Bayesian Nonparametric Inverse Reinforcement Learning

Bayesian Nonparametric Inverse Reinforcement Learning

Bernard Michini and Jonathan P. How

Massachusetts Institute of Technology,
Cambridge, Massachusetts, USA
{bmich,jhow}@mit.edu

Abstract. Inverse reinforcement learning (IRL) is the task of learning the reward function of a Markov Decision Process (MDP) given the transition function and a set of observed demonstrations in the form of state-action pairs. Current IRL algorithms attempt to find a single reward function which explains the entire observation set. In practice, this leads to a computationally-costly search over a large (typically infinite) space of complex reward functions. This paper proposes the notion that if the observations can be partitioned into smaller groups, a class of much simpler reward functions can be used to explain each group. The proposed method uses a Bayesian nonparametric mixture model to automatically partition the data and find a set of simple reward functions corresponding to each partition. The simple rewards are interpreted intuitively as subgoals, which can be used to predict actions or analyze which states are important to the demonstrator. Experimental results are given for simple examples showing comparable performance to other IRL algorithms in nominal situations. Moreover, the proposed method handles cyclic tasks (where the agent begins and ends in the same state) that would break existing algorithms without modification. Finally, the new algorithm has a fundamentally different structure than previous methods, making it more computationally efficient in a real-world learning scenario where the state space is large but the demonstration set is small.

1 Introduction

Many situations in artificial intelligence (and everyday life) involve learning a task from observed demonstrations. In robotics and autonomy, there exists a large body of literature on the topic of learning from demonstration (see [1] for a survey). However, much of the robotics work has focused on generating direct functional mappings for low-level tasks. Alternatively, one might consider assuming a rational model for the demonstrator, and using the observed data to invert the model. This process can be loosely termed *inverse decision making*, and in practice it is often more challenging (both conceptually and computationally) than more direct mapping approaches. However, inverting the decision-making process may lend more insight as to the motivation of the demonstrator, and

provide a richer explanation of the observed actions. Indeed, similar methodology has been increasingly used in psychology and cognitive science for action understanding and preference learning in humans [2, 3, 4, 5].

If the problem is formally cast in the Markov decision process (MDP) framework, the rational model described above becomes an agent who attempts to maximize cumulative reward (in a potentially sub-optimal fashion). Inverse decision making becomes the problem of finding a state reward function that explains the observed state-action pairs of the agent, and is termed *inverse reinforcement learning* (IRL) in the seminal work of [6].

There have since been a variety of IRL algorithms developed [7, 8, 9, 10, 11, 12, 13]. These algorithms attempt to find one single reward function that explains the entirety of the observed demonstration set. This reward function must then be necessarily complex in order to explain the data sufficiently, especially when the task being demonstrated is itself complicated. Searching for a complex reward function is fundamentally difficult for two reasons. First, as the complexity of the reward model increases, so too does the number of free parameters needed to describe the model. Thus the search is over a larger space of candidate functions. Second, the process of testing candidate reward functions requires solving for the MDP value function (details in Section 2), the computational cost of which typically scales poorly with the size of the MDP state space, even for approximate solutions [14]. Thus finding a single, complex reward function to explain the observed demonstrations requires searching over a large space of possible solutions and substantial computational effort to test each candidate.

One potential solution to these problems would be to partition the observations into sets of smaller sub-demonstrations. Then, each sub-demonstration could be attributed to a smaller and less-complex class of reward functions. However, such a method would require manual partitioning of the data into an unknown number of groups, and inferring the reward function corresponding to each group.

The primary contribution of this paper is to present an IRL algorithm that automates this partitioning process using Bayesian nonparametric methods. Instead of finding a single, complex reward function, the demonstrations are partitioned and each partition is explained with a simple reward function. We assume a generative model in which these simple reward functions can be interpreted as *subgoals* of the demonstrator. The generative model utilizes a Chinese Restaurant Process (CRP) prior over partitions so that the number of partitions (and thus subgoals) need not be specified *a priori* and can be potentially infinite.

As discussed further in Section 5, a key advantage of this method is that the reward functions representing each subgoal can be extremely simple. For instance, one can assume that a subgoal is a single coordinate of the state space (or feature space). The reward function could then consist of a single positive reward at that coordinate, and zero elsewhere. This greatly constrains the space of possible reward functions, yet complex demonstrations can still be explained using a sequence of these simple subgoals. Also, the algorithm has no dependence

on the sequential (i.e. temporal) properties of the demonstrations, instead focusing on partitioning the observed data by associated subgoal. Thus the resulting solution does not depend on the initial conditions of each demonstration, and moreover naturally handles cyclic tasks (where the agent begins and ends in the same state).

The paper proceeds as follows. Section 2 briefly covers preliminaries, and Section 3 describes the proposed algorithm. Section 4 presents experimental results comparing the proposed algorithm to existing IRL methods, and discussion is provided in Section 5.

2 Background

The following briefly reviews background material and notation necessary for the proposed algorithm. Throughout the paper, boldface is used to denote vectors subscripts are used to denote the elements of vectors (i.e. z_i is the ith element of vector $\boldsymbol{z}$).

2.1 Markov Decision Processes

A finite-state Markov Decision Process (MDP) is a tuple (S, A, T, γ, R) where S is a set of M *states*, A is a set of *actions*, $T : S \times A \times S \mapsto [0, 1]$ is the function of *transition probabilities* such that $T(s, a, s')$ is the probability of being in state s' after taking action a from state s, $R : S \mapsto \mathbb{R}$ is the *reward function*, and $\gamma \in [0, 1)$ is the *discount factor*.

A *stationary policy* is a function $\pi : S \mapsto A$. From [15] we have the following set of definitions and results:

1. The infinite-horizon expected reward for starting in state s and following policy π thereafter is given by the *value function* $V^\pi(s, R)$:

$$V^\pi(s, R) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R(s_i) \middle| s_0 = s \right] \tag{1}$$

   The value function satisfies the following Bellman equation for all $s \in S$:

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

   The so-called Q-function (or action-value function) $Q^\pi(s, a, R)$ is defined as the infinite-horizon expected reward for starting in state s, taking action a, and following policy π thereafter.
2. A policy π is optimal for M iff, for all $s \in S$:

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q^\pi(s, a, R) \tag{3}$$

   An optimal policy is denoted as π^* with corresponding value function V^* and action-value function Q^*.

2.2 Inverse Reinforcement Learning

When inverse decision making is formally cast in the MDP framework, the problem is referred to as *inverse reinforcement learning* (IRL)[6]. An MDP/R is defined as a MDP for which everything is specified except the state reward function $R(s)$. Observations (demonstrations) are provided as a set of state-action pairs:

$$\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), ..., (s_N, a_N)\} \tag{4}$$

where each pair $O_i = (s_i, a_i)$ indicates that the demonstrator took action a_i while in state s_i. Inverse reinforcement learning algorithms attempt to find a reward function that rationalizes the observed demonstrations. For example, find a reward function $\widehat{R}(s)$ whose corresponding optimal policy π^* matches the observations $\boldsymbol{O}$.

It is clear that the IRL problem is ill-posed. Indeed, $\widehat{R}(s) = c \ \ \forall s \in S$, where c is any constant, will make any set of state-action pairs O trivially optimal. Also, O may contain inconsistent or conflicting state-action pairs, i.e. (s_i, a_1) and (s_i, a_2) where $a_1 \neq a_2$. Furthermore, the "rationality" of the demonstrator is not well-defined (e.g., is the demonstrator perfectly optimal, and if not, to what extent sub-optimal).

Most existing IRL algorithms attempt to resolve the ill-posedness by making some assumptions about the form of the demonstrator's reward function. For example, in [7] it is assumed that the reward is a sum of weighted state features, and finds a reward function to match the demonstrator's feature expectations. In [8] a linear-in-features reward is also assumed, and a maximum margin optimization is used to find a reward function that minimizes a loss function between observed and predicted actions. In [9] it is posited that the demonstrator samples from a prior distribution over possible reward functions, and thus Bayesian inference is used to find a posterior over rewards given the observed data. An implicit assumption in these algorithms is that the demonstrator is using a single, fixed reward function.

The three IRL methods mentioned above (and other existing methods such as [10, 11, 13]) share a generic algorithmic form, which is given by Algorithm 1, where the various algorithms use differing definitions of "similar" in Step 2c. We note that each iteration of the algorithm requires re-solving for the optimal MDP value function in Step 2a, and the required number of iterations (and thus MDP solutions) is potentially unbounded.

2.3 Chinese Restaurant Process Mixtures

Since the proposed IRL algorithm seeks to partition the observed data, a Chinese restaurant process (CRP) is used to define a probability distribution over the space of possible partitions. The CRP proceeds as follows:

1. The first customer sits at the first table.
2. Customer i arrives and chooses the first unoccupied table with probability $\frac{\eta}{i-1+\eta}$, and an occupied table with probability $\frac{c}{i-1+\eta}$, where c is the number of customers already sitting at that table.

Algorithm 1: Generic inverse reinforcement learning algorithm.

`GenericIRL`(MDP/R, Observations $O_{1:N}$, Reward representation $\widehat{R}(s|w)$)

1. Initialize reward function parameters w^0
2. Iterate from $t = 1$ to T:
   (a) Solve for optimal MDP value function V^* corresponding to reward function $\widehat{R}(s|w^{(t-1)})$
   (b) Use V^* to define a policy $\widehat{\pi}$.
   (c) Choose parameters $w^{(T)}$ to make $\widehat{\pi}$ more similar to demonstrations $O_{1:N}$ in the next iteration.
3. `Return` Reward function given by $\widehat{R}(s|w^{(T)})$

The concentration hyperparameter η controls the probability that a customer starts a new table. Using $z_i = j$ to denote that customer i has chosen table j, C_j to denote the number of customers sitting at table j, and J_{i-1} to denote the number of tables currently occupied by the first $i-1$ customers, the assignment probability can be formally defined by:

$$P(z_i = j|z_{1\ldots i-1}) = \begin{cases} \frac{C_J}{i-1+\eta} & j \leq J_{i-1} \\ \frac{\eta}{i-1+\eta} & j = J_{i-1}+1 \end{cases} \tag{5}$$

This process induces a distribution over table partitions that is *exchangeable* [16], meaning that the order in which the customers arrive can be permuted and any partition with the same proportions will have the same probability. A Chinese restaurant process mixture is defined using the same construct, but each table is endowed with parameters θ of a probability distribution which generates data points x_i:

1. Each table j is endowed with parameter θ_j, where θ_j is drawn i.i.d. from a prior $P(\theta)$.
2. For each customer i that arrives:
   (a) The customer sits at table j according to (5) (the assignment variable $z_i = j$).
   (b) A datapoint x_i is drawn i.i.d. from $P(x|\theta_j)$.

