Int J Speech Technol (2014) 17:325–340
DOI 10.1007/s10772-014-9224-x

Dialogue POMDP components (Part II): learning the reward function

H. Chinaei · B. Chaib-draa

Received: 10 July 2013 / Accepted: 11 January 2014

Abstract The partially observable Markov decision process (POMDP) framework has been applied in dialogue systems as a formal framework to represent uncertainty explicitly while being robust to noise. In this context, estimating the dialogue POMDP model components (states, observations, and reward) is a significant challenge as they have a direct impact on the optimized dialogue POMDP policy. Learning states and observations sustaining a POMDP have been both covered in the first part (Part I), whereas this part (Part II) covers learning the reward function, that is required by the POMDP. To this end, we propose two specific algorithms based on inverse reinforcement learning (IRL). The first is called POMDP-IRL-BT (BT for belief transition) and it approximates a belief transition model, similar to the Markov decision process transition models. The second is a point-based POMDP-IRL algorithm, denoted by PB-POMDP-IRL (PB for point-based), that approximates the value of the new beliefs, which occurs in the computation of the policy values, using a linear approximation of expert beliefs. Ultimately, we apply the two algorithms on healthcare dialogue management in order to learn a dialogue POMDP from dialogues collected by SmartWheeler (an intelligent wheelchair).

Keywords Partially observable Markov decision processes (POMDP) · Inverse reinforcement learning · Healthcare dialogue management

H. Chinaei · B. Chaib-draa (✉)
Department of Computer Science, Laval University,
Québec, Canada
e-mail: chaib@ift.ulaval.ca

H. Chinaei
e-mail: hchinae@gmail.com

1 Introduction

As stated in Part I, in intention-based dialogue domains, performing the best action in each dialogue state is a challenging task due to the uncertainty introduced by automatic speech recognition (ASR) errors and natural language understanding (NLU) problems as well as the stochastic environment made by user behavior change. In stochastic domains where the decision making is sequential and the state is partially observable, the suitable formal framework is the partially observable Markov decision process (POMDP). This framework has been extensively used to model the uncertainty of SDSs (spoken dialogue systems) (Roy et al. 2000; Zhang et al. 2001a, b; Williams and Young 2007; Thomson 2009; Gašić 2011; Pinault and Lefèvre 2011).

In POMDPs, the agent does not know in which state it is, since the state is partially observable. In consequence, it must maintain a probability distribution, known as the belief state, $b(s)$, over all the possible states s. At each time, the agent has some belief state $b(s)$; it then takes an action a while observing o causing thus the environment to change from state s to state s'. In these conditions, $b(s)$ should take in consideration the new observation o and the transition from s to s' and be updated into $b(s')$ (more detail in the background section below). Solving a POMDP aims to maximize the expected discounted rewards over an infinite horizon and this is equivalent to find the optimal policy π^* where π is the agent's policy and it specifies an action $a = \pi(b)$ for any belief b. Solving a POMDP is also called model-based reinforcement learning (RL).

Estimating the POMDP model components, such as states, observations and reward function is a significant issue; as the POMDP model has direct impact on the POMDP policy and consequently on the applicability of the POMDP in the domain of interest. In this context, the Spoken Dialogue

System (SDS) researchers in both academia and industry have addressed several practical challenges of applying POMDPs to SDS (Roy et al. 2000; Williams 2006; Paek and Pieraccini 2008). In particular, learning the SDS dynamics ideally from the available unannotated and noisy dialogues is a challenge for researchers.

Solving a POMDP, i.e., model-based RL, works by optimizing a *defined* reward function in the (PO)MDP framework. To do that, choice of the reward function has been usually hand-crafted based on the domain expert intuition. In some cases however, it is more convenient for the expert to demonstrate the policy. When the expert demonstrates the policy and the reward can be approximated from this policy, the method is called inverse reinforcement learning (IRL). IRL started with an environment completely observable, using Markov decision process (MDP). For this type of environment, Ng and Russell (2000) proposed multiple IRL algorithms that work by maximizing the sum of the margin between the policy of the expert and the intermediate *candidate* policies. These algorithms account for the case in which the expert's policy is represented *explicitly* and the case where the expert's policy is known only through observed expert's *trajectories*. Then, these IRL algorithms have been extended to POMDP.

IRL in POMDPs, in short POMDP-IRL, is particularly challenging due to the difficulty in solving POMDPs. Recently, Choi and Kim (2011) proposed POMDP-IRL algorithms by extending MDP-IRL algorithms of Ng and Russell (2000) to POMDPs. In particular, Choi and Kim (2011) provided a general framework for POMDP-IRL by modeling the expert's policy as a finite state controller (FSC) and thus using point-based policy iteration (PBPI) (Ji et al. 2007) as POMDP solver. The trajectory-based algorithms proposed by Choi and Kim (2011) also required the FSC-based POMDP solvers (PBPI). Since such algorithms spent most of the time solving the intermediate policies, they suggested modifying them to be able to use other solvers such as point-based value iteration (PBVI) ones.

In this paper, we extend the trajectory-based MDP-IRL algorithm of Ng and Russell (2000) to POMDPs. Figure 1 shows the cycle of interaction between an agent and its partially observable environment. In this figure, circles represent learned models. The model denoted by POMDP includes the POMDP model components (states and observations) which have been learned from Part I (Chinaei and Chaib-draa 2014). The learned POMDP together with action/observation *trajectories* are used in IRL to learn the reward function, denoted by R. Then, the learned POMDP and reward function are used in a POMDP solver to learn/update the optimal policy. In our IRL algorithms, we assume that the model components are known, similar to (Ng and Russell 2000; Choi and Kim 2011). Fortunately, in dialogue management, the transition and observation models can be calculated from Wizard-of-Oz data (Choi and Kim 2011) or a real system data, as mentioned in Part I (Chinaei and Chaib-draa 2014). In particular, we proposed in Part I, methods for learning such components from data and showed the illustrative example of learning the dialogue POMDP model components from SACTI-1 dialogues, collected in a Wizard-of-Oz setting (Williams and Young 2005). Then, the learned dialogue POMDP model together with expert's trajectories can be used in IRL algorithms to learn a reward function for the expert's policy and that's what we are proposing in this second part (Part II).

The rest of the paper is organized as follows. We describe the necessary background relative to MDP-IRL in Sect. 3. In Sect. 4, we cover our POMDP-IRL methods : POMDP-IRL-BT (BT for belief transition) and PB-POMDP-IRL (PB for point-based). Then in Sect. 5, we see how our two methods perform on real dialogues collected by SmartWheeler (an intelligent wheelchair). We first revisit the same POMDP background as Part I, so that we can make this second part self-contained.

2 Background

Since a POMDP is an extension of Markov decision process (MDP) to partial observability, let's start by a brief background on MDP. An MDP is defined as $(S, A, T, \gamma, R, s_0)$ where, S is the set of discrete states, and A is the set of discrete actions. In addition, the transition model T consists of the probabilities of state transitions (dynamics of the system): $T(s, a, s') = Pr(s_{t+1} = s' | a_t = a, s_t = s)$ where s is the current state and s' is the next state. Furthermore, the reward function is $R(s, a)$ that specifies the reward of taking action a in the state s. Then γ is a number between 0 and 1 that discounts the future reward. Finally, the initial state is denoted by s_0.

In an MDP, a policy π maps each state s to an action a, i.e., $a = \pi(s)$ and the objective is to find an optimal policy π^*, where for any state s, π^* specifies an action $a = \pi^*(s)$ that *maximizes* the expected value of future rewards starting from state s:

$$V^\pi(s) = E_{s_t \sim T}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi, s_0 = s\right] \tag{1}$$

The expected value of the reward can be computed recursively by Bellman equation defined as:

$$V^\pi(s) = \left[R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s')\right] \tag{2}$$

A POMDP is a more generalized AI model for planning under uncertainty. The POMDP framework considers that the states are only partially observable through some *observations*. Thus, a POMDP is represented as a tuple $(S, A, T, \gamma, R, O, \Omega, b_0\}$. That is, a POMDP model includes

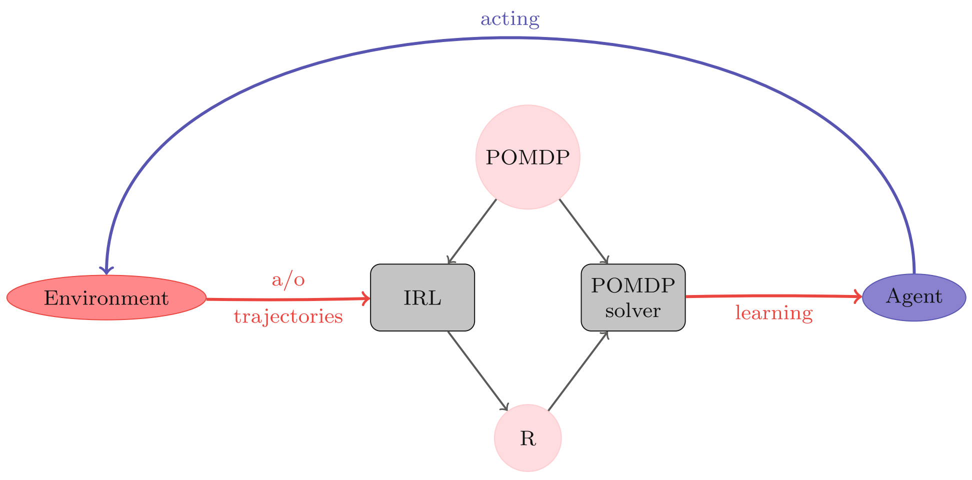

Fig. 1 The cycle acting/learning between the agent and environment. The circles represent the models. The model denoted by POMDP includes the POMDP model components, without a reward function, learned from Part I (Chinaei and Chaib-draa 2014). The learned POMDP model together with action/observation trajectories are used in IRL to learn the reward function denoted by R. The learned POMDP and reward function are used in the POMDP solver to learn/update the policy

an MDP model and adds O, the set of observations, Ω is the observation model defined as $\Omega(a, s', o') = Pr(o'|a, s')$ for the probability of observing o' after taking the action a which results in the state s'. Finally, the initial belief over all states is denoted by b_0.

The primary difference between POMDPs and MDPs is that, where an MDP has a particular current state, a POMDP (theoretically) maintains a probability distribution over all possible states. This distribution is called a *belief state* as stated previously. Given a belief state b, $b(s)$ is the probability assigned to a particular state s. After an action a is taken and observation o' is received, a new belief state, b', can be computed. This is done by using a *state estimator function* $(SE(b, a, o'))$ to calculate the probability of a new state s', $b'(s')$:

$$\begin{aligned} b'(s') &= SE(b, a, o') = Pr(s'|b, a, o') \\ &= \eta\Omega(a, s', o') \sum_{s \in S} b(s) T(s, a, s') \end{aligned} \tag{3}$$

where η is the normalization factor.

Note that an important property of the belief state is that it is a *sufficient statistic*. In other words, the belief state at time t, b_t, summarizes the initial belief b_0, as well as all the actions taken and all the observations received. Formally, we have: $b_t(s) = Pr(s|b_0, a_0, o_0, \ldots, a_{t-1}, o_{t-1})$.

In the POMDP framework, policy selects an action a for a belief state $b(s)$, i.e., $a = \pi(b(s))$. In this case, the objective in a POMDP aims to find an optimal policy π^*, where for any belief state b, π^* specifies an action $a = \pi^*(b)$ that maximizes the expected discount of future rewards starting from belief b_0:

$$V^{\pi}(b) = E_{b_t \sim SE}\left[\sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b\right], \tag{4}$$

where

$$R(b, a) = \sum_{s \in S} b(s) R(s, a) \tag{5}$$

Then, we have $\pi^*(b) = \arg\max_{\pi} V^{\pi}(b)$ that is the optimal policy. The expected value of the reward can be computed recursively using Bellman equation for V^{π}.

$$\left[V^{\pi}(b) = R(b, \pi(b)) + \gamma \sum_{o' \in O} Pr(o'|b, \pi(b)) V^{\pi}(b')\right] \tag{6}$$

3 Inverse reinforcement learning in the MDP framework (MDP-IRL)

In IRL, given an expert's policy and an underlying MDP, the problem is to learn a reward function that makes the expert's policy optimal. That is, given the expert's policy, approximate a reward function for the MDP such that the optimal policy of the MDP fits the expert's policy. In fact, IRL is an *ill-posed* problem. That is, there is not a single reward function that makes expert's policy optimal, but infinitely many of them. Since there are many reward functions that makes the expert's policy optimal, one approach is based on linear programming to find one of the possible solutions. The linear program constraints the set of possible reward functions where the rewards are represented as a linear representation of dialogue features, and finds a solution among the limited set of solutions.

In this section, we first describe IRL for MDPs (MDP-IRL) using expert trajectories, represented as $(s_0, \pi_E(s_0)$, $\ldots, s_{|S|-1}, \pi_E(s_{|S|-1}))$. To begin let us first recall that in MDP, a policy π maps each state s to an action a, i.e., $a = \pi(s)$ and the objective is to find an optimal policy π^*, where for any state s, π^* specifies an action $a = \pi^*(s)$ that *maximizes* the expected value of future rewards starting from state s:

$$V^{\pi}(s) = E_{s_t \sim T}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi, s_0 = s\right] \tag{7}$$

The expected value of the reward can be computed recursively by Bellman equation defined in Eq. (2).

Now let us introduce the following definitions:

- an *expert reward function*, denoted by R^{π_E}, is an unknown reward function for which the optimal policy is the *expert policy*:
  - the *expert's policy*, denoted by π_E, is a policy of the underlying MDP that optimizes the expert reward function R^{π_E},
  - the *value of the expert's policy*, denoted by V^{π_E}, is the value of the underlying MDP in which the reward function is the expert reward function R^{π_E}.
- a *candidate reward function*, denoted by R, is a reward function that could potentially be the expert reward model. We have the following definitions:
  - the *candidate policy*, denoted by π, is a policy of the underlying MDP that optimizes the candidate reward function R,
  - the *value of the candidate policy*, denoted by V^{π}, is the value of the candidate policy π that optimizes the candidate reward R.

Then, IRL aims to find a reward function in which the expert's policy is both optimal and maximally separated from other policies. To do this, some candidate reward functions and their subsequent policies are generated from the expert's behavior. The candidate reward function is approximated by maximizing the value of the expert's policy with respect to all candidate policies. The new candidate reward function and policy are then used to approximate another new set of models. This process iterates until the difference in values of successive candidate policies is less than a threshold. The final candidate reward function is the solution to the IRL task.

Formally, we formulate the IRL problem as an MDP without a reward function, denoted by $MDP\backslash R = \{S, A, T_a, \gamma\}$. Having t candidate policies $\pi_1, \ldots, \pi_t$, the next candidate reward is estimated by maximizing $\boldsymbol{d}^t$, the sum of the margins between value of expert's policy and each learned candidate policy. This is represented by an objective function which is as follows:

$$\text{maximize } \boldsymbol{d}^t = (\boldsymbol{v}^{\pi_E} - \boldsymbol{v}^{\pi_1}) + \cdots + (\boldsymbol{v}^{\pi_E} - \boldsymbol{v}^{\pi_t}) \qquad (8)$$

where $\boldsymbol{v}^{\pi}$ is the vector representation for value function: $\boldsymbol{v}^{\pi} = (\boldsymbol{v}^{\pi}(s_0), \ldots, \boldsymbol{v}^{\pi}(s_{|S|-1}))$, and $\boldsymbol{v}^{\pi}(s_i)$ is the value of state s_i under policy π, which can be drawn from Eq. (2). That is, we have:

$$\boldsymbol{v}^{\pi} = \boldsymbol{r}^{\pi} + \gamma \boldsymbol{T}^{\pi} \boldsymbol{v}^{\pi} \qquad (9)$$

where $\boldsymbol{v}^{\pi}$ is a vector of size $|S|$ in which $\boldsymbol{v}^{\pi}(s) = V^{\pi}(s)$, $\boldsymbol{r}^{\pi}$ is a vector of size $|S|$ in which $\boldsymbol{r}^{\pi}(s) = R(s, \pi(s))$, $\boldsymbol{T}^{\pi}$ is the transition matrix for policy π, that is a matrix of size $|S| \times |S|$ in which $\boldsymbol{T}^{\pi}(s, s') = T(s, \pi(s), s')$.

Notice that in IRL it is assumed that the reward of any state s can be represented as the linear combination of some features of state s, such as a feature vector defined as: $\boldsymbol{\phi} = (\phi_1(s,a), \ldots, \phi_K(s,a))$ where K is the number of features and each feature $\phi_i(s,a)$ is a basis function for the reward function. Then the reward function can be shown as the multiplication of two vectors $\boldsymbol{\Phi}^{\pi}$ and $\boldsymbol{\alpha}$ as:

$$\boldsymbol{r}^{\pi} = \boldsymbol{\Phi}^{\pi} \boldsymbol{\alpha} \qquad (10)$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$ are feature weights, and $\boldsymbol{\Phi}^{\pi}$ is a matrix of size $|S| \times K$ consisting of state action features for policy π and defined as:

$$\boldsymbol{\Phi}^{\pi} = \begin{pmatrix} \boldsymbol{\phi}(s_0, \pi(s_0))^{\mathrm{T}} \\ \cdots \\ \boldsymbol{\phi}(s_{|S|-1}, \pi(s_{|S|-1}))^{\mathrm{T}} \end{pmatrix}$$

For the expert's policy π_E, the state action features become:

$$\boldsymbol{\Phi}^{\pi_E} = \begin{pmatrix} \boldsymbol{\phi}(s_0, \pi_E(s_0))^{\mathrm{T}} \\ \cdots \\ \boldsymbol{\phi}(s_{|S|-1}, \pi_E(s_{|S|-1}))^{\mathrm{T}} \end{pmatrix}$$

We can now manipulate Eq. (9):

$$\begin{aligned} \boldsymbol{v}^{\pi} &= \boldsymbol{r}^{\pi} + \gamma \boldsymbol{T}^{\pi} \boldsymbol{v}^{\pi} \\ \boldsymbol{v}^{\pi} - \gamma \boldsymbol{T}^{\pi} \boldsymbol{v}^{\pi} &= \boldsymbol{r}^{\pi} \\ (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi}) \boldsymbol{v}^{\pi} &= \boldsymbol{r}^{\pi} \\ \boldsymbol{v}^{\pi} &= (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{r}^{\pi} \end{aligned}$$

Therefore, from the last equality we have:

$$\boldsymbol{v}^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{r}^{\pi} \qquad (11)$$

Using Eq. (10) in Eq. (11), we have:

$$\boldsymbol{v}^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{\Phi}^{\pi} \boldsymbol{\alpha} = \boldsymbol{x}^{\pi} \boldsymbol{\alpha} \qquad (12)$$

where $\boldsymbol{x}^{\pi}$ is a matrix of size $|S| \times K$ defined as:

$$\boldsymbol{x}^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{\Phi}^{\pi} \qquad (13)$$

Equation (12) shows that the vector of values $\boldsymbol{v}^{\pi}$ can be represented as multiplication of the vector of feature weights $\boldsymbol{\alpha}$ and another vector $\boldsymbol{x}^{\pi}$ defined by Eq. (13).

Similar to Eq. (12), for the expert's policy π_E, we have:

$$\boldsymbol{v}^{\pi_E} = \boldsymbol{x}^{\pi_E} \boldsymbol{\alpha} \qquad (14)$$

where $\boldsymbol{x}^{\pi_E}$ is a matrix of size $|S| \times K$ defined as:

$$\boldsymbol{x}^{\pi_E} = (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi_E})^{-1} \boldsymbol{\Phi}^{\pi_E} \qquad (15)$$

and $\boldsymbol{T}^{\pi_E}$ is a $|S| \times |S|$ matrix where element $\boldsymbol{T}^{\pi_E}(s_i, s_j)$ is the probability of transition from s_i to s_j with expert action $\pi_E(s_i)$.

Therefore, both a candidate reward function and its subsequent candidate policy can be represented as multiplication of features and the feature weights $\boldsymbol{\alpha}$ [see Eqs. (10) and (12)]. This enables us to solve Eq. (8) as a linear program. Using Eq. (12) and Eq. (14) in Eq. (8), we have:

$$\max_{\boldsymbol{\alpha}} \left[\left((\boldsymbol{x}^{\pi_E} - \boldsymbol{x}^{\pi_1}) + \cdots + (\boldsymbol{x}^{\pi_E} - \boldsymbol{x}^{\pi_t}) \right) \boldsymbol{\alpha} \right]$$
$$\text{subject to} \quad -1 \le \alpha_i \le +1 \; \forall i, 1 \le i \le K \tag{16}$$

Having t candidate policies $\pi_1, \ldots, \pi_t$, IRL estimates the next candidate reward by solving the above linear program. That is, IRL learns a new $\boldsymbol{\alpha}$ which represents a new candidate reward function, $\boldsymbol{r} = \boldsymbol{\Phi}^{\pi_E}\boldsymbol{\alpha}$. This new candidate reward has an "optimal policy" which is the new candidate policy π.

