Inverse Reinforcement Learning via Convex Optimization

Hao Zhu, Yuan Zhang, and Joschka Boedecker

Department of Computer Science and IMBIT//BrainLinks-BrainTools
University of Freiburg

June 26, 2025

Abstract

We consider the inverse reinforcement learning (IRL) problem, where an unknown reward function of some Markov decision process is estimated based on observed expert demonstrations. In most existing approaches, IRL is formulated and solved as a nonconvex optimization problem, posing challenges in scenarios where robustness and reproducibility are critical. We discuss a convex formulation of the IRL problem (CIRL) initially proposed by Ng and Russel, and reformulate the problem such that the domain-specific language `CVXPY` can be applied directly to specify and solve the convex problem. We also extend the CIRL problem to scenarios where the expert policy is not given analytically but by trajectory as state-action pairs, which can be strongly inconsistent with optimality, by augmenting some of the constraints. Theoretical analysis and practical implementation for hyperparameter auto-selection are introduced. This note helps the users to easily apply CIRL for their problems, without background knowledge on convex optimization.

1 Introduction

Inverse reinforcement learning (IRL) problems [NR00] aim to estimate an unknown reward function of some Markov decision process (MDP), based on observed expert demonstrations. A variety of IRL problem formulations have been proposed over the years, and have been widely used in engineering such as autonomous driving and robotics (see Arora and Doshi [AD21], Ruiz-Serra and Harré [RSH23] for a detailed review). However, most of these formulations are nonconvex and thus can only be approximately solved by searching for a local optimum, which is normally achieved via some variants of subgradient methods. As a result, convergence is not always guaranteed in these methods and usually a relatively large number of iterations are required to find the (approximate) solution. Moreover, such approximate solutions can lead to issues in some specific application domains where robustness and reproducibility are highly required. In psychology and neuroscience research, for instance, IRL appears to be emerging as mathematical behavior models for estimating the intrinsic reward function underlying animal or human subjects' behavior. Since a large number of subsequent analyses and experiment designs might be dependent on the IRL problem solution, lack of reproducibility can cause severe problems [YNI$^+$18, KDSP20, AKK$^+$23]. To promote the application of IRL in such areas, in this note, we consider the *convex IRL* (CIRL) problem formulation proposed by Ng and Russel [NR00], addressing two major limitations of their original work: First, while the authors have provided theoretical analysis and some numerical experiments demonstrating the feasibility of CIRL, the unmentioned solver implementation can be cumbersome and error-prone. Since many practitioners are not well versed in this procedure, the use of this method is limited to convex optimization experts. Second, the originally proposed CIRL would be infeasible when the expert behavior is strongly inconsistent with optimality, *e.g.*, when the expert alternates between multiple subgoals that conflict with each other. This is one common case when the expert demonstrations are represented as a sequence of state-action pairs in the environment. Moreover, we also provide theoretical analysis and practical implementation for automatically selecting some of the hyperparameters of CIRL. All code related to this note is open-sourced under:

`https://github.com/nrgrp/cvx_irl`.

The rest of this note is arranged as follows: Background knowledge about MDPs and our basic notation is introduced in §2.1. In §2.2, we briefly review the CIRL problem formulation from Ng and Russel [NR00] and extend some theoretical details about hyperparameter selection. We then provide a reformulation of the original CIRL problem such that it can be typed into some domain specific languages (DSLs) for convex optimization (§2.4) such as `CVXPY` [DB16], and extend it to estimating multiple reward functions corresponding to different subgoals of the expert (§2.5). Implementation of a solver using `CVXPY`, and some application examples, are provided in §3 and §4, respectively.

2 Convex inverse reinforcement learning

2.1 Markov decision processes

We consider a finite MDP given by a tuple $(\mathcal{S},\ \mathcal{A},\ \{P_a\}_{a\in\mathcal{A}},\ r,\ \gamma)$. The sets $\mathcal{S}$ and $\mathcal{A}$ are both finite sets, consisting of all possible states and actions of the MDP, respectively. In the rest of this note, we let $|\mathcal{S}| = m$ and $|\mathcal{A}| = k$. The set $\{P_a\}_{a\in\mathcal{A}}$ consists of all transition probability matrices $P_a \in \mathbf{R}^{m\times m}$ upon taking action $a \in \mathcal{A}$, *i.e.*, each entry of the matrix $P_{ij,a} = \mathbf{prob}(s_j \mid s_i, a)$, $i, j = 1, \dots, m$. Obviously, P_a has to satisfy $P_a\mathbf{1} = \mathbf{1}$, *i.e.*, the sum of each row of P_a equals to 1. The function $r\colon \mathcal{S} \to \mathbf{R}$ defines the reward function, which is assumed to be unknown under the context of IRL. Note that we write r as a function only on the set $\mathcal{S}$ for simplicity, while it can be readily extended to be on $\mathcal{S} \times \mathcal{A}$. Besides, we also assume the reward function is bounded by some positive real number $r^{\max} \in \mathbf{R}_{++}$. For a finite MDP, we can overload the reward function defined above as a vector $r \in \mathbf{R}^m$, and we will use this vector representation of the reward function in the rest of this note. (Although we will still refer to it as the reward 'function'.) The scalar $\gamma \in [0, 1)$ is the discount factor.