Thus each datapoint x_i has an associated table assignment $z_i = j$ and is drawn from the distribution $P(x|\theta_j)$. Throughout the paper we use i to index state-action pairs O_i of the demonstrator ("customers" in the CRP analogy). We use j to index partitions of the state-actions pairs ("tables" in the CRP analogy). Finally, the table parameters θ_j in the CRP mixture model presented above correspond to the simple reward function for each partition, which we interpret as *subgoals* throughout the paper.

3 Bayesian Nonparametric IRL Algorithm

The following section describes the Bayesian nonparametric subgoal IRL algorithm. We start with two definitions necessary to the algorithm.

Definition 1 *A state subgoal g is simply a single coordinate $g \in S$ of the MDP state space. The associated state subgoal reward function $R_g(s)$ is:*

$$R_g(s) = \begin{cases} c \text{ at state } g \\ 0 \text{ at all other states} \end{cases} \tag{6}$$

where c is a positive constant.

While the notion of a state subgoal and its associated reward function may seem trivial, a more general *feature* subgoal will be defined in the following sections to extend the algorithm to a feature representation of the state space.

Definition 2 *An MDP agent in state s_i moving towards some state subgoal g chooses an action a_i with the following probability:*

$$P(a_i|s_i, g) = \pi(a_i|s_i, g) = \frac{e^{\alpha Q^*(s_i, a_i, R_g)}}{\sum_a e^{\alpha Q^*(s_i, a, R_g)}} \tag{7}$$

Thus π defines a stochastic policy as in [15], and is essentially our model of rationality for the demonstrating agent (this is the same rationality model as in [9] and [4]). In Bayesian terms, it defines the likelihood of observed action a_i when the agent is in state s_i. The hyperparameter α represents our degree of confidence in the demonstrator's ability to maximize reward.

3.1 Generative Model

The set of observed state-action pairs O defined by (4) are assumed to be generated by the following model. The model is based on the likelihood function above, but adds a CRP partitioning component. This addition reflects our basic assumption that the demonstrations can be explained by partitioning the data and finding a simple reward function for each partition.

An agent finds himself in state s_i (because of the Markov property, the agent need not consider how he got to s_i in order to decide which action a_i to take). In analogy to the CRP mixture described in Section 2.3, the agent chooses which partition a_i should be added to, where each existing partition j has its own associated subgoal g_j. The agent can also choose to assign a_i to a new partition whose subgoal will be drawn from the base distribution $P(g)$ of possible subgoals. The assignment variable z_i is set to denote that the agent has chosen partition z_i, and thus subgoal g_{z_i}. As in equation (5), $P(z_i|z_{1:i-1}) = CRP(\eta, z_{1:i-1})$. Now that a partition (and thus subgoal) has been selected for a_i, the agent generates the action according to the stochastic policy $a_i \sim \pi(a_i|s_i, g_{z_i})$ from equation (7).

The joint probability over $O_{1:N}$, $\boldsymbol{z}$, and $\boldsymbol{g}$ is given below, since it will be needed to derive the conditional distributions necessary for sampling:

$$\begin{aligned} P(O_{1:N}, \boldsymbol{z}, \boldsymbol{g}) &= P(O_{1:N}|\boldsymbol{z}, \boldsymbol{g})\ P(\boldsymbol{z}, \boldsymbol{g}) && (8) \\ &= P(O_{1:N}|\boldsymbol{z}, \boldsymbol{g})\ P(\boldsymbol{z})\ P(\boldsymbol{g}) && (9) \\ &= \prod_{i=1}^{N} \underbrace{P(O_i|g_{z_i})}_{\text{likelihood}} \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \prod_{j=1}^{J_N} \underbrace{P(g_j)}_{\text{prior}} && (10) \end{aligned}$$

where (9) follows since subgoal parameters g_j for each new partition are drawn independently from prior $P(g)$ as described above. As shown in (10), there are three key elements to the joint probability. The likelihood term is the probability of taking each action a_i from state s_i given the associated subgoal g_{z_i}, and is defined in (7). The CRP term is the probability of each partition assignment z_i given by (5). The prior term is the probability of each partition's subgoal (J_N is used to indicate the number of partitions after observing N datapoints). The subgoals are drawn i.i.d. from discrete base distribution $P(g)$ each time a new partition is started, and thus have non-zero probability given by $P(g_j)$.

The model assumes that O_i is conditionally independent of O_j for $i \neq j$ given g_{z_i}. Also, it can be verified that the CRP partition probabilities $P(z_i|z_{-i})$ are exchangeable. Thus, the model implies that the data O_i are exchangeable [16]. Note that this is weaker than implying that the data are independent and identically distributed (i.i.d.). The generative model instead assumes that there is an underlying grouping structure that can be exploited in order to decouple the data and make posterior inference feasible.

The CRP partitioning allows for an unknown and potentially infinite number of subgoals. By construction, the CRP has "built-in" complexity control, i.e. its concentration hyperparameter η can be used to make a smaller number of partitions more likely.