Algorithm 1 shows the MDP-IRL algorithm introduced by Ng and Russell (2000). This algorithm tries to find the expert reward function given an underlying MDP framework. The idea of this algorithm is that the value of expert's policy is required to be higher than the value of any other policy under the same MDP framework. This is reflected by the maximization in line 7 of the algorithm where $\boldsymbol{v}^{\pi_E} = \boldsymbol{x}^{\pi_E}\boldsymbol{\alpha}$ and $\boldsymbol{v}^{\pi_l} = \boldsymbol{x}^{\pi_l}\boldsymbol{\alpha}$ are the value of expert's policy and the value of candidate policy π_l, respectively. Notice that this algorithm maximizes the *sum* of the margins between the value of expert's policy π_E and the value of other candidate policies π_l, and this is reflected in line 7.

The algorithm starts by randomly initiating values for $\boldsymbol{\alpha}$ to generate the initial candidate reward function R^1 in line 1. Then, using dynamic programming for the MDP with the candidate reward function R^1, the algorithm finds policy of R^1, denoted by π_1 in line 2. In line 3, π_1 is used to construct $\boldsymbol{T}^{\pi_1}$ which is used to calculate $\boldsymbol{x}^{\pi_1}$ from Eq. (13). Then, in line 4, expert's policy π_E is used to construct $\boldsymbol{T}^{\pi_E}$ which is used to calculate $\boldsymbol{x}^{\pi_E}$ from Eq. (15).

From line 5 to line 16, MDP-IRL goes through the iterations to learn expert reward function by solving the linear program in line 7 with the constraints in line 8. For instance, in the first iteration of MDP-IRL, using the linear programming above, the algorithm finds $\boldsymbol{\alpha}$ which maximizes Eq. (16). In line 9, the learned vector values, $\boldsymbol{\alpha}$, make a candidate reward function R^2 which introduces a candidate policy π_2 in line 14. Then, in line 15, $\boldsymbol{T}^{\pi_2}$ is constructed for finding $\boldsymbol{x}^{\pi_2}$ from Eq. (13). The algorithm returns to line 5 to repeat the process of learning a new candidate reward until convergence. In this optimization, we also constrain the value of the experts's policy to be greater than that of other policies in order to ensure that the expert's policy is optimal.

Note that in Ng and Russell (2000) there is a slight different algorithm for when expert's policy is available in expert trajectories. The objective function for learning the reward function of expert maximizes sum of the margin between value of expert's policy and that of other policies using a monotonic function f. That is, the objective function in Ng and Russell (2000) is as follows:

Algorithm 1: MDP-IRL: inverse reinforcement learning in the MDP framework, adapted from Ng and Russell (2000)

Input: $MDP\backslash R = \{S, A, T, \gamma\}$, expert trajectories in the form of $D = \{(s_n, \pi_E(s_n), s'_n)\}$, a vector of features $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_K)$, convergence rate ϵ, and maximum iteration *maxT*
Output: Finds reward function R where $R = \sum_i \alpha_i \phi_i(s, a)$ by approximating $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$

```
1  Choose the initial reward $R^1$ by randomly initializing feature weights $\boldsymbol{\alpha}$;
2  Set $\Pi = \{\pi_1\}$ by finding $\pi_1$ using MDP with candidate reward function $R^1$ and value iteration;
3  Set $X = \{\boldsymbol{x}^{\pi_1}\}$ by calculating $\boldsymbol{x}^{\pi_1}$ using $\boldsymbol{T}^{\pi_1}$ and Equation (13);
4  Calculate $\boldsymbol{x}^{\pi_E}$ using $\boldsymbol{T}^{\pi_E}$ and Equation (15);
5  for $t = 1$ to maxT do
6      Find values for $\boldsymbol{\alpha}$ by solving the linear program:
7          maximize $\boldsymbol{d}^t = [((\boldsymbol{x}^{\pi_E} - \boldsymbol{x}^{\pi_1}) + \cdots + (\boldsymbol{x}^{\pi_E} - \boldsymbol{x}^{\pi_t}))\boldsymbol{\alpha}]$;
8          subject to $|\alpha_i| \le 1 \; \forall i \; 1 \le i \le K$;
9      $R^{t+1} = \sum_i \alpha_i^t \phi_i(s, a)$;
10     if $\max_i |\boldsymbol{\alpha}_i^t - \boldsymbol{\alpha}_i^{t-1}| \le \epsilon$ then
11         return $R^{t+1}$;
12     end
13     else
14         $\Pi = \Pi \cup \{\pi_{t+1}\}$ by finding $\pi_{t+1}$ using MDP with candidate reward function $R^{t+1}$ and value iteration;
15         Set $X = X \cup \{\boldsymbol{x}^{\pi_{t+1}}\}$ by calculating $\boldsymbol{x}^{\pi_{t+1}}$ using $\boldsymbol{T}^{\pi_{t+1}}$ and Equation (13);
16     end
17 end
```

$$\max \boldsymbol{d}^t = \left[f(\boldsymbol{v}^{\pi_E} - \boldsymbol{v}^{\pi_1}) + \cdots + f(\boldsymbol{v}^{\pi_E} - \boldsymbol{v}^{\pi_t}) \right]$$
$$\text{subject to} \quad |\alpha_i| \le 1 \; \forall i \; 1 \le i \le K \tag{17}$$

where Ng and Russell (2000) set $f(x) = x$ if $f(x) > 0$, otherwise, $f(x) = 2x$ to penalize the cases in which the value of expert's policy is less than the candidate policy. The authors, selected 2 in $f(x) = 2x$ since it had the least sensitivity in their experiments. The maximization in Eq. (16) is similar to the one in Eq. (17), particularly when $f(x) = x$ for all x.

Moreover, Ng and Russell (2000) suggested to approximate the policy values using Monte Carlo estimator. Recall the definition of value function in MDPs, shown in Eq. (7), defined as:

$$\begin{aligned} V^\pi(s) &= E_{s_t \sim T}\Big[\gamma^0 R(s_0, \pi(s_0)) + \gamma^1 R(s_1, \pi(s_1)) \\ &\qquad + \cdots) | \pi, s_0 = s \Big] \\ &= E_{s_t \sim T}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi, s_0 = s \right] \end{aligned}$$

Using M expert trajectory of size H, the value function in MDPs can be approximated using Monte Carlo estimator:

$$\hat{V}^{\pi}(s_0) = 1/M \sum_{m=1}^{M} \sum_{t=0}^{H-1} \gamma^t R(s,a) = 1/M \sum_{m=1}^{M} \sum_{t=0}^{H-1} \gamma^t \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{\phi}(s,a) \tag{18}$$

We then extended the trajectory-based MDP-IRL algorithm of Ng and Russell (2000) to a trajectory-based POMDP-IRL algorithm, called POMDP-IRL-BT, which is presented in the next section.

4 Extending MDP-IRL to the POMDP framework

In this section, we propose two IRL algorithms from expert trajectories in the POMDP framework. First in Sect. 4.1, we extend the MDP-IRL algorithm of Ng and Russell (2000) to POMDPs by approximating the value of expert's policy and that of candidate policies [respectively Eqs. (14) and (12)] for POMDPs. This is done by fixing the number of beliefs to the expert beliefs available in expert trajectories, and by approximating the expert belief transitions, i.e., the probability of transiting from one expert belief to another after performing an action. The algorithm is called POMDP-IRL-BT (BT for belief transitions). Then, in Sect. 4.2, we propose a point-based POMDP-IRL algorithm, called PB-POMDP-IRL.

4.1 POMDP-IRL-BT

We extend the trajectory-based MDP-IRL algorithm introduced in previous section to POMDPs. Our proposed algorithm, called POMDP-IRL-BT, considers the situation when expert trajectories are in form of $(a_1, o_1, \ldots, a_B, o_B)$, where B is the number of generated expert beliefs. Notice that by application of the state estimator function in Eq. (3), and an assumed belief b_0, we can calculate expert beliefs $(b_0, \ldots, b_{B-1})$. Thus, expert trajectories can be represented as $(b_0, \pi_E(b_0), \ldots, b_{B-1}, \pi_E(b_{B-1}))$.

The POMDP-IRL-BT algorithm is similar to the MDP-IRL algorithm, described in Sect. 3, but instead of states we use the finite number of expert beliefs that occurred in expert trajectories. Moreover, we approximate a belief transition for expert beliefs in the place of the transition model in MDPs. More specifically, we approximate the value of the expert's policy and the value of candidate policies by approximating Eqs. (12) and (14), respectively, for POMDPs. Therefore, in IRL for POMDPs we maximize the margin:

$$\boldsymbol{d}^t = (\boldsymbol{v}_b^{\pi_E} - \boldsymbol{v}_b^{\pi_1}) + \cdots + (\boldsymbol{v}_b^{\pi_E} - \boldsymbol{v}_b^{\pi_t})$$

where $\boldsymbol{v}_b^{\pi_E}$ is an approximation of the value of the expert's policy. This expert's policy is based on the expert beliefs that occurred in expert trajectories. Moreover, each $v_b^{\pi_t}$ is an approximation of value of the candidate policy π_t which is calculated by approximating expert belief transitions.

To illustrate these approximations, consider the value function for POMDPs shown in Eq. (6). Using the vector representation, we can rewrite this equation as follows:

$$\boldsymbol{v}_b^{\pi} = \boldsymbol{r}_b^{\pi} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{v}_b^{\pi} \tag{19}$$

where $\boldsymbol{v}_b^{\pi}$ is a vector of size B: the number of expert beliefs in which $\boldsymbol{v}_b^{\pi}(b) = V^{\pi}(b)$ [from Eq. (2)], $\boldsymbol{r}_b^{\pi}$ is a vector of size B in which $\boldsymbol{r}_b^{\pi}(b) = R(b, \pi(b))$, where $R(b,a)$ comes from Eq. (5), $\boldsymbol{P}^{\pi}$ is a matrix of size $B \times B$ that is the *belief transition* matrix for policy π, in which:

$$\boldsymbol{P}^{\pi}(b,b') = \sum_{o' \in O} \left[Pr(o'|b,\pi(b)) \, \mathit{ifClosest}((SE(b,\pi(b),o'),b') \right] \tag{20}$$

where SE is the state estimator function in Eq. (3) and *ifClosest*(b'', b') determines if b' is the closest expert belief to b'', the belief created as result of state estimator function.

Formally, we define *ifClosest*(b'', b') as:

$$\mathit{ifClosest}(b'',b') = \begin{cases} 1, & \text{if } \; b' = \arg\min_{b_n} |b'' - b_n| \\ 0, & \text{otherwise} \end{cases}$$

where $b_n \in B$ expert beliefs that appeared within the expert trajectories.

$P^{\pi}(b,b')$ is an approximate belief state transition model. It is approximated in three steps. First, the next belief b'' is estimated using the *SE* function. Second, the *ifClosest* function is used to find, b', the nearest belief that occurred within the expert trajectories. Finally, the transition probability between b and b' is updated using Eq. (20). This avoids handling the excessive number of new beliefs created by the *SE* function. More importantly, this procedure supports the use of IRL on a fixed number of beliefs, such as expert beliefs from a fixed number of trajectories.

After that, we construct the rest of formulations similar to MDPs. The reward function, R, is represented using the vector of features $\boldsymbol{\phi}$ so that each $\phi_i(s,a)$ is a basis function for the reward function. However, in POMDPs, we need to extend state features to beliefs. To do so, we define the vector $\phi(b,a)$ as: $\boldsymbol{\phi}(b,a) = \sum_{s \in S} b(s) \boldsymbol{\phi}(s,a)$. Then, matrix $\boldsymbol{\Phi}_b^{\pi}$ is an $N \times K$ matrix of belief action features for policy π, defined as:

$$\boldsymbol{\Phi}_b^{\pi} = \begin{pmatrix} \boldsymbol{\phi}(b_0, \pi(b_0))^{\mathrm{T}} \\ \ldots \\ \boldsymbol{\phi}(b_{B-1}, \pi(b_{B-1}))^{\mathrm{T}} \end{pmatrix}$$

For the expert's policy π_E, we define $\boldsymbol{\Phi}_b^{\pi_E}$ as:

$$\boldsymbol{\Phi}_b^{\pi_E} = \begin{pmatrix} \boldsymbol{\phi}(b_0, \pi_E(b_0))^{\mathrm{T}} \\ \dots \\ \boldsymbol{\phi}(b_{B-1}, \pi_E(b_{B-1}))^{\mathrm{T}} \end{pmatrix}$$

Then $\boldsymbol{r}_b^{\pi}$ is defined as:

$$\boldsymbol{r}_b^{\pi} = \boldsymbol{\Phi}_b^{\pi}\boldsymbol{\alpha} \tag{21}$$

Similar to the MDP-IRL, we can now manipulate Eq. (19):

$$\boldsymbol{v}_b^{\pi} = \boldsymbol{r}_b^{\pi} + \gamma \boldsymbol{P}^{\pi}\boldsymbol{v}_b^{\pi}$$
$$\boldsymbol{v}_b^{\pi} - \gamma \boldsymbol{P}^{\pi}\boldsymbol{v}_b^{\pi} = \boldsymbol{r}_b^{\pi}$$
$$(\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi})\boldsymbol{v}_b^{\pi} = \boldsymbol{r}_b^{\pi}$$

Therefore, from the last equality we have:

$$\boldsymbol{v}_b^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi})^{-1}\boldsymbol{r}_b^{\pi} \tag{22}$$

Using Eq. (21) in Eq. (22), we have:

$$\boldsymbol{v}_b^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi})^{-1}\boldsymbol{\Phi}_b^{\pi}\boldsymbol{\alpha}$$
$$\boldsymbol{v}_b^{\pi} = \boldsymbol{x}_b^{\pi}\boldsymbol{\alpha} \tag{23}$$

where $\boldsymbol{x}_b^{\pi}$ is a matrix of size $N \times K$ defined as:

$$\boldsymbol{x}_b^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi})^{-1}\boldsymbol{\Phi}_b^{\pi} \tag{24}$$

Equation (23) shows that the vector of values $\boldsymbol{v}_b^{\pi}$ can be represented as multiplication of the vector of feature weights $\boldsymbol{\alpha}$ and the vector $\boldsymbol{x}^{\pi_b}$. We have a similar equation for the expert's policy: $\boldsymbol{v}_b^{\pi_E} = \boldsymbol{x}_b^{\pi_E}\boldsymbol{\alpha}$, where $\boldsymbol{x}_b^{\pi_E}$ is a matrix of size $N \times K$ defined as:

$$\boldsymbol{x}_b^{\pi_E} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi_E})^{-1}\boldsymbol{\Phi}_b^{\pi_E} \tag{25}$$

where $\boldsymbol{P}^{\pi_E}$ is an $B \times B$ matrix where each element $\boldsymbol{P}^{\pi_E}(b_i, b_j)$ is the probability of transiting from b_i to b_j with expert action $\pi_E(b_i)$.

Algorithm 2 shows POMDP-IRL-BT and similarly to MDP-IRL, it maximizes the sum of the margins between the expert's policy π_E and the candidate policies π_t(line 7 of Algorithm 2). The POMDP-IRL-BT algorithm is based on the belief transition model, as opposed to MDP-IRL which is based on transition of completely observable states.

Let's now see Algorithm 2 in detail. It starts by randomly initiating values for $\boldsymbol{\alpha}$ to generate the initial candidate reward function R^1 in line 1. Then, it finds the policy of R^1, denoted by π_1 in line 2, using a model-based POMDP algorithm such as PBVI (Pineau et al. 2003). In line 3, it constructs $\boldsymbol{P}^{\pi_1}$ which is used to calculate $\boldsymbol{x}_b^{\pi_1}$ from Eq. (24). Then, in line 4, it uses π_E to construct $\boldsymbol{P}^{\pi_E}$ which is used to calculate $\boldsymbol{x}_b^{\pi_E}$ from Eq. (25). Finally, from line 5 to line 16, it iterates to learn the expert reward function by solving the linear program in line 7 with the constraints shown in line 8. The objective function of the linear program is:

Algorithm 2: POMDP-IRL-BT: inverse reinforcement learning in the POMDP framework using belief transitions estimation.

```
Input: $POMDP\backslash R = \{S, A, T, \gamma, O, \Omega, b_0\}$, expert
       trajectories in the form of $D = \{(b_n, \pi_E(b_n), b'_n)\}$, a
       vector of features $\boldsymbol{\phi} = (\phi_1, \dots, \phi_K)$,
convergence rate $\epsilon$, and maximum Iteration $maxT$
Output: Finds reward function $R$ where $R = \sum_i \alpha_i\phi_i(s,a)$ by
        approximating $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$
1  Choose the initial reward $R^1$ by randomly initializing feature
   weights $\boldsymbol{\alpha}$;
2  Set $\Pi = \{\pi_1\}$ by finding $\pi_1$ using POMDP with candidate
   reward function $R^1$ and a PBVI variant POMDP solver;
3  Set $X = \{\boldsymbol{x}_b^{\pi_1}\}$ by calculating $\boldsymbol{x}_b^{\pi_1}$ using $\boldsymbol{P}^{\pi_1}$ and Equation (24);
4  Calculate $\boldsymbol{x}_b^{\pi_E}$ from Equation (25);
5  for $t = 1$ to maxT do
6      Find values for $\boldsymbol{\alpha}$ by solving the linear program:
7      maximize$_\alpha$ $[((\boldsymbol{x}_b^{\pi_E} - \boldsymbol{x}_b^{\pi_1}) + \dots + (\boldsymbol{x}_b^{\pi_E} - \boldsymbol{x}_b^{\pi_t}))\boldsymbol{\alpha}]$;
8      subject to $-1 \le \alpha_i \le +1 \;\forall i\; 1 \le i \le K$;
9      $R^{t+1} = \sum_i \alpha_i^t \phi_i(s,a)$;
10     if $max_i|\boldsymbol{\alpha}_i^t - \boldsymbol{\alpha}_i^{t-1}| \le \epsilon$ then
11         return $R^{t+1}$;
12     end
13     else
14         $\Pi = \Pi \cup \{\pi_{t+1}\}$ by finding $\pi_{t+1}$ using POMDP with
           candidate reward function $R^{t+1}$ and a PBVI variant
           POMDP solver;
15         Set $X = X \cup \{\boldsymbol{x}_b^{\pi_{t+1}}\}$ by calculating $\boldsymbol{x}_b^{\pi_{t+1}}$ using $\boldsymbol{P}^{\pi_{t+1}}$
           and Equation (24);
16     end
17 end
```

$$\text{maximize } \boldsymbol{d}^t = \sum_{l=1}^{t}(\boldsymbol{x}_b^{\pi_E} - \boldsymbol{x}_b^{\pi_l})\boldsymbol{\alpha}$$

for all t candidate policies learned so far up to iteration t, subject to the constraints $|\alpha_i| \le 1 \;\forall i\; 1 \le i \le K$. Thus, it maximizes the sum of the margins between expert's policy π^E and other candidate policies π_l (we have t of them at iteration t). In this optimization, we also constrain the value of the expert's policy to be greater than that of other policies in order to ensure that the expert's policy is optimal.

This algorithm requires feature expectation matrix in each iteration, which requires inverting a matrix of size B, which is about $O(B^3)$. The algorithm also requires solving the linear program with $O(K)$ variables, and $O(K\ maxT)$ constraints. Moreover, it requires solving a POMDP model in each iteration that depens on the used solver.

4.2 PB-POMDP-IRL

Here, we propose a point-based IRL algorithm for POMDPs, called PB-POMDP-IRL. The idea in this algorithm is that the value of new beliefs, i.e., the beliefs that result of performing other policies than expert's policy, are approximated using expert beliefs. Moreover, this algorithm constructs a linear program for learning a reward function for the expert's policy by going through the expert trajectories and adding variables corresponding to the expert's policy value and variables corresponding to the alternative policy values.

To understand the algorithm, we start by a few definitions: we define each history $\boldsymbol{h}$ as a sequence of observation action pairs of the expert trajectories denoted by $\boldsymbol{h} = ((a_1, o_1), \ldots, (a_t, o_t))$. Moreover, we use $\boldsymbol{hao}$ for the history of size $|\boldsymbol{h}| + 1$ which includes the history $\boldsymbol{h}$ followed by (a, o). Then, we use $b_{\boldsymbol{h}}$ to show the belief at the end of history $\boldsymbol{h}$, which can be calculated using the State Estimator in Eq. (3). We now present State Estimator function with the new notations:

$$\begin{aligned} b_{\boldsymbol{hao}}(s') &= SE(b_{\boldsymbol{h}}, a, o) \\ &= Pr(s'|b_{\boldsymbol{h}}, a, o) \\ &= \eta \Omega(a, s', o) \sum_{s \in S} b_{\boldsymbol{h}}(s) T(s, a, s') \end{aligned}$$

where η is the normalization factor.

For instance, if $\boldsymbol{h} = (a_1, o_1)$ then the belief at the end of history $\boldsymbol{h}$, $b_{\boldsymbol{h}}$ is calculated by the belief update function in Eq. (3) and using (a_1, o_1) and b_0 (usually a uniform belief) as parameters. Similarly, if $\boldsymbol{h} = ((a_1, o_1), \ldots, (a_t, o_t,))$, the belief at the end of history is calculated by sequentially applying the belief update using (a_i, o_i) and b_{i-1} as parameters.

