Reducing the Dimensionality of the Reward Space in the Inverse Reinforcement Learning Problem

Edouard Klein[1,2]
[1]Equipe ABC,
LORIA-CNRS, France

Matthieu Geist[2]
[2]Supélec,
IMS Research group, France

Olivier Pietquin[2,3]
[3]UMI 2958
GeorgiaTech-CNRS, France

***Abstract*—This paper deals with the Inverse Reinforcement Learning framework, whose purpose is to learn control policies from demonstrations by an expert. This method inferes from demonstrations a utility function the expert is allegedly maximizing. In this paper we map the reward space into a subset of smaller dimensionality without loss of generality for all *Markov Decision Processes* (MDPs). We then present three experimental results showing both the promising aspect of the application of this result to existing IRL methods and its shortcomings. We conclude with considerations on further research.**

I. Introduction

In the *Reinforcement Learning* (RL) framework [1], an agent is left to find the behavior that, in the long run, maximizes a cumulative reward provided by some oracle. By correctly defining the reward, one can use the RL framework to make an agent fulfil a certain task (without explicitly specifying the task).

For some tasks, defining the corresponding reward is a daunting process. Surely, one has at least an *idea* about what states (configurations of the system to be controled) are preferable to others. This is however not enough to precisely define the numerical reward that will yield the desired behavior. An example of such a task is driving a car. It is preferable not to be too close to the car in front of ours. Similarily it is preferable not to brake too hard in order to avoid being rear-ended. But exactly how much is it preferable not to break too hard over not being too close to the car in front of ours ? Variations between the relative weights of these two criteria in the reward will certainly lead to very different driving behaviors.

A proposed workaround is the *Inverse Reinforcement Learning* (IRL) paradigm [2], where one tries to learn the reward optimized by an expert[1]. Then, the *idea* about what states are preferable to others can be empirically used by a human expert driving the car. The reward function can be infered from the data of the demonstration, and the agent can mimic the expert by optimizing this reward.

The paper is organized as follows: we begin by introducing the necessary mathematical background and present the related work in the field. We then present and prove our main result, and explain its use in a Linear Programming framework. Finally, we explain the preliminary experimental work we have done and show where we are heading in light of these preliminary results.

[1]The expert may be non human, in which case one hopes to use a slow or costly expert system to train a less costly agent.

II. Background

Often, the RL problem is solved in a *Markov Decision Process* setting: the agent is said to be in a state $s_t \in S$ (transition probabilities respect the Markovian criterion. As a consequence states contain all the information the agent needs to take a decision) in which it has to choose an action $a_t \in A$. This will make it step stochastically to another state s_{t+1}, receiving a reward $R(s_t)$.

The reward is seen here as a mapping from the states only to the reals. This is not normalized in the literature, some authors using a mapping $S \times S \to \mathbb{R}$ (e.g., in [3]) or a mapping $S \times A \times S \to \mathbb{R}$. We argue that one should not take the action into account when computing the reward. In the case of IRL, recording the action can be difficult. It is possible to record the state an agent is in from an external point of view, but guessing the action from the outside is a supplementary engineering problem we shall get rid of if we can. As transitions follow each others, we also argue that there is no use in taking both states of the transition into account. The following state will be present in the next transition, so the associated reward can be computed then. Plus, needing both states to compute the reward is a sign that the states may not respect the markovian criterion. R is represented as a state indexed column vector.

We place ourselves in the case of finites MDPs. Actions are mapped with the probability with which they make the agent step from one state to the other. Formally, to each action a is associated a $|S| \times |S|$ matrix P_a whose element (i, j) is the probability to transit to state j when taking action a in state i. A policy is a mapping π: $s \in S \mapsto \pi(s) \in A$ which can be defined the same way, that is to say by associating it with a P_π matrix whose i-th line is the i-th line of the $(P_a, a = \pi(s))$ matrix. From now on, we will speak about actions and policies using indiscriminately the notations a and π or P_a and P_π.

The quality of each policy is quantified by its *value function*, a mapping $S \to \mathbb{R}$ which associates to each state the expected sum of discounted rewards an agent will get by following policy π from this state. Formally, given a policy π, the value function is defined by:

$$V^\pi(s) = E\left[\sum_{t\geq 0} \gamma^t R(s_t) \middle| s_0 = s, \pi\right] \quad (1)$$

and represented, as the reward, by a state indexed column vector.

The RL problem can be casted as finding the *optimal policy* (often noted π^*), the value function of which is greater, for every state, than the value function of any other policy: $\forall \pi, V^{\pi^*} \succeq V^\pi$. The value function of any policy is the fixed point of the associated Bellman evaluation operator T^π defined as $T^\pi V = R + \gamma P_\pi V$.