3.2 Inference

The generative model (10) has two sets of hidden parameters, namely the partition assignments z_i for each observation O_i, and the subgoals g_j for each partition j. Thus the job of the IRL algorithm will be to infer the posterior over these hidden variables, $P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$. While both $\boldsymbol{z}$ and $\boldsymbol{g}$ are discrete, the support of $P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$ is combinatorially large (since $\boldsymbol{z}$ ranges over the set of all possible partitions of N integers), so exact inference of the posterior is not feasible. Instead, approximate inference techniques must be used. Gibbs sampling [17] is in the family of Markov chain Monte Carlo (MCMC) sampling algorithms and is commonly used for approximate inference of Bayesian nonparametric mixture models [18, 19, 20]. Since we are interested in the posterior of both the assignments and subgoals, uncollapsed Gibbs sampling is used where both the $\boldsymbol{z}$ and $\boldsymbol{g}$ are sampled in each sweep.

Each Gibbs iteration involves sampling from the conditional distributions of each hidden variable given all of the other variables (i.e. sample one unknown at a time with all of the others fixed). Thus the conditionals for each partition assignment z_i and subgoal g_j must be derived.

The conditional for partition assignment z_i can be derived as follows:

$$
\begin{aligned}
P(z_i|z_{-i}, \boldsymbol{g}, \boldsymbol{O}) &\propto P(z_i, O_i \mid z_{-i}, O_{-i}) && (11)\\
&= P(z_i|z_{-i}, \boldsymbol{g}, O_{-i}) P(O_i|z_i, z_{-i}, \boldsymbol{g}, O_{-i}) && (12)\\
&= P(z_i|z_{-i}) \;\; P(O_i|z_i, z_{-i}, \boldsymbol{g}, O_{-i}) && (13)\\
&= \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \;\; \underbrace{P(O_i|g_{z_i})}_{\text{likelihood}} && (14)
\end{aligned}
$$

where (11) is the definition of conditional probability, (12) applies the chain rule, (13) follows from the fact that assignment z_i depends only on the other assignments z_{-i}, and (14) follows from the fact that each O_i depends only on its assigned subgoal g_{z_i}.

Algorithm 2: Bayesian nonparametric IRL.

BNIRL(MDP/R, Observations $O_{1:N}$, Confidence α, Concentration η)

1. **for each unique** $s_i \in O_{1:N}$:
   (a) Solve for and store $V^*(R_g)$, where $g = s_i$ and R_g is defined by (20)
   (b) Sample an initial subgoal $g_1^{(0)}$ from prior $P(g)$ and set all assignments $z_i^{(0)} = 1$
2. **for** $t = 1$ to T:
   (a) **for each current subgoal** $g_j^{(t-1)}$: Sample subgoal $g_j^{(t)}$ from (17)
   (b) **for each observation** $O_i \in O_{1:N}$: Sample assignment $z_i^{(t)}$ from (14)
3. **Return** samples $z^{(1:T)}$ and $g^{(1:T)}$, discarding samples for burn-in and lag if desired

When sampling from (14), the exchangeability of the data is utilized to treat z_i as if it was the last point to be added. Probabilities (14) are calculated with z_i being assigned to each existing partition, and for the case when z_i starts a new partition with subgoal drawn from the prior $P(g)$. While the number of partitions is potentially infinite, there will always be a finite number of groups when the length of the data N is finite, so this sampling step is always feasible.

The conditional for each partition's subgoal g_j is derived as follows:

$$P(g_j|\boldsymbol{z},\boldsymbol{O}) \propto P(O_{I_j}|g_j,\boldsymbol{z},O_{-I_j})P(g_j|\boldsymbol{z},O_{-I_j}) \tag{15}$$

$$= \sum_{i\in I_j} P(O_i|g_{z_i})\ P(g_j|\boldsymbol{z},O_{-I_j}) \tag{16}$$

$$= \sum_{i\in I_j} \underbrace{P(O_i|g_{z_i})}_{\text{likelihood}}\ \underbrace{P(g_j)}_{\text{prior}} \tag{17}$$

where (15) applies Bayes' rule, (16) follows from the fact that each O_i depends only on its assigned subgoal g_{z_i}, and (17) follows from the fact that the subgoal g_j of each partition is drawn i.i.d. from the prior over subgoals. The index set $I_j = \{i : z_i = j\}$.

Sampling from (17) depends on the form of the prior over subgoals $P(g)$. When the subgoals are assumed to take the form of *state subgoals* (Definition 1), then $P(g)$ is a discrete distribution whose support is the set S of all states of the MDP. In this paper, we propose the following simplifying assumption to increase the efficiency of the sampling process.

Proposition 1 *The prior $P(g)$ is assumed to have support only on the set S_O of MDP states, where $S_O = \{s \in S : s = s_i$ for some observation $O_i = (s_i, a_i)\}$.*

This proposition assumes that the set of all possible subgoals is limited to only those states encountered by the demonstrator. Intuitively it implies that during the demonstration, the demonstrator achieves each of his subgoals. This is not the same as assuming a perfect demonstrator (the expert is not assumed to get to each subgoal optimally, just eventually). Sampling of (17) now scales with the number of unique states in the observation set $O_{1:N}$. While this proposition may seem limiting, the experimental results in Section 4 indicate that it does not affect performance compared to other IRL algorithms and greatly reduces the required amount of computation. Algorithm 2 defines the proposed Bayesian nonparametric inverse reinforcement learning method. The algorithm outputs samples which form a Markov chain whose stationary distribution is the posterior, so that sampled assignments $\boldsymbol{z}^{(T)}$ and subgoals $\boldsymbol{g}^{(T)}$ converge to a

sample from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$ as $T \to \infty$ [17, 21]. Note that instead of solving for the MDP value function in each iteration (as is typical with IRL algorithms, see Algorithm 1), Algorithm 2 pre-computes all of the necessary value functions. The number of required value functions is upper bounded by the number of elements in the support of the prior $P(g)$. When we assume Proposition 1, then the support of $P(g)$ is limited to the set of unique states in the observations $O_{1:N}$. Thus the required number of MDP solutions scales with the size of the observed data set $O_{1:N}$, *not* with the number of required iterations. We see this as an advantage in a learning scenario when the size of the MDP is potentially large but the amount of demonstrated data is small.

3.3 Convergence in Expected 0-1 Loss

To demonstrate convergence, it is common in IRL to define a loss function which in some way measures the difference between the demonstrator and the predictive output of the algorithm [8, 9, 10]. In Bayesian nonparametric IRL, the assignments $\boldsymbol{z}$ and subgoals $\boldsymbol{g}$ represent the hidden variables of the demonstration that must be learned. Since these variables are discrete, a 0-1 loss function is suitable:

$$\mathcal{L}\left[(\boldsymbol{z}, \boldsymbol{g}), (\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})\right] = \begin{cases} 1 \text{ if } \ (\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}}) = (\boldsymbol{z}, \boldsymbol{g}) \\ 0 \text{ otherwise} \end{cases} \tag{18}$$

The loss function evaluates to 1 if the estimated parameters $(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})$ are exactly equal to the true parameters $(\boldsymbol{z}, \boldsymbol{g})$, and 0 otherwise. We would like to show that, for the Bayesian nonparametric IRL algorithm (Algorithm 2), the expected value of the loss function (18) given a set of observations $O_{1:N}$ is minimized as the number of iterations T increases. Theorem 1 establishes this.

Theorem 1 *Assuming observations $O_{1:N}$ are generated according to the generative model defined by (10), the expected 0–1 loss defined by (18) is minimized by the empirical mode of the samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$ output by Algorithm 2 as the number of iterations $T \to \infty$.*

Proof. It can be verified that the maximum *a posteriori* (MAP) parameter values, defined here by

$$(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}}) = \underset{(\boldsymbol{z}, \boldsymbol{g})}{\operatorname{argmax}} P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$$

minimize the expected 0–1 loss defined in (18) given the observations $O_{1:N}$ (see [22]). By construction, Algorithm 2 defines a Gibbs sampler whose samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$ converge to samples from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$ so long as the Markov chain producing the samples is ergodic [17]. From [23], a sufficient condition for ergodicity of the Markov chain in Gibbs sampling requires only that the conditional probabilities used to generate samples are non-zero. For Algorithm 2, these conditionals are defined by (14) and (17). Since clearly the likelihood (7) and CRP prior (5) are always non-zero, then the conditional (14) is always non-zero. Furthermore, the prior over subgoals $P(g)$ is non-zero for all possible g by assumption, so that (17) is non-zero as well.

Thus the Markov chain is ergodic and the samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$ converge to samples from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$ as $T \to \infty$. By the strong law of large numbers, the empirical mode of the samples, defined by

$$(\widetilde{\boldsymbol{z}}, \widetilde{\boldsymbol{g}}) = \underset{(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})}{\operatorname{argmax}} P(\boldsymbol{z}, \boldsymbol{g}|O_{1:N})$$

converges to the true mode $(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})$ as $T \to \infty$, and this is exactly the MAP estimate of the parameters which was shown to minimize the 0–1 loss. □

We note that, given the nature of the CRP prior, the posterior will be multimodal (switching partition indices does not affect the partition probability even though the numerical assignments $\boldsymbol{z}$ will be different). As such, the argmax above is used to define the *set* of parameter values which maximize the posterior. In practice, the sampler need only converge on one of these modes to find a satisfactory solution.

The rate at which the loss function decreases relies on the rate the empirical sample mode(s) converges to the true mode(s) of the posterior. This is a property of the approximate inference algorithm and, as such, is beyond the scope of this paper (convergence properties of the Gibbs sampler have been studied, for instance in [24]). As will be seen empirically in Section 4, the number of iterations required for convergence is typically similar to (or less than) that required for other IRL methods.

3.4 Action Prediction

IRL algorithms find reward models with the eventual goal of learning to predict what action the agent will take from a given state. As in Algorithm 1, the typical output of the IRL algorithm is a single reward function that can be used to define a policy which predicts what action the demonstrator would take from a given state.

In Bayesian nonparametric IRL (Algorithm 2), in order to predict action a_k from state s_k, a subgoal must first be chosen from the mode of the samples $\widehat{\boldsymbol{g}} = \text{mode}(\boldsymbol{g}^{(1:T)})$. This is done by finding the most likely partition assignment z_k after marginalizing over actions using Equation (11):

$$z_k = \underset{z_i}{\text{argmax}} \sum_a P\left(z_i \mid \widehat{z}_{-i}, \widehat{\boldsymbol{g}}, O_k = (s_k, a)\right) \tag{19}$$

where $\widehat{\boldsymbol{z}}$ is the mode of samples $\boldsymbol{z}^{(1:T)}$. Then, an action is selected using the policy defined by (7) with $\widehat{g}_{z_k}$ as the subgoal.