The PB-POMDP-IRL algorithm is described in Algorithm 3. In this algorithm, the value of new beliefs, i.e., the beliefs which result of performing other policies (than expert's policy), are approximated using expert beliefs. That is, given the belief $b_{\boldsymbol{hao}}$ where $a \neq \pi_E(b_{\boldsymbol{hao}})$, the value of $V^{\pi_E}(b_{\boldsymbol{hao}})$ is approximated using expert histories $\boldsymbol{h}'_i$ of the same size as $\boldsymbol{hao}$, i.e., $|\boldsymbol{h}'_i| = |\boldsymbol{hao}|$. This approximation is demonstrated in lines 16 and 17 of the algorithm:

$$V^{\pi_E}(b_{\boldsymbol{hao}}) = \sum_{i=0}^{n} w_i V(b_{\boldsymbol{h}'_i})$$

such that w_is follow $b_{\boldsymbol{hao}} = \sum_{i=0}^{n} w_i b_{\boldsymbol{h}'_i}$

Notice that due to the piecewise linearity of the optimal value function, this approximation corresponds to the true value if the expert's policy in the belief state $b_{\boldsymbol{hao}}$ is the same as the one in the belief states $b_{\boldsymbol{h}'_i}$, which is used in the linear combination. This condition is more likely to be true when the beliefs $b_{\boldsymbol{h}'_i}$ are closer to the approximated belief $b_{\boldsymbol{hao}}$.

The algorithm also constructs a linear program for learning the reward function by going through expert trajectories and adding variables corresponding to the expert's policy value and variables corresponding to alternative policy values. These variables are subject to the linear constraints that are subject to the Bellman equation (lines 22 and 27). In line 23, the linear constraint for the expert's policy value at end of history $\boldsymbol{h}$ is added. This constraint is based on the Bellman

Algorithm 3: Point-based POMDP-IRL: a point-based algorithm for IRL in the POMDP framework.

```
   Input: A POMDP\R as $(S, A, O, T, \Omega, b_0, \gamma)$, and a set $D$ of
          trajectories $a_1^m o_1^m \ldots a_{t-1}^m o_{t-1}^m a_t^m$, $t \leq H$ demonstrated
          by a human;
   Output: Reward weights $\alpha_i \in R$;
1  Extract the human's policy $\pi_E$ from the trajectories;
2  Initialize the set of variables $V$ with the weights $\alpha_i$;
3  Initialize the set of linear constraints $C$ with
   $\{\forall (s, a) \in S \times A : R_{min} \leq \sum_{i=1}^{k} \alpha_i \phi_i(s, a) \leq R_{max}\}$;
4  for $t = H$ to $1$ do
5      foreach $\boldsymbol{h} \in D$, such that $\boldsymbol{h}$ is a trajectory of length $t$, do
6          Calculate $b_{\boldsymbol{h}}$, the belief state at the end of trajectory $\boldsymbol{h}$;
7          foreach $(a, o) \in A \times O$ do
8              Add the variable $V^{\pi_E}(b_{\boldsymbol{hao}})$ to $V$;
9              // $V^{\pi_E}(b_{\boldsymbol{hao}})$ is an approximation of the value of $\pi^{\pi_E}$
               at $b_{\boldsymbol{hao}}$ defined below;
10             if $\boldsymbol{hao} \notin D$ and $t = H$ then
11                 Add the constraint $V^{\pi_E}(b_{\boldsymbol{hao}}) = 0$ to the set $C$ ;
12             end
13             if $\boldsymbol{hao} \notin D$ and $t < H$ then
14                 Let $b_{\boldsymbol{hao}}$ be the belief corresponding to the
                   trajectory $\boldsymbol{hao}$;
15                 Calculate the belief states $b_{\boldsymbol{h}'_i}$ corresponding to the
                   trajectories in $D$ of length $t + 1$ ;
16                 Find a list of weights $w_i$ such that
                   $b_{\boldsymbol{hao}} = \sum_{i=0}^{n} w_i b_{\boldsymbol{h}'_i}$ ;
17                 Add to $C$ the constraint
                   $V^{\pi_E}(b_{\boldsymbol{hao}}) = \sum_{i=0}^{n} w_i V(b_{\boldsymbol{h}'_i})$;
18                 // $V^{\pi_E}(b_{\boldsymbol{hao}})$ is an approximation of the value of
                   $\pi_E$ at the belief corresponding to the trajectory $\boldsymbol{hao}$
19             end
20         end
21         Add the variable $V^{\pi_E}(b_{\boldsymbol{h}})$ to $V$;
22         // $V^{\pi_E}(b_{\boldsymbol{h}})$ is the value of $\pi^{\pi_E}$ at $b_{\boldsymbol{h}}$;
23         Add to $C$ the constraint
           $V^{\pi_E}(b_{\boldsymbol{h}}) = [\sum_{s \in S} b_{\boldsymbol{h}}(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, \pi_E(b_{\boldsymbol{h}})) +$
           $\gamma \sum_{o \in O} Pr(o|b_{\boldsymbol{h}}, \pi_E(b_{\boldsymbol{h}})) V^{\pi_E}(b_{\boldsymbol{h}\pi_E(b_{\boldsymbol{h}})o})]$ ;
24         foreach $a \in A$ do
25             Add the variable $V^a(b_{\boldsymbol{h}})$ to $V$;
26             // $V^a(b_{\boldsymbol{h}})$ is the value of an alternative policy choosing
               $a$ after the trajectory $\boldsymbol{h}$;
27             Add to $C$ the constraint
               $V^a(b_{\boldsymbol{h}}) = \sum_{s \in S} b_{\boldsymbol{h}}(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, a) +$
               $\gamma \sum_{o \in O} Pr(o|b_{\boldsymbol{h}}, a) V^{\pi_E}(b_{\boldsymbol{hao}})$;
28             Add the variable $\epsilon_{\boldsymbol{h}}^a$ to the set $V$;
29             Add to $C$ the constraint $V^{\pi_E}(b_{\boldsymbol{h}}) - V^a(b_{\boldsymbol{h}}) \geq \epsilon_{\boldsymbol{h}}^a$;
30         end
31     end
32 end
33 maximize $\sum_{\boldsymbol{h} \in H} \sum_{a \in A} \epsilon_{\boldsymbol{h}}^a$  subject to the constraints of set $C$;
```

Eq. (6) which we present it again here:

$$V^{\pi}(b) = R(b, \pi(b)) + \gamma \sum_{o' \in O} Pr(o'|b, \pi(b)) V^{\pi}(b')$$

where here the rewards $R(b, \pi(b))$ are presented as linear combination of state features: $R(b, \pi(b)) = \sum_{i=1}^{k} \alpha_i \phi_i(s, a)$ and $R(b, a)$ is defined as $R(b, a) = \sum_{s \in S} b(s) R(s, a)$.

So, the value of expert's policy at end of history $\boldsymbol{h}$ becomes:

$$V^{\pi_E}(b_{\boldsymbol{h}}) = \left[\sum_{s\in S} b_{\boldsymbol{h}}(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, \pi^{\pi_E}(b_{\boldsymbol{h}})) + \gamma \sum_{o\in O} Pr(o|b_{\boldsymbol{h}}, \pi^{\pi_E}(b_{\boldsymbol{h}})) V^{\pi_E}(b_{\boldsymbol{h}\pi^{\pi_E}(b_{\boldsymbol{h}})o})\right]$$

Similarly, in line 27 the linear constraint for the alternative policy value at the end of history $\boldsymbol{h}$ is added. Notice that an alternative policy is a policy that selects an action $a \neq \pi^{\pi_E}(b_{\boldsymbol{h}})$ and then follows the expert's policy for the upcoming time-steps. This constraint is also based on the Bellman Eq. (6). That is, the value of performing action a at the belief $b_{\boldsymbol{h}}$ where $a \neq \pi^{\pi_E}(b_{\boldsymbol{h}})$ and then following expert's policy π_E becomes:

$$V^{a}(b_{\boldsymbol{h}}) = \sum_{s\in S} b_{\boldsymbol{h}}(s) \sum_{i=1}^{k} \alpha_i \phi_i(s, a) + \gamma \sum_{o\in O} Pr(o|b_{\boldsymbol{h}}, a) V^{\pi_E}(b_{\boldsymbol{h}ao})$$

Finally, in line 29 we explicitly state that the expert's policy value at any history $\boldsymbol{h}$, $V^{\pi_E}(b_{\boldsymbol{h}})$ is higher than any alternative policy value, $V^{a}(b_{\boldsymbol{h}})$ where $a \neq \pi_E(b_{\boldsymbol{h}})$, by a margin $\epsilon^{a}_{\boldsymbol{h}}$ that should be maximized in line 33.

This algorithm requires solving one linear program with $O(|A||O|MH)$ variables, and $O(|A|MH)$ constraints. Furthermore, the computational complexity of each linear approximation is $O(M^3H^3)$, given that there are $O(MH)$ belief states where the human's policy is known. Therefore, the overall complexity of the algorithm is polynomial in the parameters $|A|$, $|O|$, M and H.

5 Experiments

5.1 Experiments with SACTI-1

We applied our methods on the dialogue POMDP learned from SACTI-1 (Chinaei and Chaib-draa 2011). Williams and Young (2005) collected SACTI-1 dialogues, publicly available at: http://mi.eng.cam.ac.uk/projects/sacti/corpora/. SACTI stands for simulated ASR channel tourist information. It contains 144 dialogues between 36 users and 12 experts who play the role of the machine for 24 total tasks on this data set. The utterances are first recognized using a speech recognition error simulator, and then are sent to human experts for a response. There are four levels of ASR noise in SACTI-1 data: *none*, *low*, *medium*, and *high* noise. There is a total of 2,048 utterances that we used for our experiments which have 817 distinct words. Our learned dialogue POMDP model, learned from SACTI-1, consists of 5 states, 3 non-terminal states for *visits*, *transports*, and *foods* intentions, as well as two terminal states *success* and *failure*. It also includes 14 actions, 5 intention observations, and the learned transition and observation models. The SACTI-1 specifications for IRL experiments are described in Table 1.

Table 1 The SACTI-1 specifications for IRL experiments

Problem	$\|S\|$	$\|A\|$	$\|O\|$	γ	$\|\boldsymbol{\phi}\|$	$\|trajectories\|$
SACTI-1	5	14	5	0.90	70	50

Table 2 Number of matches for hand-crafted (HC) reward POMDPs, and learned reward POMDPs, w.r.t. 1,415 human expert actions

Noise level	None (%)	Low (%)	Med (%)	High (%)
HC	339–24	327–23	375–26	669–47
Learned	869–61	869–61	408–28	387–27

5.1.1 PB-POMDP-IRL evaluation on human policy

In our previous work (Boularias et al. 2010), we evaluated the PB-POMDP-IRL performance as the ASR noise level increases. The results are shown in Table 2. We applied the algorithm on four dialogue POMDPs learned from SACTI-1 dialogues with four levels of noise *none*, *low*, *medium*, and *high*, respectively. Our experimental results showed that the PB-POMDP-IRL algorithm is able to learn a reward function for *human* expert's policy. Note that SACTI dialogues have been collected in a Wizard-of-Oz setting. The results also show that the algorithm performs better in the lower noise levels (*none* and *low*) than in higher noise levels (*medium* and *high*). In Sect. 5, we compare the PB-POMDP-IRL algorithm to the POMDP-IRL-BT algorithm on POMDP policies.