The IRL problem is the following: given an optimal policy π^* and the set of actions A from which the policy was drawn, guess the reward with respect to which this policy is optimal. The reward being the unknown, we will sometimes add the reward in the notation of the value function. Thus, V_R^π is the value function of policy π under reward R.

We denote the set of the optimal policies with respect to a certain reward R and a certain discount factor γ as $\Pi^*(R) = \left\{\pi^* | \forall \pi, V_R^{\pi^*} \succeq V_R^\pi\right\}$.

Throughout this paper, we will use the transitive notion of equivalence between rewards:

Definition 1. *Two rewards R_1 and R_2 are said to be* equivalent *if $\Pi^*(R_1) = \Pi^*(R_2)$. This is noted $R_1 \equiv R_2$.*

A reward is said to be degenerative when it is equivalent to the null vector ($R \equiv [0 \dots 0]^T$). Indeed the null vector admits any policy as an optimal policy.

III. Related work

The IRL problem was first defined in [4], where its ill-posed nature was already noted: the set of reward with respect to which the expert's policy is optimal is far from being a singleton. Furthermore there exists some degenerative rewards that admit every policy as optimal and such bear no useful information.

The work of [4] was further pushed in [2], providing a sufficient and necessary condition for rewards to be solution of the IRL problem. The problem of the non uniqueness and possible degenerativeness of the solution was worked around using a criterion according to which the difference between the value of the expert's actions and the value of the next-to-best actions is maximized. A penalty term rewarding sparse solutions is also introduced. For large (or continuous) problems, a criterion for sampled trajectories is proposed.

The seminal work of [5] proposes an iterative algorithm, where the difference between the value of the expert and the value of the second best policy is maximized. Further work (partially summed up in [6]) often used the same iterative structure, changing the argument that allows finding a unique solution. In [7], [8] and [9], the authors use a game theoretic approach, in [10], [11] and [12] the IRL problem is casted as a multiclass classification problem whereas in [13] and [6] the reward is computed using gradient methods so that the agent's behavior matches the expert's observed behavior. Finally, Bayesian methods have been proposed ; the work in [14] being very similar to previous work in [15], which was not cast as an IRL problem. Follow ups include [16] and [17]. Maximum entropy priors are introduced in [18], [19] and [20].

This paper studies reward shaping and begins to explore its potential use in the framework of [2]. We do not explore any new cost function, the references given above cover that ground extensively. We do present some preliminary experimental results that pave the way for future research, aiming at speeding up search in the reward space or defining more precisely the notion of reward sparsity.

IV. Dimensionality reduction

A. Main result

In this subsection, we will show that there exists a set of dimension $|S| - 2$ so that every non degenerative reward is equivalent to at least one element of the set.

Lemma 1. *Let R_1 be a reward vector,*
let R_2 be a reward vector so that $\exists \alpha > 0, R_2 = \alpha R_1$
the following holds: $R_1 \equiv R_2$.

Lemma 2. *Let R_1 be a reward vector,*
let $\mathbf{1}$ be the column vector whose $|S|$ elements are all equal to 1,
let R_2 be a reward vector so that $\exists \lambda \in \mathbb{R}, R_2 = R_1 + \lambda \mathbf{1}$,
the following holds: $R_1 \equiv R_2$.

Proofs of this can be found in [21].

Proposition 1. *Let $M = \{R | \mathbf{1}^T R = 0, ||R||_1 = 1\}$ be,*
the following holds: $\forall R \in \mathbb{R}^{|S|} \setminus \{\lambda \mathbf{1}, \lambda \in \mathbb{R}\}, \exists R' \in M, R' \equiv R$.

This means that the search for the reward can take place in the unit sphere intersected with and hyperplane of the reward space, thus leading to a dimensionality reduction of 2.

The proof goes as follow: by defining $R' = \alpha(R + \lambda \mathbf{1})$, with $\lambda = -\frac{\mathbf{1}^T R}{|S|}$ and $\alpha = \frac{1}{||R + \lambda \mathbf{1}||_1}$, one can see that $R' \in M$ and $R' \equiv R$.

B. Linear programming constraints

In [2], we are given a necessary and sufficient condition for a reward R to admit a given policy as optimal: for every

action P_a the expert following P_π had the possibility to take, the following matrix inequality must be met:

$$(P_\pi - P_a)(I - \gamma P_\pi)^{-1} R \succeq 0 \tag{2}$$

Although the proof can be found in the forementionned paper, we find useful to recall its main argument here: the expert is following the best policy with respect to the unknown reward function. These inequalities stem from the fact that for every action a, the expected value of the next state must be less than or the same as the expected value of the next state for taking action $\pi(s)$. This is written as $P_\pi V^\pi \succeq P_a V^\pi$. Consequently, the i^{th} line of the $(P_\pi - P_a)(I - \gamma P_\pi)^{-1}$ matrix is a constraint on R that, if satisfied, will make it preferable to choose action $\pi(i)$ over a in state i.