2.2 CIRL problem formulation

Policy and optimality condition. A (deterministic) policy for an MDP is defined as a function $\pi\colon \mathcal{S} \to \mathcal{A}$. Suppose the observed expert policy π is optimal for an MDP with some unknown reward function r, to simplify the notation, we can assume $\pi(s) = a^\star \in \mathcal{A}$ for all $s \in \mathcal{S}$ without losing of generality. (This can be done by reordering the actions and permuting the rows between all P_a if necessary.) The corresponding state transition matrix is then $P_{a^\star}$, and the value function (in vector form) $v \in \mathbf{R}^m$ evaluated for this policy π is

$$v = r + \gamma P_{a^\star} v \implies v = (I - \gamma P_{a^\star})^{-1} r, \tag{1}$$

where the matrix $I - \gamma P_{a^\star}$ is always invertible as $\gamma < 1$ [SB18]. The optimality of this policy π suggests that the corresponding reward function r must satisfy the Bellman optimality condition [SB18]

$$\begin{aligned}
& \pi(s_i) = a^\star \in \operatorname*{argmax}_{a \in \mathcal{A}} p_{i,a}^T v, \quad i = 1, \dots, m \\
\iff \quad & P_{a^\star} v \succeq P_a v, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
\iff \quad & P_{a^\star} (I - \gamma P_{a^\star})^{-1} r \succeq P_a (I - \gamma P_{a^\star})^{-1} r, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
\iff \quad & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\},
\end{aligned} \tag{2}$$

where $p_{i,a}^T$ denotes the ith row of P_a, and $\succeq$ denotes componentwise inequality between vectors.

The CIRL problem. The optimality condition (2) and boundedness assumption in §2.1 provide a solution set of the reward functions for the IRL problem. However, such a set contains some trivial solutions, *e.g.*, $r = c$, where $c \in \mathbf{R}^m$, $c_1 = \cdots = c_m$ is any constant vector. To choose a nontrivial (or 'meaningful') reward function from this solution set, we consider the following optimization problem:

$$\begin{array}{ll}
\text{minimize} & J(r) + \lambda \phi(r) \\
\text{subject to} & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
& r^{\max} \succeq r \succeq -r^{\max},
\end{array} \tag{3}$$

where the unknown reward function $r \in \mathbf{R}^m$ is the variable, $\{P_a\}_{a \in \mathcal{A}}$ and γ are the problem data, and $r^{\max} > 0$, $\lambda \geq 0$ are the hyperparameters. Note that we use the aforementioned solution set as the feasible set of (3). (The boundedness constraint on r can also be tightened to be asymmetric as, *e.g.*, $r^{\max} \succeq r \succeq 0$, if some prior information about r such as nonnegativity is given. Nevertheless, we will keep the general case as in (3) for the rest of this note.) The two functions $J(r)$ and $\phi(r)$ represent two criteria that a reward function is considered to be meaningful [NR00]. Firstly, it is natural to favor reward functions that maximize the margin between the observed expert policy π and all other possible policies at all states as much as possible, corresponding to the primary objective:

$$\begin{aligned}
J(r) &= -\sum_{i=1}^m \left(p_{i,a^\star}^T v - \sup_{a \in \mathcal{A} \setminus \{a^\star\}} p_{i,a}^T v \right) \\
&= -\sum_{i=1}^m \inf_{a \in \mathcal{A} \setminus \{a^\star\}} (p_{i,a^\star}^T - p_{i,a}^T) v \\
&= -\sum_{i=1}^m \inf_{a \in \mathcal{A} \setminus \{a^\star\}} \left((p_{i,a^\star}^T - p_{i,a}^T)(I - \gamma P_{a^\star})^{-1} r \right),
\end{aligned}$$

where $p_{i,a}^T$ denotes the ith row of P_a, and the last equality is from substituting (1). Secondly, it is also believed that the reward function r should be as sparse as possible, resulting in the ℓ_1-penalty function $\phi(r) = \|r\|_1$. Put together, we have