5.1.2 Evaluations on POMDP policies

We then assume that the expert's policy is a POMDP policy. For the expert reward function, we assumed that it is as follows: (i) any action in non-terminal states receives −1 as reward; (ii) any action in the *success* terminal state receives +50 as reward; (iii) any action in the *failure* terminal state receives −F50 as reward. Then, we solved the POMDP model to find the optimal policy and assumed it as the expert's policy to generate 10 trajectories. Each trajectory is generated from the initial belief and by performing the expert action. After receiving an observation the expert belief is updated and the next action is performed. The trajectory ends when reaching one of the two terminal states. The 10 generated trajectories were then used in our two fold cross validation experiments.

Table 3 POMDP-IRL-BT and PB-POMDP-IRL results on the learned POMDP from SACTI-1: Number of matched actions to the expert actions

Algorithm	# of matched actions	Matched (%)
POMDP-IRL-BT	42	84
PB-POMDP-IRL	29	58

We applied the POMDP-IRL-BT and PB-POMDP-IRL algorithms on the SACTI-1 dialogue POMDP using *state-action-wise* features in which there is an indicator function for each state-action pair. Since there are 5 states and 14 actions in the example dialogue POMDP, the size of features equals $70 = 5 \times 14$. To solve each POMDP model, we used the Perseus solver which is a point-based value iteration (PBVI) solver (Spaan and Vlassis 2005). PBVI solvers are approximate solvers that use a finite number of beliefs for solving a POMDP model. We set the solver to use 10,000 random samples for solving the optimal policy of each candidate reward. The other parameter is max-time for execution of the algorithm, which is set to 1,000.

The twofold cross validation experiments are done as follows. We randomly selected 5 trajectories from the 10 expert trajectories, introduced above, for training and the rest of 5 trajectories for testing. Then we tested POMDP-IRL-BT and PB-POMDP-IRL. For each algorithm experiment, the algorithm was used to learn a reward function for the expert trajectories using the training trajectories. Then the learned policy, i.e., the policy of the learned reward function, was applied on the testing trajectories. Finally, we calculated the number of learned actions that matched to the expert actions on the testing trajectories, and they were added up for the two folds to make the cross validation experiments complete.

The experimental results are shown in Table 3. The result of the experiments showed that POMDP-IRL-BT significantly outperforms PB-POMDP-IRL on SACTI-1 expert's policy (where expert's policy is a POMDP policy). More specifically, the POMDP-IRL-BT algorithm was able to learn a reward function that matched with 42 actions out of 50 actions in the data set. That is, the policy of the learned reward function was equal to the expert's policy for 84 % of the beliefs. On the other hand, the learned policy using PB-POMDP-IRL matched to 29 actions out of the 50 actions in the data set. That is, the learned policy using the PB-POMDP-IRL algorithm matched to the expert's policy for only 58 % of the beliefs. As POMDP-IRL-BT outperforms PB-POMDP-IRL, we adopt it for learning a dialogue POMDP solely from unannotated and noisy dialogues, collected by SmartWheeler.

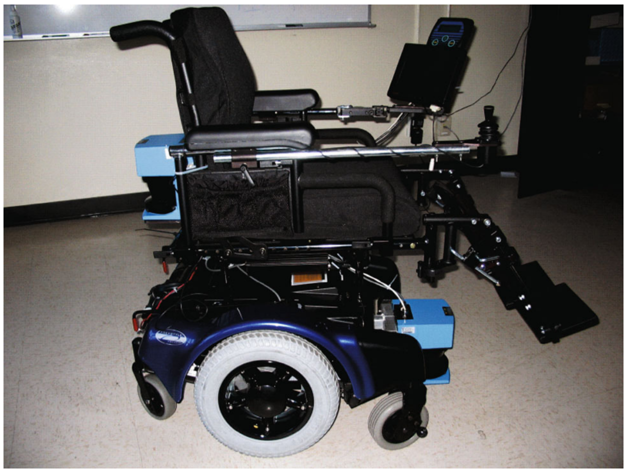
Fig. 2 The SmartWheeler robot platform

5.2 Experiments with SmartWheeler

5.2.1 SmartWheeler dialogues and choice of features

The SmartWheeler project (Pineau et al. 2011) aims to build an intelligent wheelchair for persons with disabilities. This project has been initiated in 2006, and a first prototype, shown in Fig. 2, was built in-house at McGill's Center for Intelligent Machines. Its objective aims to minimize the physical and cognitive load required in steering it. This is achieved via a spoken dialogue system where elderly can ask the SmartWheller to achieve some commands.

SmartWheeler dialogues We used the dialogues collected by SmartWheeler to develop a dialogue POMDP learned primarily from data. The data include eight dialogues with healthy users and nine dialogues with target users of SmartWheeler (Pineau et al. 2011). The dialogues with target users, who are the elderly, are somehow more noisy than the ones with healthy users. More specifically, the average word error rate (WER) is 13.9 % for the healthy user dialogues and 18.5 % for the target user dialogues. In order to perform our experiments on a larger amount of data, we used all the healthy and target user dialogues. In total, there are 2,853 user utterances and 422 distinct words in the SmartWheeler dialogues.

Table 4 shows a sample of SmartWheeler dialogues captured for training the dialogue POMDP model components. The first line denoted by u_1 shows the true user utterance, that is the one which has been extracted manually from user audio recordings. The following line denoted by $\tilde{u}_1$ is the recognized user utterances by ASR. Finally, the last line denoted by a_1 shows the performed action in response to the ASR output at the time of collecting the dialogues.

Table 4 A sample from the SmartWheeler dialogues Pineau et al. (2011)

	…
u_1 :	turn right
$\tilde{u}_1$:	[turn right]
a_1 :	TURN RIGHT
u_2 :	turn right a little
$\tilde{u}_2$:	[10 writer little]
a_2 :	PLEASE REPEAT YOUR COMMAND
u_3 :	turn right a little
$\tilde{u}_3$:	[turn right to lead a]
a_3 :	TURN RIGHT A LITTLE
u_4 :	stop
$\tilde{u}_4$:	[stop]
a_4 :	STOP
	…

Notice that the true user utterance is not observable to SmartWheeler, and thus it requires to perform the action based on the recognized utterance by ASR. That is, for each dialogue utterance recognized by ASR, the machine aims to estimate the user intention and then perform the best action that satisfies the user intention. The recognized utterance by ASR is not however reliable for decision making. For instance, the first utterance,

u_1 : [turn right a little],

shows the true user utterance. The ASR output for this utterance is,

$\tilde{u}_1$: [10 writer little].

As such, the action performed by SmartWheeler at this dialogue turn is, the *general query* action

u_1 : PLEASE REPEAT YOUR COMMAND.

The query action, is the SmartWheeler action for getting more information. For instance, in the example in Table 4, when SmartWheeler receives the second ASR output [10 writer little], it performs a general query action to get more information before it performs the right action for the user intention, i.e.,TURN RIGHT A LITTLE.

The list of all SmartWheeler actions are shown in Table 5. Each action is the right action of one state (the user intention for a specific command). So, ideally, there should be 24 states for SmartWheeler dialogues (There are 24 actions other than the general query action). However, in the next subsection we see that we only learned 11 of the states, mainly because of number of dialogues. That is, not all of the states appeared in the data frequently enough. There are also states that do not appear in dialogues at all.

Table 5 The list of the possible actions, performed by SmartWheeler

a_1	DRIVE FORWARD A LITTLE
a_2	DRIVE BACKWARD A LITTLE
a_3	TURN RIGHT A LITTLE
a_4	TURN LEFT A LITTLE
a_5	FOLLOW THE LEFT WALL
a_6	FOLLOW THE RIGHT WALL
a_7	TURN RIGHT DEGREE
a_8	GO THROUGH THE DOOR
a_9	SET SPEED TO MEDIUM
a_{10}	FOLLOW THE WALL
a_{11}	STOP
a_{12}	TURN LEFT
a_{13}	DRIVE FORWARD
a_{14}	APPROACH THE DOOR
a_{15}	DRIVE BACKWARD
a_{16}	SET SPEED TO SLOW
a_{17}	MOVE ON SLOPE
a_{18}	TURN AROUND
a_{19}	PARK TO THE RIGHT
a_{20}	TURN RIGHT
a_{21}	DRIVE FORWARD METER
a_{22}	PARK TO THE LEFT
a_{23}	TURN LEFT DEGREE
a_{24}	PLEASE REPEAT YOUR COMMAND

Choice of features Recall from the previous section that IRL needs features to represent the reward function. We propose *keyword* features for applying IRL on the learned POMDP dialogue from SmartWheeler. The keyword features are SmartWheeler keywords, i.e., 1-top words for each user intention that have been automatically learned during learning the user intentions (Chinaei et al. 2012). There are nine learned keywords:

forward, *backward*, *right*, *left*, *turn*, *go*, *for*, *top*, *stop*.

The keyword features for each state of SmartWheeler dialogue POMDP are represented in a vector, as shown in Table 6. The figure shows that states s_3, (*turn-right-little*) and s_6 (*follow-right-wall*) share the same features, i.e., *right*. Moreover, states s_4 (*turn-left-little*) and s_5 (*follow-left-wall*) share the same feature, i.e., *left*. In our experiments, we used *keyword-action-wise* features. Such features include the indicator functions for each pair of state-keyword and action. Thus, the feature size for SmartWheeler equals $216 = 9 \times 24$ (9 keywords and 24 actions).