Equation 2 yields at most $|A| \cdot |S| - |S| = (|A| - 1)|S|$ constraints. There are $|A|$ matrices P_a, each yielding $|S|$ constraints. $|S|$ of these, however, are null because once for each state we will have $a = \pi(s)$ and the resulting line will be filled with zeros.

When a cost function is added, this is a linear programming problem. The constraints stemming from the definition of M, i.e. that $\mathbf{1}^T R = 0$ and $||R||_1 = 1$, can be added to it quite easily, thus restricting the solutions to the previously defined $(|S| - 2)$-dimensional subset.

C. Sparsity of the reward vector

Sparsity is often stated to be a quality of the reward vector. For example one of the cost functions given in [2] includes a regularization term ($P(i)$ denotes the i-th row of P):

$$J(R) = \left(\sum_{i=1}^{|S|} \min_{a \in A} (P_\pi(i) - P_a(i))(I - \gamma P_\pi)^{-1} R \right) - \lambda ||R||_1 \tag{3}$$

V. Preliminary experiment

In the previous section, no cost function of our own has been provided, as this is beyond the scope of this paper. If anything cost functions provided in [2] (e.g., the one given Equation 3) would yield good results with our LP formulation above.

A first experiment was run to see if a sparse reward could be found by modifying the simplex algorithm to work without a cost function, but enumerating only sparse rewards. In the simplex algorithm, one start by selecting a *basic feasible solution* that is to say a vertex of the polytope defined by the linear constraints. Then, one jumps from vertex to vertex by minimizing (or maximizing) the cost function. Basic feasible solutions can be found by choosing which constraints are binding and which are not. The linear system is then solved for the free variables (the variables corresponding to the non binding constraints); if a solution exists, it is a basic feasible solution.

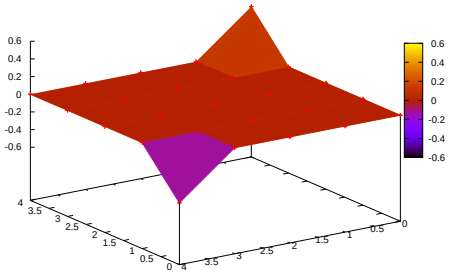

Fig. 1. Reward found by our algorithm on the classic gridworld problem. this is very similar to what can be found in [2] or [22].

As we did not make use of a cost function, we looked for sparse basic feasible solutions directly. As the constraints are inequalities, the system is loaded with slack variables. If all the slack variables are considered free, then one only needs two additional variables to get a symetric linear system. Only a few of them are solvable, they are sparse as only two components of the reward vector are non zero.

This has proved successful on the now classical gridworld problem, see Fig. 1. In this setting, the expert goes from the lower left corner $(0, 4)$ of a 5×5 gridworld to its upper right corner $(4, 0)$. The true reward function the expert has been trained with its 0 everywhere but in the upper right corner where the reward is 1. The reward found by our algorithm just adds a negative reward at the starting point that does not change the behavior. It is sparse. However, with its complexity of $O(|S|^5)$ this algorithm is not very practical.

This success on the gridworld cannot be generalized to all MDPs. It is easy to create MDPs where the constraints are so that there is no sparse basic feasible solution that explains the expert's behavior. Randomly generating the transition probabilities and the reward on a 4-states, 2-actions MDP will yield one such counter-example quite quickly.

VI. Further work

We have room for improvement in the computational complexity of the algorithm. The $O(|S|^5)$ complexity mentioned above is the worst-case complexity. The mean complexity can be vastly reduced by solving only the solvable systems and detecting the unsolvable ones earlier in the algorithm. We can also preprocess the constraints to eliminate redundancy.

The fact that our algorithm does not solve all kinds of MDPs is problematic. We have two ideas to circumvent

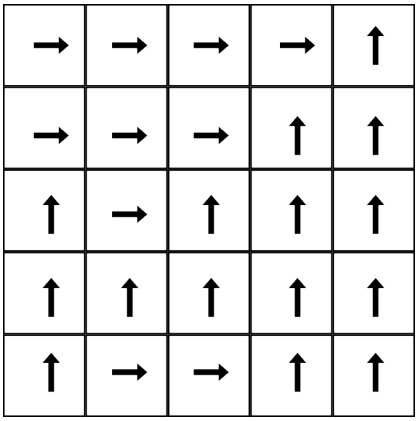

Fig. 2. Policy of the expert. This policy is found by a dynamic programming algorithm when the reward is 0 everywhere except in the upper right where it is 1. Actions are executed noisily : with probability 0.3 another action than what the agent chooses is executed.

this. The first idea is to characterize the class of MDPs our algorithm is able to work with. By restricting ourselves to a certain class of MDPs we could provide theoretical guarantees about our method. This could also help to reduce the computational complexity by allowing a quicker preprocessing of the linear constraints.