$$\begin{array}{ll}
\text{minimize} & -\sum_{i=1}^m \inf_{a \in \mathcal{A} \setminus \{a^\star\}} \left((p_{i,a^\star}^T - p_{i,a}^T)(I - \gamma P_{a^\star})^{-1} r \right) + \lambda \|r\|_1 \\
\text{subject to} & (P_{a^\star} - P_a)(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad \text{for all } a \in \mathcal{A} \setminus \{a^\star\} \\
& r^{\max} \succeq r \succeq -r^{\max},
\end{array} \tag{4}$$

which is a convex optimization problem originally proposed by Ng and Russel [NR00] for IRL problems. The convexity of problem (4) is readily shown: The two constraints are all linear inequality constraints on r, and the objective is convex since it is a nonnegative sum of a sequence of convex functions, *i.e.*, the negative infimum over affine $-\inf_{a\in\mathcal{A}\setminus\{a^\star\}}\left((p_{i,a^\star}^T - p_{i,a}^T)(I-\gamma P_{a^\star})^{-1}r\right)$, $i = 1,\ldots,m$, and the ℓ_1-norm function $\|r\|_1$, which are all convex functions about variable r [BV04].

2.3 Hyperparameter selection

The scalarization weight λ is a free parameter such that by varying it in $[0,\infty)$ we trade off between the primary objective $J(r)$ and the penalty term $\phi(r)$. Specifically, there exists a value

$$\lambda^{\max} = \inf_{z\in\partial J(0)} \|z\|_\infty, \tag{5}$$

where $\partial J(0)$ is the subdifferential of $J(r)$ at $r=0$, such that if $\lambda \geq \lambda^{\max}$, the optimal of (3) is achieved at $r=0$. To show this, note that $r=0$ is always in the feasible set defined by the constraints of (3), thus to ensure the optimal value is achieved by $r=0$ for a given λ, we only need to guarantee

$$0 \in \partial J(0) + \lambda\partial\|x\|_1\Big|_{x=0} \quad\Longleftrightarrow\quad -\partial J(0)\cap\lambda\partial\|x\|_1\Big|_{x=0} \neq \emptyset.$$

This is equivalent to requiring that there exists some $z\in\partial J(0)$ such that

$$-z \in \lambda\partial\|x\|_1\Big|_{x=0} \quad\Longleftrightarrow\quad \|z\|_\infty \leq \lambda,$$

which is guaranteed if $\lambda \geq \inf_{z\in\partial J(0)}\|z\|_\infty$. This corresponds to the critical value

$$\lambda^{\max} = \inf_{z\in\partial J(0)} \|z\|_\infty$$

as stated in (5), which ends the proof.

By calculating $\lambda^{\max}$ according to (5) and assigning the value to λ, one can find the 'simplest' r under the given problem data. In practice, however, instead of directly calculating $\lambda^{\max}$, it is more common to use iterative methods, such as bisection, to find a $\lambda < \lambda^{\max}$ that is close to $\lambda^{\max}$ under some threshold. (We demonstrate an implementation of this procedure for automatically determining λ in §A.) Regarding the reward function bound $r^{\max}$, since it has no influence on the indices of the nonzero entries of r, but just tight the absolute value of these nonzero entries such that (3) is not unbounded below, one can simply assign their favorite positive number to $r^{\max}$ when specifying the problem.

2.4 CIRL in regularized linear program form

We introduce a reformulation of the problem (4) such that it can be directly specified and solved by `CVXPY`. Introducing the epigraph variable $s\in\mathbf{R}^m$ for the primary objective

$$J(r) = -\sum_{i=1}^m \inf_{a\in\mathcal{A}\setminus\{a^\star\}}\left((p_{i,a^\star}^T - p_{i,a}^T)(I-\gamma P_{a^\star})^{-1}r\right),$$

the problem (4) is straightforwardly equivalent to

$$
\begin{array}{ll}
\text{minimize} & \mathbf{1}^T s + \lambda \|r\|_1 \\
\text{subject to} & \begin{bmatrix} p_{i,a^\star}^T - p_{i,\tilde{a}_1}^T \\ \vdots \\ p_{i,a^\star}^T - p_{i,\tilde{a}_{k-1}}^T \end{bmatrix} (I - \gamma P_{a^\star})^{-1} r + s_i \succeq 0, \quad i = 1, \dots, m \\
& (P_{a^\star} - P_{\tilde{a}_i})(I - \gamma P_{a^\star})^{-1} r \succeq 0, \quad i = 1, \dots, k-1 \\
& r^{\max} \succeq r \succeq -r^{\max},
\end{array}
\tag{6}
$$

where $\tilde{a}_1, \dots, \tilde{a}_{k-1} \in \mathcal{A} \setminus \{a^\star\}$ (which is trivially equivalent to $a_1, \dots, a_{k-1}$ if $a^\star = a_k$). The problem (6) is then an ℓ_1-regularized linear program over variables s and r, with $2mk$ inequality constraints. Notice that the optimal value of (6) is expected to be attained with some $s \preceq 0$, given that the problem is feasible. Thus one can also explicitly introduce $s \preceq 0$ as an additional constraint on s to (6), but it will not influence the solution. In practice, the potential prior sparsity pattern of the involved matrices in (6) can be readily incorporated to accelerate the matrix operations.