Notice that the choice of features is application dependent. The reason for using keywords as state features is that in the intention-based dialogue applications the states are the dialogue intentions, where each intention is described as a

Table 6 Keyword features for the SmartWheeler dialogues

	Forward	Backward	Right	Left	Turn	Go	For	Top	Stop
s_1	1	0	0	0	0	0	0	0	0
s_2	0	1	0	0	0	0	0	0	0
s_3	0	0	**1**	0	0	0	0	0	0
s_4	0	0	0	**1**	0	0	0	0	0
s_5	0	0	0	**1**	0	0	0	0	0
s_6	0	0	**1**	0	0	0	0	0	0
s_7	0	0	0	0	1	0	0	0	0
s_8	0	0	0	0	0	1	0	0	0
s_9	0	0	0	0	0	0	1	0	0
s_{10}	0	0	0	0	0	0	0	1	0
s_{11}	0	0	0	0	0	0	0	0	1

Table 7 The assumed expert reward function for the dialogue POMDP learned from SmartWheeler dialogues

Assumed expert reward function													
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	…	REPEAT
s_1	1.0	0	0	0	0	0	0	0	0	0	0	0	…	0.4
s_2	0	1.0	0	0	0	0	0	0	0	0	0	0	…	0.4
s_3	0	0	1.0	0	0	0	0	0	0	0	0	0	…	0.4
s_4	0	0	0	1.0	0	0	0	0	0	0	0	0	…	0.4
s_5	0	0	0	0	1.0	0	0	0	0	0	0	0	…	0.4
s_6	0	0	0	0	0	1.0	0	0	0	0	0	0	…	0.4
s_7	0	0	0	0	0	0	1.0	0	0	0	0	0	…	0.4
s_8	0	0	0	0	0	0	0	1.0	0	0	0	0	…	0.4
s_9	0	0	0	0	0	0	0	0	1.0	0	0	0	…	0.4
s_{10}	0	0	0	0	0	0	0	0	0	1.0	0	0	…	0.4
s_{11}	0	0	0	0	0	0	0	0	0	0	1.0	0	…	0.4

vector of k-top words from the domain dialogues. Therefore, the keyword features are relevant features for the states.

5.2.2 *POMDP-IRL-BT evaluation with SmartWheeler*

We now show our experimental results using the POMDP-IRL-BT algorithm on the intention dialogue POMDP learned from SmartWheeler. As mentioned earlier, to evaluate the IRL algorithms, we consider that expert's policy is a POMDP policy using an assumed reward function. To do that, we assumed that the expert reward function is the one represented in Table 7 (top). For the choice of features, we also used the keyword features shown in Table 6.

Similar to the previous experiments using SACTI-1 dialogues, we performed two fold cross validation experiments by generating 10 expert trajectories. The expert trajectories are truncated after 20 steps, since there is no terminal state here. We then used the Perseus software with the same setting as described in Sect. 5.

Based on the specification above, we performed POMDP-IRL-BT on SmartWheeler expert trajectory for training. The experimental results showed that the policy of the learned reward was the same as the expert's policy for 194 beliefs inside the testing trajectory out of the 200 beliefs, i.e., 97 % matched actions. For all the 6 errors, the expert action was TURN RIGHT LITTLE, i.e., the right action for the state TURN- RIGHT- LITTLE, while the action of the learned reward suggested FOLLOW RIGHT WALL. However, this error did not happen in all the cases which the expert action was TURN RIGHT LITTLE in the testing trajectory.

Afterwards, we used state-action-wise features as defined previously where there are 11 states and 24 actions. In this case, the size of state-action-wise features equals $264 = 11 \times 24$. This is a slight increase compared to the size of keyword features, i.e., 216. We observed that in our experiment the learned policy is exactly the same as the expert's policy for the 200 beliefs inside the testing trajectory using state-action-wise features, i.e., 100 % matched with the expert's policy. In words, POMDP-IRL-BT was able to learn a reward function for the expert's policy using the learned dialogue POMDP from SmartWheeler dialogues.

5.2.3 Comparison of POMDP-IRL-BT to POMDP-IRL-MC

Choi and Kim (2011) proposed IRL algorithms in POMDP framework by assuming policies in the form of an FSC and thus using PBPI (point-based policy iteration) (Ji et al. 2007), as POMDP solver. In their algorithm, they used Monte Carlo estimator to estimate the value of expert's policy. We also implemented the Monte Carlo estimator as denoted by Eq. (18) for the estimation of policy values in line 7 in Algorithm 2, and used the Perseus software (Spaan and Vlassis 2005) as the POMDP solver. This new algorithm is called POMDP-IRL-MC.

We compared the two algorithms, POMDP-IRL-BT and POMDP-IRL-MC, based on the following criteria:

1. Percentage of the learned actions that matches to the expert actions.
2. Value of learned policy with respect to the value of expert policy.
3. CPU time spent by the algorithm as the number of expert trajectories (training data) increases.

Criteria 1 and 2 are used to evaluate the quality of the learned reward function for the expert. As in the previous experiment, the higher the matched actions, the better the learned reward function is. Similarly, criterion 2 compares the value of the learned reward function with the value of expert reward function. The higher the value of the learned policy, the better the learned reward function is. The results for these criteria is based on two fold cross validation using 400 expert trajectories, i.e., each fold contains of 200 expert trajectories.

Note that the value of learned policy (in criterion 2) is the sampled value of the policy. This was done by running the policy starting from a uniform belief to the maximum $maxT = 20$ time step or until it reaches the terminal state. The sampled values are averaged over 100 runs, and are calculated using:

$$\hat{V}^{\pi}(b) = \left[\sum_{t=0}^{maxT} \gamma^t R(b_t, \pi(b_t)) | \pi, b_0 = b\right]$$

Finally, criterion 3 evaluates the CPU time spent by the algorithm as the number of expert trajectories increases. This is to verify which of the two algorithms, POMDP-IRL-BT and POMDP-IRL-MC, requires more computation time. Below, we report on our experiments on SmartWheeler domain based on the above mentioned criteria.

Evaluation of the quality of the learned rewards First, we evaluated POMDP-IRL-BT and POMDP-IRL-MC using keyword features based on criteria 1 and 2. The results are shown in Fig. 3 (top) and (bottom). The two figures show consistent results in which the performance of POMDP-IRL-BT and POMDP-IRL-MC are comparable.

Figure 3 (top) shows percentage of the matched actions to those of expert, as the number of iterations increases (the first criteria). The figure demonstrates that after around 15 iterations the learned actions for 95 % of testing trajectories matches to actions suggested by the expert's policy, in both the POMDP-IRL-BT and POMDP-IRL-MC algorithms. The figure also shows that after iteration 15, percentage of the matched actions fluctuates slightly as the number of iterations increases, however percentage remains above 90 %.

Moreover, Fig. 3 (bottom) plots the value of the learned policy (the sampled value) as the number of iterations increases (criterion 2). Similar to Fig. 3 (top), we observe that for both POMDP-IRL-BT and POMDP-IRL-MC after iteration 15 the learned policy value becomes close to the expert's policy value. Moreover, though the learned policy values fluctuate slightly, it remains close to the expert's policy value after iteration 15.

The reason for these fluctuations could be the choice of features. In the experiments reported above we used the automatically learned keyword features for our POMDP-IRL experiments. In Table 6, we saw that the states 3 and 6 share the same feature *right*. Similarly, the states 4 and 5 share

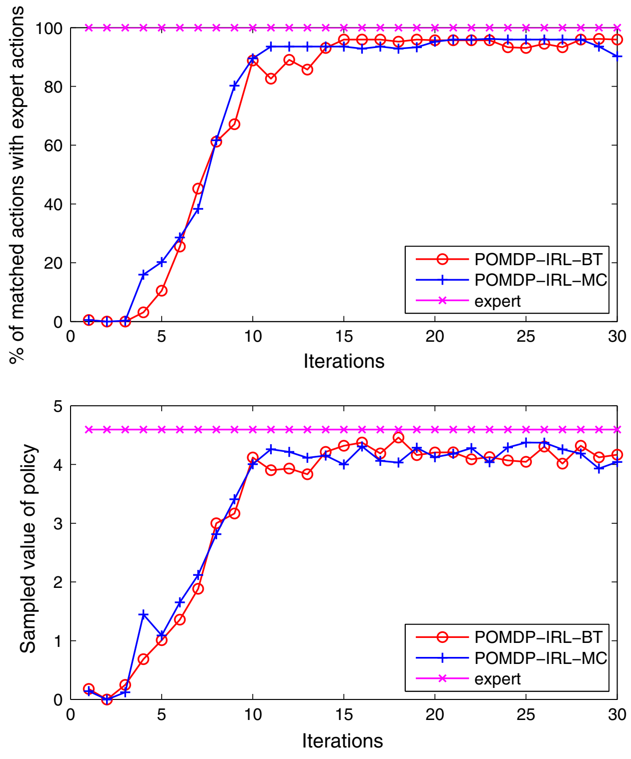

Fig. 3 Comparison of the POMDP-IRL algorithms using keyword features on the dialogue POMDP policy learned from SmartWheeler. *Top* percentage of matched actions. *Bottom* sampled value of the learned policy

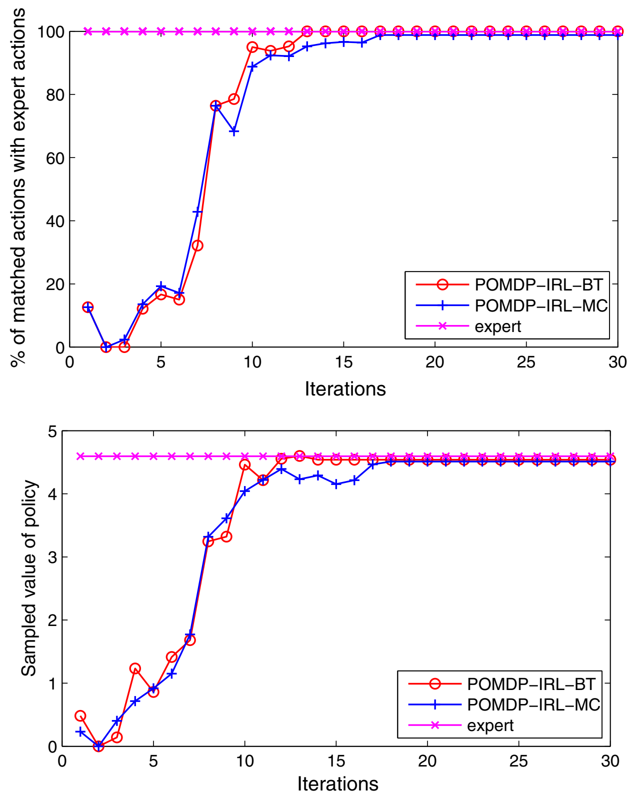

Fig. 4 Comparison of the POMDP-IRL algorithms using state-action-wise features on the dialogue POMDP learned from SmartWheeler. *Top* percentage of matched actions. *Bottom* sampled value of learned policy

the same feature *left*. Although this kind of feature sharing can reduce the size of features, it can lead to learning wrong actions for the sharing states.

Therefore, we performed similar experiments on SmartWheeler but this time using state-action features. These features include the indicator functions for each pair of state and action. Thus, the feature size for SmartWheeler equals $11 \times 24 = 264$, which is a slight increase compared to the size of keyword features, i.e., 216. Similar to the keyword features, we evaluated state-action features on SmartWheeler based on criteria 1 and 2. The results are shown in Fig. 4 (top) and Fig. 4 (bottom).

