Neurocomputing 104 (2013) 83–96

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Apprenticeship learning with few examples

Abdeslam Boularias [a,*], Brahim Chaib-draa [b]

[a] Max Planck Institute for Intelligent Systems, Spemannstraße 41, 72076 Tuebingen, Germany
[b] Computer Science and Software Engineering Department, Laval University, Quebec, Canada G1V 0A6

ARTICLE INFO

Article history:
Received 15 November 2011
Received in revised form
19 October 2012
Accepted 1 November 2012
Communicated by H.R. Karimi
Available online 16 November 2012

Keywords:
Imitation learning
Inverse reinforcement learning
Transfer learning
Bootstrapping

ABSTRACT

We consider the problem of imitation learning when the examples, provided by an expert human, are scarce. Apprenticeship learning via inverse reinforcement learning provides an efficient tool for generalizing the examples, based on the assumption that the expert's policy maximizes a value function, which is a linear combination of state and action features. Most apprenticeship learning algorithms use only simple empirical averages of the features in the demonstrations as a statistics of the expert's policy. However, this method is efficient only when the number of examples is sufficiently large to cover most of the states, or the dynamics of the system is nearly deterministic. In this paper, we show that the quality of the learned policies is sensitive to the error in estimating the averages of the features when the dynamics of the system is stochastic. To reduce this error, we introduce two new approaches for bootstrapping the demonstrations by assuming that the expert is near-optimal and the dynamics of the system is known. In the first approach, the expert's examples are used to learn a reward function and to generate furthermore examples from the corresponding optimal policy. The second approach uses a transfer technique, known as graph homomorphism, in order to generalize the expert's actions to unvisited regions of the state space. Empirical results on simulated robot navigation problems show that our approach is able to learn sufficiently good policies from a significantly small number of examples.

1. Introduction

Modern robots are designed to perform complicated planning and control tasks, such as manipulating objects, navigating in unfamiliar environments, and driving in urban settings. However, manually programming these tasks is almost infeasible in practice due to their high complexity. Markov decision processes (MDPs) provide an efficient tool for handling such tasks with a minimal help from an expert human. The human's help consists in simply specifying an appropriate reward function. Unfortunately, in many practical problems, even specifying a reward function is not easy. In fact, it is often easier to provide examples of a desired behavior than to define the corresponding reward function [14].

Learning policies from demonstration, a.k.a. imitation learning, is a technique that has been widely used in robotics [20,4,15,3,9]. An efficient approach to imitation learning, known as apprenticeship learning via inverse reinforcement learning (IRL) [1], consists in recovering a reward function under which the demonstrated policy is near-optimal, rather than directly mimicking the expert's actions. The learned reward function is then used for finding an optimal policy. Consequently, the expert's actions can be predicted in states that have not been encountered during the demonstration. Unfortunately, as already pointed by Abbeel and Ng [1], recovering a reward function is an ill-posed problem. In fact, the expert's policy can be optimal for an infinite number of reward functions, including degenerate ones. Most of the work on apprenticeship learning focused on solving this particular problem by using different types of regularization and different loss functions [17,16,22,25,10].

In this paper, we focus on another important problem in apprenticeship learning occurring when the number of demonstrations is small. Previous algorithms rely on the assumption that the reward function is a linear combination of state and action features. Therefore, the expected average of rewards, i.e. the value function, of any policy is also a linear combination of the expected average of the features. In particular, the value function of the expert's policy is approximated by a linear combination of the empirical averages of the features, estimated from the demonstration.

Consequently, most of state-of-the-art apprenticeship algorithms [1,23,18,25] work efficiently only when the number of examples is large enough to accurately calculate the feature averages, or when the dynamics of the system is nearly deterministic. However, this is not often verified in practice. In fact, most real-world systems are characterized by stochastic transition

* Corresponding author.
E-mail addresses: abdeslam.boularias@tuebingen.mpg.de, boularias@gmail.com (A. Boularias), chaib@ift.ulaval.ca (B. Chaib-draa).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.11.002

functions. This results in a high variance of the feature values, and a large number of examples is needed for estimating these values. In fact, the features of sampled trajectories may take extreme values that do not accurately represent the average ones when their distribution has a large variance, therefore more examples are needed. Examples of highly stochastic systems include robot locomotion [12], autonomous helicopters [2], and object grasping and manipulation [8]. A large number of examples may also be needed to accurately approximate the average values of the feature counts when the state space is too large. In fact, Hoeffding's inequality shows that the convergence rate of an empirical estimate of a random variable decreases exponentially when it has a large support, which is usually the case when the state space is large.

Moreover, the training examples are often provided by a human expert that needs to repeat the task several times, which can be a laborious process, costing time and money. For instance, Abbeel et al. [2] used a professional helicopter pilot for recording trajectories. In the motor game ball-in-a-cup [7], we were able to record only 17 expert trajectories given the difficulty of performing the task while holding the robot arm. In general, it would be much easier to teach robots using as few examples as possible. Therefore, we are interested in developing apprenticeship learning techniques that work well with few training examples.

We propose two bootstrapping techniques for approximating the expected values of the features using a small number of examples. These methods take advantage of the fact that the expert's partially demonstrated policy is near-optimal, and generalizes the expert's policy beyond the states that appear in the demonstration. The expected values of the features are then calculated from the generalized policy and used in an apprenticeship learning algorithm.

In the first approach, we propose a modification to the loss functions of known apprenticeship learning algorithms. Given a hypothesized reward function, the modified loss function compares the value of the optimal policy to the value of the best policy that is consistent with the examples.

The second bootstrapping approach uses a transfer learning technique, known as soft homomorphism [21], in order to explicitly generalize the expert's actions to unvisited regions of the state space. The generalized policy can then be used along with the known system dynamics to analytically calculate the expected average values of the features. Contrary to other direct imitation methods, homomorphisms measure the similarity between two states by taking into account all the possible future states.

We show that the proposed techniques improve the performance of two well-known apprenticeship learning algorithms, namely maximum margin planning (MMP) [17], and linear programming apprenticeship learning (LPAL) [23].

In the next two sections, we present a background of Markov decision processes (MDPs) and a brief overview of apprenticeship learning. Then, we provide a theoretical analysis regarding the error in the learned reward function for a particular apprenticeship learning algorithm. The core of the paper is divided into two parts, the first one is related to the analytical bootstrapping approach, while the second presents the transfer-based techniques. We also present empirical evaluations that demonstrate the effectiveness of the suggested methods. The paper concludes with a discussion of the pros and cons of these techniques, as well as the proposed extension of this work. Note that earlier preliminary versions of this work appeared in [5,6].

2. Background

A finite-state Markov decision process (MDP) is a tuple $(\mathcal{S},\mathcal{A},T,R,\alpha,\gamma)$, where: $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, T is a transition matrix ($T(s,a,s')=\Pr(s'|s,a), s,s'\in\mathcal{S}, a\in\mathcal{A}$), R is a reward function ($R(s,a)$ is the reward given for executing action a in state s), α is the initial state distribution, and γ is a discount factor ($\gamma\in[0,1]$). We denote by MDP\R a Markov decision process without a reward function, i.e. a tuple $(\mathcal{S},\mathcal{A},T,\alpha,\gamma)$. We assume that there exists a set of k feature functions f_i such that the reward function R is given by a linear combination of these features with real-valued weights w_i:

$$\forall s\in\mathcal{S},\quad \forall a\in\mathcal{A}: R(s,a)=\sum_{i=0}^{k} w_i f_i(s,a). \tag{1}$$

A deterministic policy π is a function that returns an action $\pi(s)$ for each state s. A stochastic policy π is a probability distribution on the action to be executed in each state, defined as $\pi(s,a)=\Pr(a|s)$. The value $V(\pi)$ of a policy π is the expected sum of rewards that will be received if we follow policy π,

$$V(\pi)=\mathbb{E}_{(s_t,a_t)\sim d_{\pi,t}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\,|\,\alpha,\pi,T\right], \tag{2}$$

where $d_{\pi,t}$ is the distribution on the states and actions at time-step t induced by policy π and the transition matrix. An optimal policy π is one satisfying

$$\pi\in\arg\max_{\pi} V(\pi). \tag{3}$$

The occupancy μ_π of a policy π is the discounted state–action visit distribution, defined as

$$\mu_\pi(s,a)=\mathbb{E}_{(s_t,a_t)\sim d_{\pi,t}}\left[\sum_{t=0}^{\infty}\gamma^t \delta_{s_t,s}\delta_{a_t,a}\,|\,\alpha,\pi,T\right], \tag{4}$$

where δ is the Kronecker delta. We also use $\mu_\pi(s)$ to denote $\sum_a\mu_\pi(s,a)$. The following linear constraints, known as Bellman-flow constraints, are necessary and sufficient for defining an occupancy measure of a policy

$$\begin{cases}\mu_\pi(s)=\alpha(s)+\gamma\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}}\mu_\pi(s',a)T(s',a,s),\\ \mu_\pi(s)=\sum_{a\in\mathcal{A}}\mu_\pi(s,a),\\ \mu_\pi(s,a)\geq 0.\end{cases} \tag{5}$$

A policy π is well-defined by its occupancy measure μ_π, one can interchangeably use π and μ_π to denote a policy. The set of all vectors $\mu_\pi\in\mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ that satisfy Eq. (5) is denoted by $\mathcal{G}$. The expected discounted value of a feature f_i for a policy π is given by

$$\begin{aligned}\nu_{i,\pi}&=\mathbb{E}_{(s_t,a_t)\sim d_{\pi,t}}\left[\sum_{t=0}^{\infty}\gamma^t f_i(s_t,a_t)\,|\,\alpha,\pi,T\right]\\ &=F(i,\cdot)\mu_\pi,\end{aligned} \tag{6}$$

where F is a k by $|\mathcal{S}||\mathcal{A}|$ feature matrix, such that $F(i,(s,a))=f_i(s,a)$. Using this definition, the value of a policy π can be written as a linear function of the feature counts

$$V(\pi)=w^T F\mu_\pi=w^T\nu_\pi, \tag{7}$$

where ν_π is a vector of entries $\nu_{i,\pi}$, for $i=1,\ldots,k$. Therefore, the value of a policy π is determined by ν_π.

3. Apprenticeship learning

3.1. Overview

The aim of apprenticeship learning is to find a policy π that is nearly as good as a policy π^E demonstrated by an expert, i.e. $V(\pi)\geq V(\pi^E)-\epsilon$. The value functions of π and π^E cannot be compared directly, unless a reward function is given. Ng and Russell [14] proposed to first learn a reward function, assuming that the expert's policy is optimal, and then use it to recover the

expert's generalized policy. However, as previously stated, the problem of learning a reward function given an optimal policy is ill-posed [1]. In fact, a large class of reward functions, including all constant functions for instance, may lead to the same optimal policy. To overcome this problem, Abbeel and Ng [1] did not consider recovering a reward function, instead, their algorithm directly returns a policy π with a bounded loss of the value, i.e. $|V(\pi)-V(\pi^E)| \leq \epsilon$, where the value is measured with respect to the expert's reward function in the worst case. Such guarantee can be achieved by finding a policy π such that $\|\nu_\pi-\nu_{\pi^E}\| \leq \epsilon$. This is a direct consequence of the fact that when two policies have similar expected feature values, they also have similar cumulative rewards, assuming that the reward is a linear function of the features. In the next two subsections, we briefly describe two apprenticeship learning algorithms that we build on in this paper. The first one, known as maximum margin planning (MMP) [17], is a robust algorithm that learns a reward function. The second one, known as linear programming apprenticeship learning (LPAL) [23], is a fast algorithm that returns a policy with a bounded loss in the value.

3.2. Maximum margin planning (MMP)

The MMP algorithm returns a vector of weights w, such that the value of the expert's policy π^E (Eq. (7)) is higher than the value of an alternative policy π by a margin that scales with the number of expert's actions that are different from the actions of the alternative policy. This criterion is explicitly specified in the loss function minimized by the MMP algorithm

$$c_q(w) = \left(\max_{\mu \in \mathcal{G}}(w^T F + l)\mu - w^T F \mu_{\pi^E}\right)^q + \frac{\lambda}{2}\|w\|^2. \tag{8}$$

This latter equation corresponds to the loss used in MMP for one MDP domain, where $q \in \{1,2\}$ defines a slack penalization, λ is a regularization parameter, l is a deviation cost vector that can be defined as

$$l(s,a) = 1 - \pi^E(s,a), \tag{9}$$

and $\mathcal{G}$ is the set of occupancy measures, i.e. the set of all vectors μ that satisfy Eq. (5). Notice that when policy π^E is deterministic, then $l(s,a) = 1$ if $\pi^E(s,a) = 0$ and $l(s,a) = 0$ if $\pi^E(s,a) = 1$.

Intuitively, a policy that maximizes the cost-augmented reward vector $(w^T F + l)$ tends to be completely different from the expert's policy, since an additional reward $l(s,a)$ is given for actions that are different from those of the expert. The MMP algorithm minimizes the difference between the value difference, given by $w^T F \mu_{\pi^E} - w^T F \mu$, and the actions difference $l\mu$.

The loss function c_q is convex, but nondifferentiable. Nevertheless, Ratliff et al. [17] showed that c_q can be minimized by using a generalization of the gradient ascent called the *subgradient method*. For a given reward w, a subgradient g_w^q of the objective function is given by

$$g_w^q = q((w^T F + l)\mu^+ - w^T F \mu_{\pi^E})^{q-1} F \Delta_w \mu_{\pi^E} + \lambda w, \tag{10}$$

where

$$\mu^+ \stackrel{def}{=} \arg\max_{\mu \in \mathcal{G}}(w^T F + l)\mu, \quad \Delta_w \mu_{\pi^E} \stackrel{def}{=} \mu^+ - \mu_{\pi^E}.$$

The subgradient g_w^q is used for updating the reward weights w in an iterative loop, $w_{t+1} = w_t - \eta g_{w_t}^q$, until a minimum is reached.

3.3. Linear programming apprenticeship learning (LPAL)

The LPAL algorithm [23] formulates the problem of learning a reward function as a zero-sum game between two players. In this game, one player chooses reward weights w that minimize the value function of a policy π, and the other player chooses an optimal policy π given reward weights w. The proposed solution is based on the following observation: if the reward weights are positive and sum to 1, then $V(\pi)-V(\pi^E) \geq \min_i[\nu_{i,\pi}-\nu_{i,\pi^E}]$, for any policy π ($\nu_{i,\pi}$ is defined in Eq. (6)). LPAL consists in finding a policy that maximizes the margin $\min_i[\nu_{i,\pi}-\nu_{i,\pi^E}]$. The maximum margin is given by the linear program:

$$\max_{\nu \in \mathbb{R}, \mu_\pi \in \mathcal{G}} \nu$$

subject to $\forall i \in \{0,\ldots,k-1\}$:

$$\nu \leq \underbrace{\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_\pi(s,a) f_i(s,a)}_{\nu_{i,\pi}} - \underbrace{\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_{\pi^E}(s,a) f_i(s,a)}_{\nu_{i,\pi^E}}. \tag{11}$$

The learned policy π is given by

$$\pi(s,a) = \frac{\mu_\pi(s,a)}{\sum_{a' \in \mathcal{A}} \mu_\pi(s,a')}. \tag{12}$$

3.4. Empirical estimation of the expected feature values

Notice that both MMP and LPAL require the knowledge of the expected feature values ν_{i,π^E}. These values can be analytically calculated (using Bellman-flow constraints) only when π^E is known for every state. Given a sequence of M demonstration trajectories $t_m = (s_1^m, a_1^m, \ldots, s_H^m, a_H^m,)$, the values ν_{i,π^E} can be empirically estimated as

$$\hat{\nu}_{i,\pi^E} = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{H} \gamma^t f_i(s_t^m, a_t^m). \tag{13}$$

However, there are many issues related to this approximation. First, the estimated values $\hat{\nu}_{i,\pi^E}$ can be very different from the true ones when the demonstration trajectories are scarce. Second, $\hat{\nu}_{i,\pi^E}$ is estimated for a finite horizon H, whereas $\nu_{i,\pi}$, used in the objective function (Eqs. (8) and (11)), is calculated for an infinite horizon (Eq. (5)). In practice, these two values have different scales and cannot be compared as done in these loss functions. