easy to calculate. Based on this interval, we can decide whether our model ranks $\boldsymbol{\beta}$ significantly higher than $\boldsymbol{\beta}'$ ($s_{\min}(\boldsymbol{\beta}, \boldsymbol{\beta}') > 1/2$), $\boldsymbol{\beta}'$ significantly higher than $\boldsymbol{\beta}$ ($s_{\max}(\boldsymbol{\beta}, \boldsymbol{\beta}') < 1/2$), or whether the prediction is not considered confident enough to reliably compare $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ ($s_{\min}(\boldsymbol{\beta}, \boldsymbol{\beta}') \leq 1/2 \leq s_{\max}(\boldsymbol{\beta}, \boldsymbol{\beta}')$).

[3] Here one can use the rescaled values for $e_1, \ldots, e_k$ where $e_k = 1$, but in this case, the confidence intervals should be also rescaled appropriately.

4 Interactive Q-learning

In this section, we extend a well-known model-free reinforcement learning algorithm called Q-learning [25], so as to make it amenable to vectorial reward functions as described in Section 2.2. We consider the infinite horizon case, where the discount factor is denoted by γ. Furthermore, we assume the availability of a generative model, whence policy learning can be carried out in an online fashion.

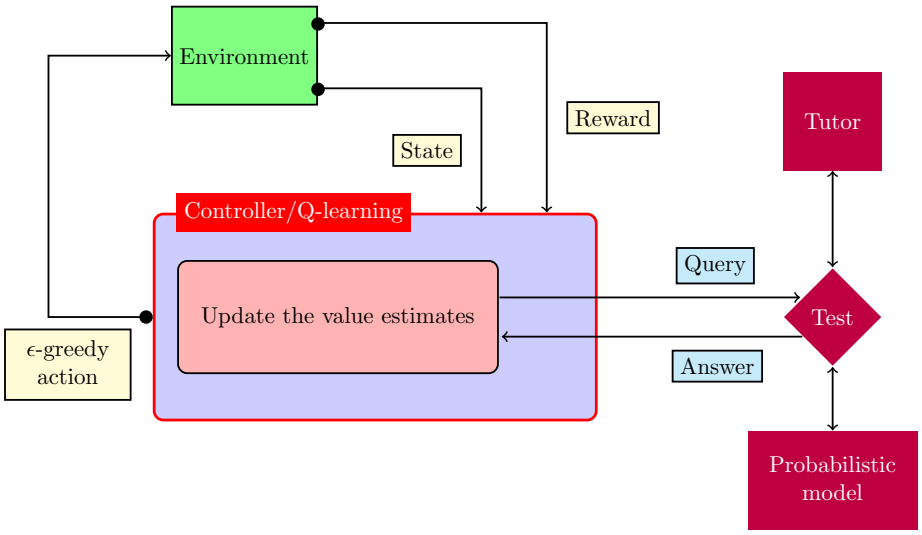

Fig. 1. A schematic overview of our policy search framework.

Figure 1 provides a schematic overview describing the process of policy search in our setup. Like in standard reinforcement learning, the RL agent and the environment are interacting with each other via actions, states and rewards. Based on the feedback coming from the environment, which here consists of a state and a vectorial reward, the agent updates its model after each step. In our ordinal reinforcement learning setup, the agent's goal is to approximate a vectorial action-value function for each state. However, to select the best action (based on the vectorial action-value function approximation), it has to be aware of the preference relation $\simeq$ over vectorial rewards. Here, we assume that the agent can query the tutor about the preferences $\simeq$ of vector pairs.

Since querying the tutor/expert might be expensive, the number of queries asked during the policy search should be kept as low as possible. As explained previously, we fit a logistic model for the answers given by the tutor so far, and compute the confidence intervals of the model parameters. When the agent needs to compare two new vectors, the logistic model is used first. If the model is confident enough, we simply adopt its prediction and do not query the tutor.

Algorithm 1 Interactive Q-Learning$(S, A, \hat{r}, \gamma)$

1: $t \leftarrow 0$
2: compute $\overline{r}$ from $\hat{r}$
3: $\mathcal{B} = \emptyset$
4: $\forall s \in S, \forall a \in A$, $\overline{Q}(s,a) \leftarrow (0, \ldots, 0)$
5: $\forall s \in S$, $B(s) \leftarrow$ Random action from A ▷ Keep track of the best action
6: **repeat**
7: Choose action a_t (by following, for example, an ϵ-greedy policy)
8: $(s_{t+1}, \overline{r}_{t+1}) \leftarrow \text{Simulate}(s_t, a_t)$
9: $\delta_t \leftarrow \overline{r}_{t+1} + \gamma \overline{Q}(s_{t+1}, B(s_{t+1})) - \overline{Q}(s_t, a_t)$
10: $\overline{Q}(s_t, a_t) \leftarrow \overline{Q}(s_t, a_t) + \alpha_t \delta_t$
11: **if** $a_t \neq B(s_t)$ **then**
12: $[B(s_t), \mathcal{M}, \mathcal{B}] \leftarrow \text{getBest}(\overline{Q}(s_t, a_t), a_t, \overline{Q}(s_t, B(s_t)), B(s_t), \mathcal{M}, \mathcal{B})$
13: **else**
14: **for** $a \in A$ **do**
15: $[B(s_t), \mathcal{M}, \mathcal{B}] \leftarrow \text{getBest}(\overline{Q}(s_t, a), a, \overline{Q}(s_t, B(s_t)), B(s_t), \mathcal{M}, \mathcal{B})$
16: $t \leftarrow t + 1$
17: **until** stopping condition
18: **return** $\overline{Q}$ and B

Otherwise, the tutor is queried, and the model is refitted based on the answer given.

Algorithm 1 shows the pseudo-code of the extended Q-learning algorithm. Since we are dealing with a vectorial Q-function, the operations made with $\overline{Q}(.,.)$ are meant componentwise. Beside the Q-value estimates, we also need to keep track of the actions that are currently considered best for each state; in our setting, figuring out which action is best for a given state is indeed more complex than comparing two real-valued numbers like in standard value-based MDPs. We denote the function that stores the current best action for a state by $B : S \to A$.

The core of Algorithm 1 is the same as for the original Q-learning method: select an action a_t, generate reward and next state based on the generative model, and update the Q-function estimate (lines 7 – 10). If the action a_t selected in line 7 is not the current best action ($a_t \neq B(s_t)$), then we only need to check whether the Q-value estimate for (s_t, a_t) has improved compared to $\overline{Q}(s_t, B(s_t))$ (i.e., the vectorial Q-value estimate for the current best action $B(s_t)$). The function *getBest* (to be detailed below) implements this comparison. In the case where the selected action a_t coincides with the current optimal one ($a_t = B(s_t)$), we have to check whether the Q-value estimate $\overline{Q}(s_t, a_t)$ for the current best action has really preserved its dominance over all other actions (lines 14–15).

Algorithm 2 implements the comparison between two vectors mentioned above, that is, it decides whether our current probabilistic model is reliable enough to decide which vector of rewards, either $\boldsymbol{\beta}$ or $\boldsymbol{\beta}'$, is preferred, or whether this decision should better be left to the tutor. The set of preferences $\mathcal{B}$ obtained from the tutor so far is provided as an input parameter. If $\mathcal{B}$ is empty, the tutor

Algorithm 2 getBest($\boldsymbol{\beta}, a, \boldsymbol{\beta}', a', \mathcal{M}, \mathcal{B}$)

```
1: if B = ∅ then
2:     g = 1/2, c = 1/2
3: else
4:     [g, c] ← M(β, β′)          ▷ Forecast the preference by using the current model
5: if 1/2 ∉ [g − c, g + c] then                          ▷ The forecast is confident
6:     if g > 1/2 then
7:         return < a, M, B >
8:     else
9:         return < a′, M, B >
10: else                                          ▷ The forecast is not reliable enough
11:     Ask the tutor about the relation of β and β′
12:     if β ⪰ β′ then
13:         B ← B ∪ {(β, β′)}
14:         M ← Build model on B
15:         return < a, M, B >
16:     else
17:         B ← B ∪ {(β′, β)}
18:         M ← Build model on B
19:         return < a′, M, B >
```

has to be queried anyway. Otherwise, we calculate the score and the confidence interval produced by our current model $\mathcal{M}$ as described in Section 3.2. If the model's prediction is confident enough (line 6), we simply return the predicted preference. Otherwise, we query the tutor and add her answer to $\mathcal{B}$, refit the model (line 14), and return the answer of the tutor along with the updated model.