Figure 4 (top) and (bottom) show consistent results in which the performance of POMDP-IRL-BT reaches to expert performance. Figure 4 (top) shows percentage of the matched actions between the learned and expert policies, as the number of iterations increases. The figure shows that this percentage reaches to 100 % in POMDP-IRL-BT, while it reaches to 97 % in POMDP-IRL-MC.

Moreover, Fig. 4 (bottom) plots the value of the learned policy as the number of iterations increases. We observe that the learned value equals the value of expert's policy in POMDP-IRL-BT (at iteration 13), while in POMDP-IRL-MC it only gets close to the value of expert's policy (at iteration 17).

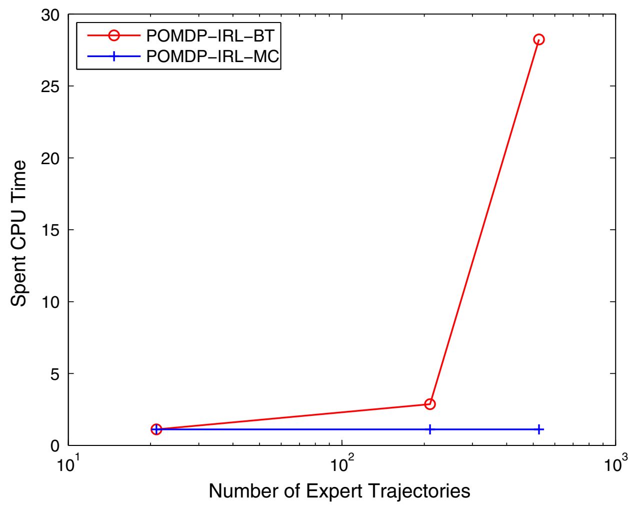

Fig. 5 Spent CPU time by POMDP-IRL algorithms on SmartWheeler, as the number of expert trajectories (training data) increases

Evaluation of the spent CPU time Figure 5 demonstrates the spent time by POMDP-IRL-BT and POMDP-IRL-MC as the number of expert trajectories (training data) increases. The results show that by increasing the number of expert trajectories, POMDP-IRL-BT requires considerably more time than POMDP-IRL-MC. Note that the figure plots the spent time by the number of trajectories in the logarithm base. This increase is due to increase of the size of belief transition matrix, [as reflected by Eq. (20)], as the number of expert trajectories increases. In other words, the belief transition matrix requires much more time to be constructed as the number of beliefs in expert trajectories increases. Also, note that this matrix is constructed for each candidate policy, which in turn increases the CPU time.

In sum, our experimental results showed that using state-action features, the POMDP-IRL-BT is able to learn a reward function in which the policy matches the expert's policy for 100 % of beliefs in the testing trajectories, while POMDP-IRL-MC learned a reward function in which the policy matched the expert's policy for only 97 % of beliefs in testing trajectories. However, POMDP-IRL-MC *does* scale substantially better than POMDP-IRL-BT. In the case of large number of expert trajectories, POMDP-IRL-BT can still be useful. For instance, we can use all expert trajectories to estimate the transition and observation models, but, select part of the expert trajectories to learn the reward model.

6 Conclusions and future work

In this paper, we proposed two POMDP-IRL algorithms: POMDP-IRL-BT and PB-POMDP-IRL. The first one approximates a belief transition model, which is similar to the state transition model in MDPs. On the other hand, PB-POMDP-IRL is a point-based POMDP-IRL algorithm that approximates the value of the new beliefs, which occurs in the computation of the policy values, using a linear approximation of expert beliefs. The two algorithms are able to learn a reward function that accounts for expert's policy. Our experimental results showed that POMDP-IRL-BT outperforms PB-POMDP-IRL, when expert's policy is a POMDP policy, using SACTI-1 learned POMDP.

To perform the IRL experiments, we introduced the automatically learned keyword features and then we evaluated the POMDP-IRL-BT algorithm on the learned intention POMDP from SmartWheeler. We observed that POMDP-IRL-BT is able to learn a reward function that accounts for the expert's policy using keyword-action-wise features.

Finally, we compared the POMDP-IRL-BT algorithm that uses belief transition estimation to the POMDP-IRL-MC algorithm that uses Monte Carlo estimation. Our experimental results showed that both algorithms are able to learn a reward function that accounts for the expert's policy. The experimental results showed that POMDP-IRL-BT slightly outperforms the POMDP-IRL-MC algorithm based on matched actions to the expert actions as well as the learned policy values. On the other hand, the POMDP-IRL-MC algorithm does scale much better than the POMDP-IRL-BT algorithm.

Overall, the experiments on SmartWheeler dialogues showed that the proposed methods are able to learn the dialogue POMDP model components from real dialogues.

In this work, we introduced the basic MDP-IRL algorithm of Ng and Russell (2000), and extended it for POMDPs. However, there are a vast number of IRL algorithms in the MDP framework (Abbeel and Ng 2004; Ramachandran and Amir 2007; Neu and Szepesvári 2007; Syed and Schapire 2008; Ziebart et al. 2008; Boularias et al. 2011). All these MDP-IRL algorithms can potentially be extended to POMDPs (Kim et al. 2011). In particular, Kim et al. (2011) extended the MDP-IRL algorithm of Abbeel and Ng (2004), called max-margin between feature expectations (MMFE), to a finite state controller (FSC) based POMDP-IRL algorithm. The authors showed that the extension of MMFE for POMDPs performs pretty well based on experiments on several POMDP benchmarks. The MMFE POMDP algorithm of Kim et al. (2011) also could be extended as a point-based POMDP-IRL algorithm in order to take advantage of the computational efficiency of point-based POMDP solvers such as Perseus. In addition, our proposed POMDP-IRL algorithms can be used for user simulation similar to MDP-IRL algorithms (Chandramohan et al. 2012).

Furthermore, the IRL algorithms requires (dialogue) features for representing the reward function. A relevant reward function to the dialogue system and users can be only learned by studying and extracting relevant features from the dialogue domain. Future research should be devoted on automatic methods for learning the relevant and proper features that are suitable for reward representation and reward function learning. We also observed that POMDP-IRL-BT algorithm does not scale as the number of trajectories increase. Although, the scalability may not be a great issue as the algorithm can learn the reward function of the expert using a small number of trajectories, another future avenue of research can be enhancing the scalability of the POMDP-IRL-BT algorithm and test the algorithm on larger dialogue domains.

References

Abbeel, P., Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine learning (ICML'04)*. Banff, AB, Canada.

Boularias, A., Chinaei, H. R., & Chaib-draa, B., (2010). Learning the reward model of dialogue POMDPs from data. In *NIPS 2010 Workshop on Machine Learning for Assistive Technologies*. Vancouver, BC, Canada.

Boularias, A., Kober, J., & Peters, J. (2011). Relative entropy inverse reinforcement learning. *Journal of Machine Learning Research—Proceedings Track*, *15*, 182–189.

Chandramohan, S., Geist, M., Lefèvre, F., & Pietquin, O. (2012). Behavior specific user simulation in spoken dialogue systems. In *Proceedings of the IEEE ITG Conference on Speech Communication*. Braunschweig, Germany.

Chinaei, H. R., & Chaib-draa, B. (2011). Learning dialogue POMDP models from data. In *Proceedings of the 24th Canadian Conference on Advances in Artificial Intelligence (Canadian AI'11)*. St. John's, NL, Canada.

Chinaei, H. R., & Chaib-draa, B. (2014). Dialogue POMDP components (Part I): Learning states and observations. *International Journal of Speech Technologyn* doi:10.1007/s10772-014-9244-6.

Chinaei, H. R., Chaib-draa, B., & Lamontagne, L. (2012). Learning observation models for dialogue POMDPs. In *Proceedings of the 24th Canadian conference on advances in Artificial Intelligence (Canadian AI'12)*. Toronto, ON, Canada.

Choi, J., & Kim, K.-E. (2011). Inverse reinforcement learning in partially observable environments. *Journal of Machine Learning Research*, *12*, 691–730.

Gašić, M. (2011). Statistical Dialogue Modelling. PhD thesis, Department of Engineering, University of Cambridge.

Ji, S., Parr, R., Li, H., Liao, X., & Carin, L. (2007). Point-based policy iteration. In *Proceedings of the 22nd National Conference on Artificial Intelligence* (vol. 2) (AAAI'07). Vancouver, BC, Canada.

Kim, D., Kim, J., & Kim, K. (2011). Robust performance evaluation of POMDP-based dialogue systems. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(4), 1029–1040.

Neu, G., Szepesvári, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI'07)*. Vancouver, BC, Canada.

Ng, A. Y., Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML'00)*. Stanford, CA, USA.

Paek, T., & Pieraccini, R. (2008). Automating spoken dialogue management design using machine learning: An industry perspective. *Speech Communication*, *50*(8), 716–729.

Pinault, F. and Lefèvre, F. (2011). Semantic graph clustering for pomdp-based spoken dialog systems. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association (INTERSPEECH'11)*. Florence, Italy.

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI'03)*. Acapulco, Mexico.

Pineau, J., West, R., Atrash, A., Villemure, J., & Routhier, F. (2011). On the feasibility of using a standardized test for evaluating a speech-controlled smart wheelchair. *International Journal of Intelligent Control and Systems*, *16*(2), 124–131.

Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*. Hyderabad, India.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (ACL'00)*. Hong Kong.

Spaan, M., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*(1), 195–220.

Syed, U. and Schapire, R. (2008). A game-theoretic approach to apprenticeship learning. In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*. Vancouver, BC, Canada.

Thomson, B. (2009). *Statistical Methods for Spoken Dialogue Management*. PhD thesis, Department of Engineering, University of Cambridge.

Williams, J. D. (2006). *Partially Observable Markov Decision Processes for Spoken Dialogue Management*. PhD thesis, Department of Engineering, University of Cambridge.

Williams, J. D., & Young, S. (2005). The SACTI-1 corpus: Guide for research users. Technical Report. Department of Engineering, University of Cambridge.

Williams, J. D., & Young, S. (2007). Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, *21*, 393–422.

Zhang, B., Cai, Q., Mao, J., Chang, E., & Guo, B. (2001a). Spoken dialogue management as planning and acting under uncertainty. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech'01)*. Aalborg, Denmark.

Zhang, B., Cai, Q., Mao, J., & Guo, B. (2001b). Planning and acting under uncertainty: A new model for spoken dialogue system. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI'01)*, Seattle, WA, USA.

Ziebart, B., Maas, A., Bagnell, J., & Dey, A. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08)*. Chicago, IL, USA.