The second idea is to transform the state space so that a sparse basic feasible solution always exists. Although more difficult, this would be more powerful as we could tackle any kind of MDPs. In the gridworld, the expert is going from somewhere to somewhere else and the topology bends itself quite well to a configuration where there is one attractive state and one repulsive state. This is the kind of configuration our algorithm outputs. Some problems however do not present this kind of topology. The balancing pole problem is typical example. The expert is trying to balance a pole with one degree of freedom in the vertical position. There is one attractive state (the vertical position) but there are two repulsive states as the pole can fall on one side or the other. This is certainly problematic for our algorithm, but a state space transformation that would bend the state space so that both repulsive states are close to each other would solve this problem. Our hope is to find some kind of automatic feature discovery mechanism that could do this.

One last track for future work is sampling or approximation. Small, discrete state spaces are fine for testing purpose. We should be able to tackle large or continuous problems. The policy of the expert is then unknown but observable. Our algorithm can be modified to work with sample transitions from the expert. Methods exist in the linear programming framework to work with sampled constraints, they could be adapted to our setting.

VII. Conclusion

In this paper, we give a proposition reducing the dimensionality of the set in which the reward is looked for and an experimental result using it. Although promising on a certain light, there still are serious shortcomings before this can be applied in a practical IRL algorithm. Directions for further work in order to remove these shortcomings are explained.

References

[1] R. Sutton and A. Barto, *Reinforcement learning*. MIT Press, 1998.

[2] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*. Morgan Kaufmann Publishers Inc., 2000, pp. 663–670.

[3] A. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning (ICML)*, 1999, pp. 278–287.

[4] S. Russell, "Learning agents for uncertain environments (extended abstract)," in *Annual Conference on Computational Learning Theory*. ACM, 1998, p. 103.

[5] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*. ACM, 2004, p. 1.

[6] G. Neu and C. Szepesvári, "Training parsers by inverse reinforcement learning," *Machine learning*, vol. 77, no. 2, pp. 303–337, 2009.

[7] U. Syed and R. Schapire, "A game-theoretic approach to apprenticeship learning," *Annual Conference on Neural Information Processing Systems (NIPS)*, vol. 20, pp. 1449–1456, 2008.

[8] U. Syed, M. Bowling, and R. Schapire, "Apprenticeship learning using linear programming," in *International Conference on Machine Learning (ICML)*. ACM, 2008, pp. 1032–1039.

[9] A. Boularias and B. Chaib-Draa, "Bootstrapping Apprenticeship Learning," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

[10] N. Ratliff, J. Bagnell, and M. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning (ICML)*. ACM, 2006, p. 736.

[11] N. Ratliff, D. Bradley, J. Bagnell, and J. Chestnutt, "Boosting structured prediction for imitation learning," *Advances in Neural Information Processing Systems*, vol. 19, p. 1153, 2007.

[12] N. Ratliff, J. Bagnell, and S. Srinivasa, "Imitation learning for locomotion and manipulation," in *International Conference on Humanoid Robots*. IEEE, 2007, pp. 392–397.

[13] G. Neu and C. Szepesvári, "Apprenticeship learning using inverse reinforcement learning and gradient methods," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007, pp. 295–302.

[14] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," *Urbana*, vol. 51, p. 61801, 2007.

[15] U. Chajewska, D. Koller, and D. Ormoneit, "Learning an agent's utility function by observing behavior," in *International Conference on Machine Learning (ICML)*, 2001, pp. 35–42.

[16] C. Dimitrakakis and C. Rothkopf, "Bayesian multitask inverse reinforcement learning," *European Workshop on Reinforcement Learning (EWRL)*, 2011.

[17] C. A. Rothkopf, "Modular models of task based guided behavior," Ph.D. dissertation, University of Rochester, 2008.

[18] B. Ziebart, A. Maas, J. Bagnell, and A. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 1433–1438.

[19] A. Boularias, J. Kober, and Peters, "Relative entropy inverse reinforcement learning," *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 15, pp. 20–27, 2011.

[20] N. Aghasadeghi and T. Bretl, "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals," in *International Conference on Machine Learning (ICML)*, 2011.

[21] M. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

[22] Z. Jin, H. Qian, and M. Zhu, "Gaussian processes in inverse reinforcement learning," in *International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1. IEEE, 2010, pp. 225–230.