5 Experimental results

To validate our approach, we applied the interactive Q-learning algorithm in environments modeled as random instances of MDPs [9]. Each random instance can be described as $\mathcal{M} = (S, A, p, r, \gamma)$, where each pair (s, a) has $\lceil \log_2(|S|) \rceil$ possible successors, the unknown rewards are integers drawn uniformly between 0 and 99, and the discount factor γ is set to 0.95. The algorithm is stopped after 10^5 iterations. All results are averaged over 20 runs.

In a first series of experiments, we tried our algorithm for different sizes $k = |E|$ of the ordinal scale E, namely $k = 5, 10, \ldots, 25$, while fixing $|S| = 100$ and $|A| = 5$. The experimental results are shown in Figure 2. Without surprise, we observe that the relative number of queries to the tutor with respect to the number of all queries is increasing with the number of different reward values indicating that the preference learning task is becoming more complex with the size of ordinal rewards scale.

In a second series of experiments, we tested our algorithm on different numbers of states $|S| = 100, 200, \ldots, 500$, with $|A| = 5$ and $k = 20$ fixed. The exper-

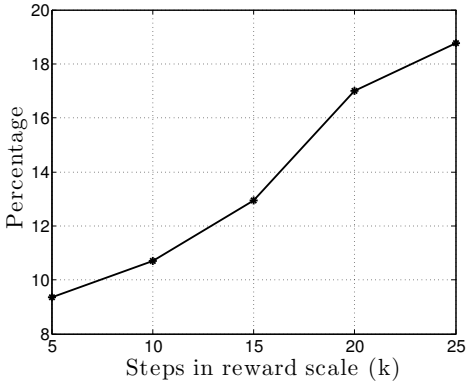

Fig. 2. The percentage of times the tutor was queried with respect to all queries (tutor or model) asked as a function of the size of the ordinal reward scale.

imental results are shown in Figure 3. We observe that larger sizes of MDPs are more difficult to solve, and thus, the tutor is queried relatively more, indicated by slightly more queries and less usage of the model.

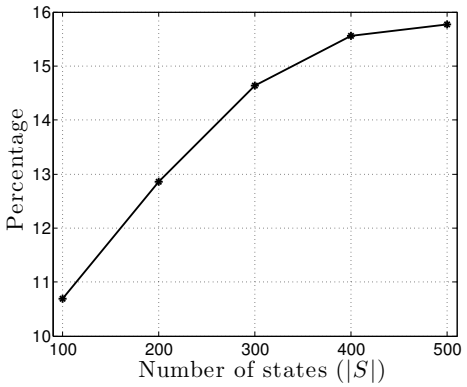

Fig. 3. The percentage of times the tutor was queried with respect to all queries (tutor or model) asked in function of the number of states.

Finally, we have also analyzed whether the use of our probabilistic model as a surrogate of the tutor, or at least a partial surrogate, may have a negative impact on the quality of the learned policy. To this end, we compared our method with a variant in which the agent always queries the tutor (and does not learn a preference representation)—this is actually equivalent to standard Q-learning using the true reward function. Assuming numerical scale E, we computed the

percentage of loss expressed as

$$\frac{\sum_s \big(v(s) - V(s)\big)}{|S| \max_{s,a} r(s,a)} \times 100 \ ,$$

where $v(s) = \max_a \sum_{i=1}^{k} \overline{Q}(s,a).e_i$ and $V(s) = \max_a Q(s,a)$ are, respectively, the estimated value functions computed by Q-learning with and without preference model. As shown in Figure 4, the average percentage of loss is lower than 1% on different sizes of the state space.

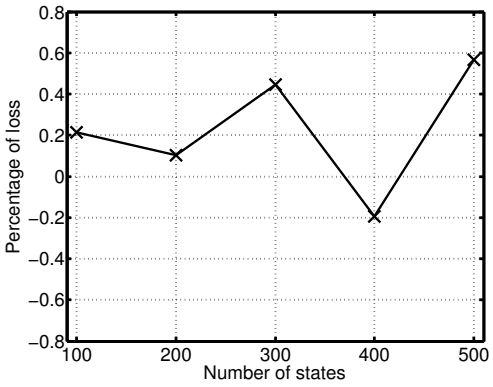

Fig. 4. Percentage of loss between Q-learning with and without model.

6 Conclusion

In this paper, we proposed a reinforcement learning algorithm based on Q-learning, which can be applied in settings where environmental feedback is provided in the form of ordinal rewards. In this context, the learning agent is allowed to consult a tutor, who can (unreliably) answer to preference queries comparing sequences of rewards. Moreover, we experimentally showed that our method, which uses a Bradley-Terry model for learning the tutor's preferences from her answers, is indeed helpful in learning a good policy by lowering the number of queries needed.

Our preliminary experimental results are promising. For future work, we nevertheless plan to test our algorithm more thoroughly, especially on real benchmarks that are more realist than random instances of MDPs. Besides, it would be interesting to investigate more sophisticated strategies for querying the tutor, thereby decreasing the number of queries even further.

Acknowledgments. This work was supported by the German Research Foundation (DFG) within the scope of the "Autonomous Learning" priority program, and by the ANR-10-BLAN-0215 grant of the French National Research Agency.

References

1. Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), 1995.
2. J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *IEEE-RAS International Conference on Humanoid Robots*, 2003.
3. Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *ICML*, 2006.
4. Tim C. Kietzmann and Martin Riedmiller. The neuro slot car racer: Reinforcement learning in a real world setting. In *International Conference on Machine Learning and Applications*, pages 311–316, 2009.
5. Lihong Li, Jason D. Williams, and Suhrid Balakrishnan. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. In *Interspeech*, 2009.
6. R. Givan, S. Leach, and T. Dean. Bounded-parameter Markov decision process. *Artif. Intell.*, 122(1-2):71–109, 2000.
7. F.W. Trevizan, F.G. Cozman, and L.N. de Barros. Planning under risk and Knightian uncertainty. In *IJCAI*, pages 2023–2028, 2007.
8. J.Y. Yu and S. Mannor. Online learning in markov decision processes with arbitrarily changing rewards and transitions. In *IEEE Game Theory for Networks (GAMENETS)*, pages 314–322, 2009.
9. K. Regan and C. Boutilier. Regret based reward elicitation for Markov decision processes. In *UAI*, pages 444–451, 2009.
10. A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.
11. Arkady Epshteyn, Adam Vogel, and Gerald DeJong. Active reinforcement learning. In *ICML*, 2008.
12. U.A. Syed. *Reinforcement learning without rewards*. PhD thesis, Princeton University, 2010.
13. J. Fürnkranz, E. Hüllermeier, W. Cheng, and S.H. Park. Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. *Machine Learning*, 89(1):123–156, 2012.
14. R. Akrour, M. Schoenauer, and M. Sebag. Preference-based policy learning. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.
15. S. Schaal. Learning from demonstration. In *NIPS*, 1997.
16. Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.
17. P. Weng and B. Zanuttini. Interactive value iteration for markov decision processes with unknown rewards. In *International Joint Conference on Artificial Intelligence*, 2013.
18. M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley, 1994.
19. P. Weng. Markov decision processes with ordinal rewards: Reference point-based preferences. In *ICAPS*, volume 21, pages 282–289, 2011.

20. M. Shaked and J.G. Shanthikumar. *Stochastic Orders and Their Applications*. Academic press, 1994.
21. J. von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.
22. P. Weng. Ordinal decision models for Markov decision processes. In *ECAI*, volume 20, pages 828–833, 2012.
23. R.A. Bradley and M.E. Terry. Rank analysis of incomplete block designs, i. the method of paired comparisons. *Biometrika*, 39:324–345, 1952.
24. B. Efron and R.J. Tibshirani. *An introduction to the bootstrap*, volume 57. Chapman & Hall/CRC, 1994.
25. R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.