Alternatively, the subgoals can simply be used as waypoints which are followed in the same order as observed in the demonstrations. In addition to predicting actions, the subgoals in $\widehat{\boldsymbol{g}}$ can be used to analyze which states in the demonstrations are important, and which are just transitory.

3.5 Extension to Discrete Feature Spaces

Linear combinations of state features are commonly used in reinforcement learning to approximately represent the value function in a lower-dimensional space [14, 15]. Formally, a k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{R}^k$. Likewise, a discrete k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{Z}^k$, where $\mathbb{Z}$ is the set of integers.

Many of the IRL algorithms listed in Section 2.2 assume that the reward function can be represented as a linear combination of features. We extend Algorithm 2 to accommodate discrete feature vectors by defining a *feature subgoal* in analogy to the state subgoal from Definition 1.

Definition 3 *Given a k-dimensional discrete feature vector Φ, a feature subgoal $g(\boldsymbol{f})$ is the set of states in S which map to the coordinate $\boldsymbol{f}$ in the feature space. Formally,*

$g(\boldsymbol{f}) = \{s \in S : \Phi(s) = \boldsymbol{f}\}$ where $\boldsymbol{f} \in \mathbb{Z}^k$. The associated feature subgoal reward function $R_{g(\boldsymbol{f})}(s)$ is defined as follows:

$$R_{g(f)}(s) = \begin{cases} c, & s \in g(\boldsymbol{f}) \\ 0, & s \notin g(\boldsymbol{f}) \end{cases} \tag{20}$$

where c is a positive constant.

From this definition it can be seen that a state subgoal is simply a specific instance of a feature subgoal, where the features are binary indicators for each state in S. Algorithm 2 runs exactly as before, with the only difference being that the support of the prior over reward functions $P(g)$ is now defined as the set of unique feature coordinates induced by mapping S through ϕ. Proposition 1 is also still valid should we wish to again limit the set of possible subgoals to only those feature coordinates in the observed demonstrations, $\Phi(s_{1:N})$. Finally, feature subgoals do not modify any of the assumptions of Theorem 1, thus convergence is still attained in 0-1 loss.

4 Experiments

Experimental results are given for three test cases. All three use a 20×20 Grid World MDP (total of 400 states) with walls. Note that while this is a relatively simple MDP, it is similar in size and nature to experiments done in the seminal papers of each of the compared algorithms. Also, the intent of the experiments is to compare basic properties of the algorithms in nominal situations (as opposed to finding the limits of each).

The agent can move in all eight directions or choose to stay. Transitions are noisy, with probability 0.7 of moving in the chosen direction. The discount factor $\gamma = 0.99$, and value iteration is used to find the optimal value function for all of the IRL algorithms tested. The demonstrator in each case makes optimal decisions based on the true reward function. While this is not required for Bayesian nonparametric IRL, it is an assumption of one of the other algorithms tested [7]. In all cases, the 0-1 policy loss function is used to measure performance. The 0-1 policy loss simply counts the number of times that the learned policy (i.e. the optimal actions given the learned reward function) does not match the demonstrator over the set of observed state-action pairs.

4.1 Grid World

The first example uses the state-subgoal Bayesian nonparametric IRL algorithm. The prior over subgoal locations is chosen to be uniform over states visited by the demonstrator (as in Proposition 1). The demonstrator chooses optimal actions towards each of three subgoals $(x, y) = \{(10, 12), (2, 17), (2, 2)\}$, where the next subgoal is chosen only after arrival at the current one. Figure 3 shows the state-action pairs of the demonstrator (left), the 0-1 policy loss averaged over 25 runs (center), and the posterior mode of subgoals and partition assignments (colored arrows denote assignments to the corresponding colored boxed subgoals) after 100 iterations (right). The algorithm reaches a minimum in loss after roughly 40 iterations, and the mode of the posterior subgoal locations converges to the correct coordinates. We note that while the subgoal locations have correctly converged after 100 iterations, the partition assignments for each state-action pair have not yet converged for actions whose subgoal is somewhat ambiguous.

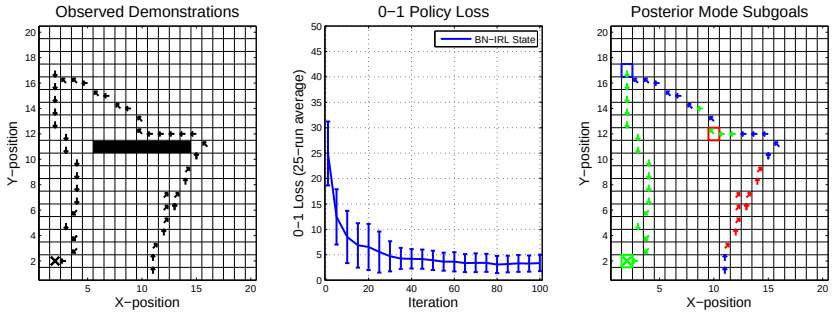

Fig. 3: Observed state-action pairs for simple grid world example (left), where arrows indicate direction of the chosen action and X's indicate choosing the "stay" action. 0-1 policy loss for Bayesian nonparametric IRL (center). Posterior mode of subgoals and partition assignments (right). Colored arrows denote assignments to the corresponding colored boxed subgoals.

4.2 Grid World with Features Comparison

In the next test case, Bayesian nonparametric IRL (for both state- and feature-subgoals) is compared to three other IRL algorithms, using the same Grid World setup as in Section 4.1: "Abbeel" IRL using the quadratic program variant [7], Maximum Margin Planning using a loss function that is non-zero at states not visited by the demonstrator [8], and Bayesian IRL [9]. A set of six features $\phi_{1:6}(s)$ are used, where feature k has an associated state s_{ϕ_k}. The value of feature k at state s is simply the Manhattan distance (1-norm) from s to s_{ϕ_k}:

$$\phi_k(s) = ||s - s_{\phi_k}||_1 \tag{21}$$

The true reward function is defined as $R(s) = \boldsymbol{w}^T \boldsymbol{\phi}(s)$ where $\boldsymbol{w}$ is a vector of randomly-chosen weights. The observations consist of five demonstrations starting at state $(x, y) = (15, 1)$, each having 15 actions which follow the optimal policy corresponding to the true reward function. Note that this dataset satisfies the assumptions of the three compared algorithms, though it does not strictly follow the generative process of Bayesian nonparametric IRL. Figure 4 shows the state-action pairs of the demonstrator (left) and the 0-1 policy loss, averaged over 25 runs versus iteration for each algorithm (right). All but Bayesian IRL achieve convergence to the same minimum in policy loss by 20 iterations, and Bayesian IRL converges at roughly 100 iterations (not shown). Even though the assumptions of the Bayesian nonparametric IRL were not strictly satisfied (the assumed model (10) did not generate the data), both the state- and feature-subgoal variants of the algorithm achieved performance comparable to the other IRL methods.

Table 1 compares average initialization and per-iteration run-times for each of the algorithms. These are given only to show general trends, as the Matlab implementations of the algorithms were by no means optimized for efficiency. The initialization of Bayesian nonparametric IRL takes much longer than the others, since during this period the algorithm is pre-computing optimal value functions for each of the possible subgoal locations (i.e. each of the states encountered by the demonstrator). However, the Bayesian nonparametric IRL per-iteration time is roughly an order of magnitude less than the other algorithms, since the others must re-compute an optimal value function each iteration.

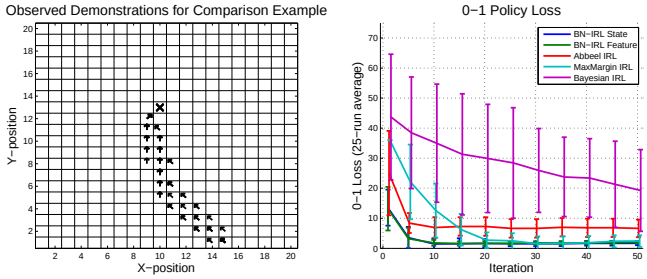

Fig. 4: Observed state-action pairs for grid world comparison example (left). Comparison of 0-1 Policy loss for various IRL algorithms (right).

Table 1: Run-time comparison for various IRL algorithms.

	Initialization (sec)	Per-iteration (sec)
BN-IRL	15.3	0.21
Abbeel-IRL	0.42	1.65
MaxMargin-IRL	0.41	1.16
Bayesian-IRL	0.56	3.27

4.3 Grid World with Loop

In the final experiment, five demonstrations are generated using subgoals as in Section 4.1. The demonstrator starts in state $(x, y) = (10, 1)$, and proceeds to subgoals $(x, y) = \{(19, 9), (10, 17), (2, 9), (10, 1)\}$. Distance features (as in Section 4.2) are placed at each of the four subgoal locations. Figure 5 (left) shows the observed state-action pairs. This dataset clearly violates the assumptions of all three of the compared algorithms, since more than one reward function is used to generate the state-action pairs. However, the assumptions are violated in a reasonable way. The data resemble a common robotics scenario in which an agent leaves an initial state, performs some tasks, and then returns to the same initial state.

Figure 5 (center) shows that the three compared algorithms, as expected, do not converge in policy loss. Both Bayesian nonparametric algorithms, however, perform almost exactly as before and the mode posterior subgoal locations converge to the four true subgoals (Figure 5 right). Again, the three compared algorithms would have worked properly if the data had been generated by a single reward function, but such a reward function would have to be significantly more complex (i.e. by including temporal elements). Bayesian nonparametric IRL is able to explain the demonstrations without modification or added complexity.

5 Discussion

5.1 Comparison to Existing Algorithms

The example in Section 4.2 shows that, for a simple problem, Bayesian nonparametric IRL performs comparably to existing algorithms in cases where the data are generated using a single reward function. Approximate computational run-times indicate that overall required computation is similar to existing algorithms. As noted in Section 3.2,

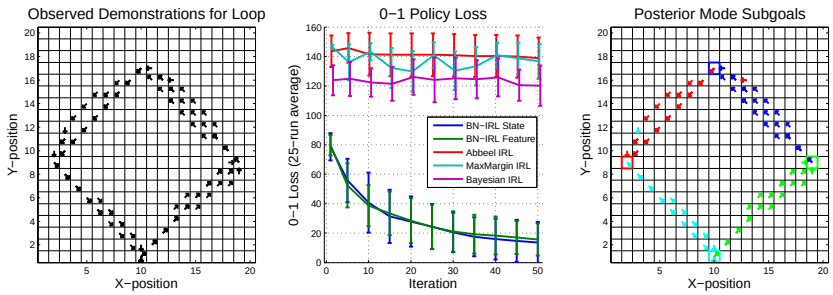

Fig. 5: Observed state-action pairs for grid world loop example (left). Comparison of 0-1 Policy loss for various IRL algorithms (center). Posterior mode of subgoals and partition assignments (right).

however, Bayesian nonparametric IRL solves for the MDP value function once for each unique state in the demonstrations. The other algorithms solve for the MDP value function once per iteration. We see this fundamental difference as an advantage which will make the new algorithm scalable to real-world domains where the size of the state space is large and the set of demonstrations is small. Testing in these larger domains is an area of ongoing work.