2.5 Reward estimation for subgoals

The optimality condition (2) requires that the observed expert policy is globally optimal under the unknown reward function. However, in real-world applications, this is not necessarily the case, especially when the expert's policy is not directly given but represented as a sequence of state-action pairs. For instance, in many behavioral experiments, *e.g.*, Hamaguchi *et al.* [HTAW22], De La Crompe *et al.* [DLCSS+23], the subject needs to achieve a series of subgoals such that the whole task is fulfilled, *i.e.*, the globally optimal policy consists of a series of locally optimal policies (and might even conflict with each other). In these scenarios, directly solving (6) is likely to have the trivial solution $r = 0$ returned, which is useless to the researchers.

To make the solution of (6) useful when the given expert demonstration is only locally optimal (optimal within a subset of all states during some specific time period), we introduce relaxation on some of the constraints below. Consider the expert policy is indirectly represented by a sequence of n state-action pairs $\mathcal{D} = ((s_1, a_1), \dots, (s_n, a_n))$, where $s_1, \dots, s_n \in \mathcal{S}$ and $a_1, \dots, a_n \in \mathcal{A}$. Let the n-demonstrations partitioned by indices $t_0, \dots, t_p \in \mathbf{Z}_{++}$, $t_0 = 1 < t_1 < \dots < t_p = n$, and each of the partition $\mathcal{D}_j = ((s_{t_j}, a_{t_j}),\ \dots,\ (s_{t_{j+1}}, a_{t_{j+1}}))$, $j = 0, \dots, p-1$, corresponds to a locally optimal policy within the subset of states $\mathcal{S}_j = \{s \mid (s, a) \in \{(s_{t_j}, a_{t_j}),\ \dots,\ (s_{t_{j+1}}, a_{t_{j+1}})\}\}$. (In practice, such partition indices $t_0, \dots, t_p$ can be obtained using different methods, such as via change point detection of time series [JTL+24], or by fitting some latent variable models [ZCK+24], *etc.*, which is, however, not the major focus of this note.) We can then relax the first and second constraints in (6) from global optimality on the set of all states $\mathcal{S}$ to local optimality on the subset of visited states $\mathcal{S}_j$ under demonstrations in $\mathcal{D}_j$, $j = 0, \dots, p$, *i.e.*,

$$
\begin{bmatrix} \tilde{p}_{i,a^\star}^T - \tilde{p}_{i,\tilde{a}_1}^T \\ \vdots \\ \tilde{p}_{i,a^\star}^T - \tilde{p}_{i,\tilde{a}_{k-1}}^T \end{bmatrix} (I - \gamma \tilde{P}_{a^\star})^{-1} r + s_i \succeq 0, \quad i = 1, \dots, |\mathcal{S}_j| \tag{7}
$$

and

$$
(\tilde{P}_{a^\star} - \tilde{P}_{\tilde{a}_i})(I - \gamma \tilde{P}_{a^\star})^{-1} r \succeq 0, \quad i = 1, \dots, k-1, \tag{8}
$$

where the matrices $\tilde{P}_{a^\star}, \tilde{P}_{\tilde{a}_1}, \dots, \tilde{P}_{\tilde{a}_{k-1}} \in \mathbf{R}^{|\mathcal{S}_j| \times |\mathcal{S}_j|}$ are the submatrices of $P_{a^\star}, P_{\tilde{a}_1}, \dots, P_{\tilde{a}_{k-1}}$, respectively, by only taking those entries related to the states in $\mathcal{S}_j$, and the vector $\tilde{p}_{i,a}^T \in \mathbf{R}^{|\mathcal{S}_j|}$ are again the ith row of matrix $\tilde{P}_a$. The objective as well as the box constraint on r of (6) remain unchanged. By solving this relaxed problem of (6) for expert policy given by $\mathcal{D}_1, \dots, \mathcal{D}_p$ we can obtain the reward function r_j corresponding to the jth subgoal, $j = 1, \dots, p$. One advantage of such relaxation is that the implementation for specifying and solving the problem (6) (see §3) can be directly applied without further adaptation.

A small technical condition we should note is that some rows of the submatrices $\tilde{P}_a$ in (7) and (8) might no longer satisfy $\tilde{p}_{i,a}^T \mathbf{1} = 1$. Under many cases, this violation will only introduce some approximation error, but it can also cause severe issues when the matrix $I - \gamma \tilde{P}_{a^\star}$ is not invertible. We address this problem by renormalizing each row of $\tilde{P}_a$ if necessary as follows. If $\tilde{p}_{i,a}^T \neq 0$, we can simply overload $\tilde{p}_{i,a}^T$ with the normalized version $\tilde{p}_{i,a}^T / (\tilde{p}_{i,a}^T \mathbf{1})$, $i = 1, \dots, |\mathcal{S}_j|$, to guarantee it is still a proper probability distribution. If $\tilde{p}_{i,a}^T = 0$ (which can happen if the transition matrix P_a is deterministic), we explicitly set the ith entry of $\tilde{p}_{i,a}^T$ equals 1, *i.e.*, $(\tilde{p}_{i,a}^T)_i = 1$. Such operation can be interpreted as introducing a 'barrier' around state s_i under action a, where performing the action will simply make the state unchanged.

We introduce an example involving the above procedure in §4.2.

3 Implementation

We depict below the `CVXPY` code that specifies and solves (6).

```python
import numpy as np
import cvxpy as cp

# problem information (input from user)
m = None  # number of states
gamma = None  # discount factor
Pastr = None  # transition matrix of the optimal action
lPa = []  # list of transition matrices of the other actions

# hyperparameters (input from user)
rmax = None  # reward function bound
lbd = None  # scalarization weight

r = cp.Variable(m)
s = cp.Variable(m)

constraints = []
H = np.linalg.inv(np.identity(m) - gamma * Pastr)
D = np.array([[Pastr[i] - Pa[i] for Pa in lPa] for i in range(m)])
for i in range(m):
    constraints.append(D[i] @ H @ r + s[i] >= 0)
for Pa in lPa:
    constraints.append((Pastr - Pa) @ H @ r >= 0)
constraints.append(rmax >= r)
constraints.append(r >= -rmax)
```

```

obj = cp.Minimize(cp.sum(s) + lbd * cp.norm(r, 1))
prob = cp.Problem(obj, constraints)
prob.solve()
```

To fully specify the problem, users only have to assign the required problem information according to their MDP and expert policy, as well as the hyperparameters $r^{\max}$ and λ. Note that in this implementation λ is declared as a constant value. If one would like to hand-tune the value of λ, especially when m is large, declaring λ as a `Parameter` object of `CVXPY` would substantially save the computing time upon solving the problem multiple times with different λ values (see §A).

4 Examples

In this section, we show two examples of IRL via solving the convex optimization problem (6). All experiments were performed on an AMD Ryzen™ 9 7950X (4.5 GHz) CPU. The code to reproduce these examples is available at

`https://github.com/nrgrp/cvx_irl`.

4.1 Example 1: Gridworld

In the first example we consider a 16×16 gridworld, where the agent can choose to go *left*, *right*, *up*, *down*, or *stay* at the current state. Upon each action execution, except for moving towards the selected direction, there will always be a 10% chance of moving randomly out of all 5 possible directions. The ground truth reward distribution and corresponding optimal policy are shown in the left part of figure 1.

To recover the true reward function given the depicted optimal policy, the problem (6) was solved with hyperparameter $\lambda = 2$ and $r^{\max} = 100$. The recovered reward function and the corresponding optimal policy under this recovered reward are shown in the right part of figure 1. It can be seen immediately that the recovered reward function aligns well with the true reward. Quantitatively, the cosine similarity between the true reward and the CRIL recovered reward is 0.85, where this similarity between the true and some random reward should be around 0. Besides, by comparing the true optimal policy and the optimal policy under the recovered reward, the fraction of matched action over all states is 0.94. Note that in this example, the running time for solving (6) without incorporating any matrix sparsity pattern is 1.65 seconds, which is to the best of our knowledge significantly faster than the other existing IRL algorithms, whose computing time are generally measured in minutes under an environment with similar scale as a 16×16 gridworld.

4.2 Example 2: The greedy snake

We consider a 48×48 gridworld environment with nonstationary reward distribution. Only one state is rewarded at each time. An expert (the greedy snake) will chase for the reward, and once the reward is collected, another reward will be generated at a random state. This task is a typical example where the expert needs to alternate between multiple subgoals, such that its policy might conflict with each other throughout the whole episode, but is locally (in time) optimal for each reward location. The action space consists of 5 actions: *left*, *right*, *up*, *down*, and *stay*. Upon each action execution of the expert, except for moving towards the selected direction, there will always be a 10% chance of moving

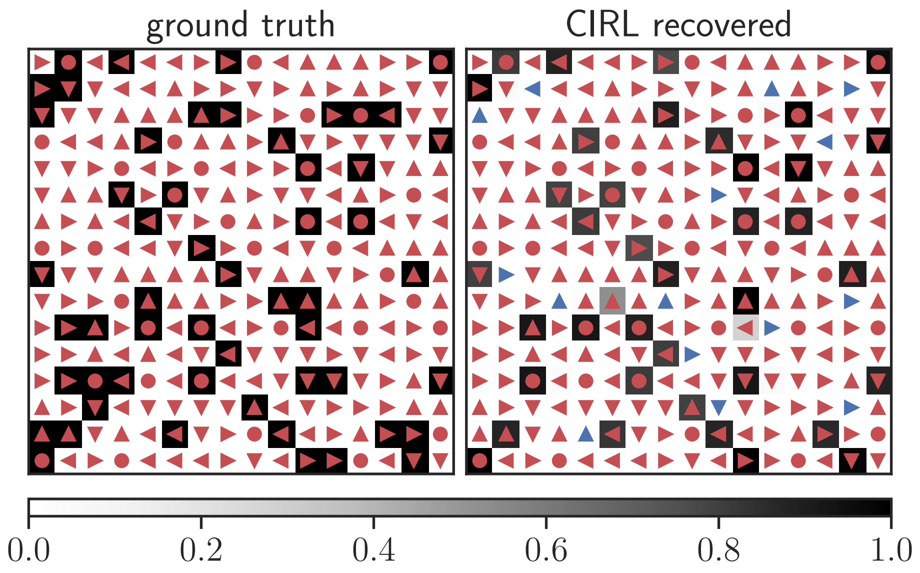

Figure 1 *Left.* The 16×16 gridworld. The grey colormap depicts the reward distribution in the environment. The red arrows (and dots for the action *stay*) indicate the optimal moving direction under the aforementioned reward distribution. *Right.* Recovered reward distribution and corresponding optimal policy from CIRL. The inconsistent optimal policy compared to the ground truth is colored blue. Both the ground truth and recovered reward function are normalized to be in the range of $[0, 1]$.

randomly out of all 5 possible directions. The expert's policy is given by its trajectory as a sequence of state-action pairs $\mathcal{D}$, and the partition time where the rewarded state changed is also given. (In practice, such partition time might indeed not be known, but various methods can be applied to obtain this information according to the application scenarios [JTL+24, ZCK+24]. Here we assume the trajectory partition is given only for the convenience of demonstrating the feasibility of estimating the nonstationary reward function according to the expert's trajectory via solving (6) with augmented constraints (7) and (8).) The hyperparameters are set to be $r^{\max} = 100$ and $\lambda = 0.5$ when solving (6) for all trajectory partitions.

Figure 2 shows the ground truth and estimated reward location for individual trajectory segments. The estimated reward locations were obtained by taking the state with the largest estimated reward value, for individual trajectory segments. The full expert trajectory consists of 2022 actions with 48 reward location changes. It can be seen that our recovered reward distribution matches exactly to the ground truth. This leads to a 0.94 fraction of the matched action through the whole trajectory between the observed expert action and the predicted optimal action under the estimated reward. We also provide a video showing the aforementioned results in figure 2, which can be found at

https://github.com/nrgrp/cvx_irl.

The average computing time for the individual trajectory segments are 9.01 ± 1.10 ms (mean $\pm$ standard error), which corresponds to a total computing time of 432 ms.

These results indicate that by relaxing the constraints of problem (6) to (7) and (8), we are able to apply CIRL for nonstationary reward function without knowing the expert's policy at all states.

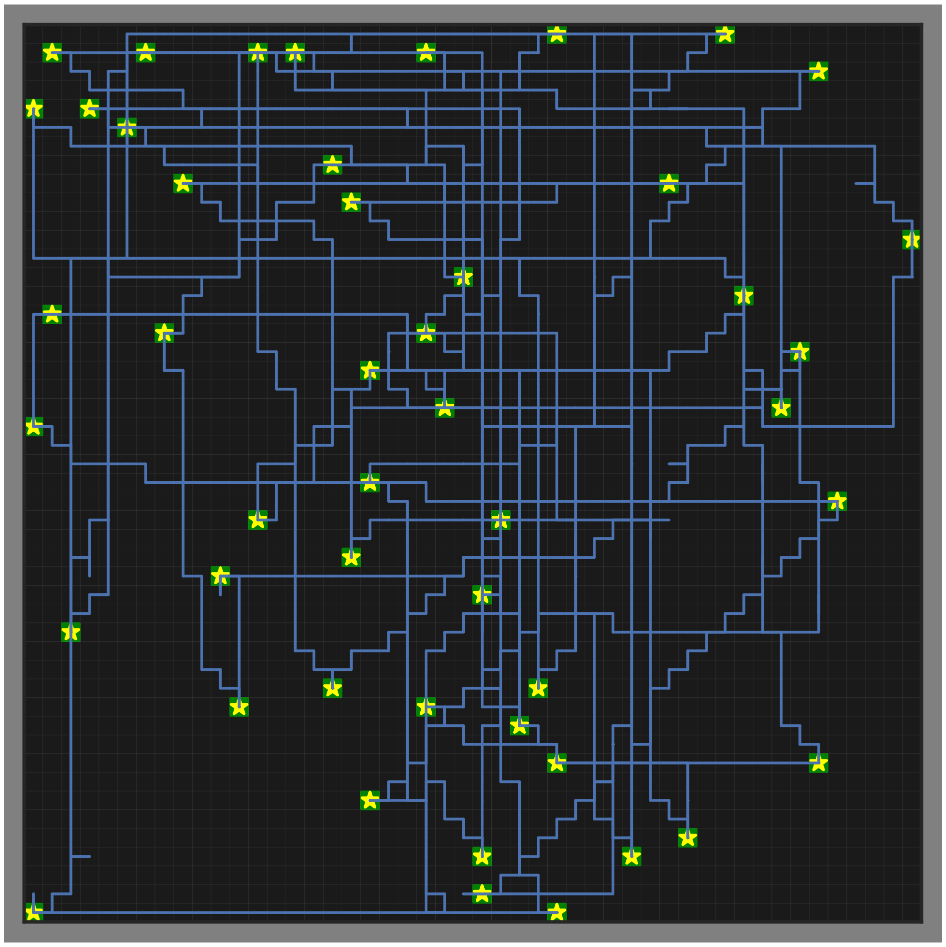

Figure 2 The greedy snake environment. The blue line shows the expert's trajectory. The ground truth and estimated reward distribution for each trajectory segmentation are depicted as green squares and yellow stars, respectively.

In the original work from Ng and Russel [NR00], if the expert's policy is given by trajectories, Monte Carlo simulation is required to obtain the full analytical expert policy π, which can be time-consuming even under moderate scaled environments, and their required optimality condition for all states will make the problem (6) infeasible (or only feasible at trivial solution $r = 0$) where the expert's policy is only locally optimal for some time period before the reward function changing.

5 Related work and comments

Inverse reinforcement learning. The IRL problem was initially proposed by Ng and Russel [NR00]. Many variants of the IRL problem formulations has been introduced over the decades, such as maximum margin methods [AN04], probabilistic methods [LMM09], and the most widely used maximum entropy methods [ZMBD08]. (The references listed here only serve as examples for different types of methods for IRL; interested readers can refer to Arora and Doshi [AD21], Ruiz-Serra and Harré [RSH23] for a detailed comprehensive review.) To the best of our knowledge, except for the method from Ng and Russel [NR00] mentioned in this note, most of these IRL problem formulations are nonconvex. Exceptions are some maximum margin methods, in particular, which are based on linear programming (*e.g.*, Syed *et al.* [SBS08]). These methods, however, focus more on apprenticeship learning, which aims at obtaining a policy π comparable with the experts, instead of the reward function itself. Hence, the reward function r is only a byproduct of the primary objective in these methods and sometimes is not even returned explicitly. In this note, on the contrary, we consider the IRL formulation where the primary objective is to obtain the unknown reward function r according to which the expert optimized their policy. Besides, we do not tend to depreciate the class of maximum entropy IRL methods which has been widely used in different areas, instead, we only claim that the CIRL method could be more suitable for the scenarios where reproducibility and robustness of the IRL results are highly required, or the running time is strictly constrained.

Contribution. Our major contributions in extending the original work by Ng and Russel [NR00] are threefold. First, we reformulated the CIRL problem into the epigraph form such that it can be easily typed into some DSLs for convex optimization. In particular, we provide an implementation of CIRL based on `CVXPY`, which allows practitioners that are not well versed in convex optimization to have access to applying this method easily. Second, we augment the constraints of CIRL such that it can be applied in scenarios where the expert's behavior is given by trajectory as state-action pairs, which can be strongly inconsistent with optimality. One common instance is when the expert alternates between multiple subgoals that conflict with each other, as demonstrated in §4.2. Last but not least, we provide a thorough theoretical analysis of the selection of the ℓ_1-penalty parameter λ, which is (conceptually) useful if one would like to obtain the most sparse reward function given some problem data without obtaining the trivial solution $r = 0$. As a complementary for the theory, we also introduce a `CVXPY` implementation for fast automatic tuning of the hyperparameter λ in practice.

Acknowledgements

This work has been funded as part of BrainLinks-BrainTools, which is funded by the Federal Ministry of Economics, Science and Arts of Baden-Württemberg within the sustainability program for projects of the Excellence Initiative II, and CRC/TRR 384 'IN-Code'.

References

[AD21] S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[AKK+23] M. Alyahyay, G. Kalweit, M. Kalweit, G. Karvat, J. Ammer, A. Schneider, A. Adzemovic, A. Vlachos, J. Boedecker, and I. Diester. Mechanisms of premotor-motor cortex interactions during goal directed behavior. *bioRxiv*, pages 2023–01, 2023.

[AN04] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DLCSS+23] B. De La Crompe, M. Schneck, F. Steenbergen, A. Schneider, and I. Diester. FreiBox: A versatile open-source behavioral setup for investigating the neuronal correlates of behavioral flexibility via 1-photon imaging in freely moving mice. *Eneuro*, 10(4), 2023.

[HTAW22] K. Hamaguchi, H. Takahashi-Aoki, and D. Watanabe. Prospective and retrospective values integrated in frontal cortex drive predictive choice. *Proceedings of the National Academy of Sciences*, 119(48):e2206067119, 2022.

[JTL+24] Z. Jia, V. Thumuluri, F. Liu, L. Chen, Z. Huang, and H. Su. Chain-of-thought predictive control. In *International Conference on Machine Learning (ICML)*, 2024.

[KDSP20] M. Kwon, S. Daptardar, P. R. Schrater, and X. Pitkow. Inverse rational control with partially observable continuous nonlinear dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 7898–7909, 2020.

[LMM09] M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Batabases*, pages 31–46. Springer, 2009.

[NR00] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

[RSH23] J. Ruiz-Serra and M. S. Harré. Inverse reinforcement learning as the algorithmic basis for theory of mind: Current methods and open problems. *Algorithms*, 16(2):68, 2023.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[SBS08] U. Syed, M. Bowling, and R. E. Schapire. Apprenticeship learning using linear programming. In *International Conference on Machine Learning (ICML)*, pages 1032–1039, 2008.

[YNI+18] S. Yamaguchi, H. Naoki, M. Ikeda, Y. Tsukada, S. Nakano, I. Mori, and S. Ishii. Identification of animal behavioral strategies by inverse reinforcement learning. *PLoS Computational Biology*, 14(5):e1006122, 2018.

[ZCK$^+$24] H. Zhu, B. De La Crompe, G. Kalweit, A. Schneider, M. Kalweit, I. Diester, and J. Boedecker. Multi-intention inverse Q-learning for interpretable behavior representation. *Transactions on Machine Learning Research*, 2024.

[ZMBD08] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

A Implementation of auto-tuning the scalarization weight

In this section, we extend the implementation of CIRL in §3 to automatically select the hyperparameter λ to find the reward function r with maximum sparsity given the problem data, which can make full use of the capacity of CVXPY.

```python
import numpy as np
import cvxpy as cp

# problem information (input from user)
m = None  # number of states
gamma = None  # discount factor
Pastr = None  # transition matrix of the optimal action
lPa = []  # list of transition matrices of the other actions

# hyperparameters (input from user)
rmax = None  # reward function bound
lbd_up = None  # upper bound on lambda
lbd_low = None  # lower bound on lambda
epsilon = None  # terminate if lbd_up - lbd_low <= epsilon

r = cp.Variable(m)
s = cp.Variable(m)
lbd = cp.Parameter(nonneg=True)

constraints = []
H = np.linalg.inv(np.identity(m) - gamma * Pastr)
D = np.array([[Pastr[i] - Pa[i] for Pa in lPa] for i in range(m)])
for i in range(m):
    constraints.append(D[i] @ H @ r + s[i] >= 0)
for Pa in lPa:
    constraints.append((Pastr - Pa) @ H @ r >= 0)
constraints.append(rmax >= r)
constraints.append(r >= 0)

obj = cp.Minimize(cp.sum(s) + lbd * cp.norm(r, 1))
prob = cp.Problem(obj, constraints)

while True:
    lbd.value = 0.5 * (lbd_up + lbd_low)
    opt_val = prob.solve()

    if np.abs(opt_val) < 1e-6:
        lbd_up = lbd.value
    elif lbd_up - lbd_low <= epsilon:
        break
    else:
        lbd_low = lbd.value
```

In this implementation, the hyperparameter λ is declared as a nonnegative instance of the object `CVXPY.Parameter` instead of simply a number, such that the computing time can be saved when trying to re-solve the optimization problem multiple times using different λ values during the auto-tuning process. The user needs to specify a range on λ so that the bisection process will be automatically performed based on this interval. Once the width of the interval of λ is below some threshold `epsilon` given by the user, and the optimal value of (6) is not 0, the algorithm will quit with the current λ and problem solution as the output.

The code listed above can be found at

https://github.com/nrgrp/cvx_irl.