The loop example in Section 4.3 highlights several fundamental differences between Bayesian nonparametric IRL and existing algorithms. While the example resembles the fairly-common traveling salesman problem, it breaks the fundamental assumption of existing IRL methods that the demonstrator is optimizing a *single* reward function. These algorithms could be made to properly handle the loop case, but not without added complexity or manual partitioning of the demonstrations. Bayesian nonparametric IRL, on the other hand, is able to explain the loop example without any modifications. The ability of the new algorithm to automatically partition the data and explain each group with a simple subgoal reward eliminates the need to find a single, complex temporal reward function. Furthermore, the Chinese restaurant process prior naturally limits the number of partitions in the resulting solution, rendering a parsimonious explanation of the data.

5.2 Related and Future Work

Aside from the relation to existing IRL methods, we see a connection to option methods for MDPs [25]. While the original work explains how to *use* options to perform potentially complex tasks in an MDP framework, Bayesian nonparametric IRL could be used to *learn* options from demonstrations. Options in this case would take the form of optimal policies corresponding to each of the learned subgoal rewards. Exploration of the connection to option learning is an avenue of future work.

There are several other areas of ongoing and future work. First, the results given here are for simple problems and are by no means exhaustive. Ongoing work seeks to apply the method in more complex robotics domains where the size of the state space is significantly larger, and the observations are generated by an actual human demonstrator. Also, Bayesian nonparametric IRL could be applied to higher-level planning problems where the list of subgoals found by the algorithm may be useful in more richly analyzing the human demonstrator.

Bibliography

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

[2] H. Kautz and J. F. Allen, "Generalized plan recognition," in *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 32–37, AAAI, 1986.

[3] D. Verma and R. Rao, "Goal-Based Imitation as Probabilistic Inference over Graphical Models," *Advances in Neural Information Processing Systems 18*, vol. 18, pp. 1393–1400, 2006.

[4] C. L. Baker, R. Saxe, and J. B. Tenenbaum, "Action understanding as inverse planning.," *Cognition*, vol. 113, no. 3, pp. 329–349, 2009.

[5] A. Jern, C. G. Lucas, and C. Kemp, "Evaluating the inverse decision-making approach to preference learning," *Processing*, pp. 1–9, 2011.

[6] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. of the 17th International Conference on Machine Learning*, pp. 663–670, 2000.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Twentyfirst international conference on Machine learning ICML 04*, no. ICML, p. 1, 2004.

[8] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," *Proc. of the 23rd International Conference on Machine Learning*, pp. 729–736, 2006.

[9] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," *IJCAI*, pp. 2586–2591, 2007.

[10] G. Neu and C. Szepesvari, "Apprenticeship learning using inverse reinforcement learning and gradient methods," in *Proc. UAI*, 2007.

[11] U. Syed and R. E. Schapire, "A Game-Theoretic Approach to Apprenticeship Learning," *Advances in Neural Information Processing Systems 20*, vol. 20, pp. 1–8, 2008.

[12] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum Entropy Inverse Reinforcement Learning," in *Proc AAAI*, pp. 1433–1438, AAAI Press, 2008.

[13] M. Lopes, F. Melo, and L. Montesano, "Active Learning for Reward Estimation in Inverse Reinforcement Learning," *Machine Learning and Knowledge Discovery in Databases*, pp. 31–46, 2009.

[14] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, vol. 5 of *the Optimization and Neural Computation Series*. Athena Scientific, 1996.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[16] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis*, vol. 2 of *Texts in statistical science*. Chapman & Hall/CRC, 2004.

[17] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[18] E. B. Sudderth, "Graphical Models for Visual Object Recognition and Tracking by," *Thesis*, vol. 126, no. 1, p. 301, 2006.

[19] M. D. Escobar and M. West, "Bayesian density estimation using mixtures," *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 577– 588, 1995.

[20] R. M. Neal, "Markov Chain Sampling Methods for Dirichlet Process Mixture Models," *Journal Of Computational And Graphical Statistics*, vol. 9, no. 2, p. 249, 2000.

[21] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An Introduction to MCMC for Machine Learning," *Science*, vol. 50, no. 1, pp. 5–43, 2003.

[22] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis (Springer Series in Statistics)*. Springer, 1985.

[23] R. M. Neal, "Probabilistic Inference Using Markov Chain Monte Carlo Methods," *Intelligence*, vol. 62, no. September, p. 144, 1993.

[24] G. O. Roberts and S. K. Sahu, "Updating schemes, correlation structure, blocking and parameterisation for the Gibbs sampler," *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, vol. 59, no. 2, pp. 291–317, 1997.

[25] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.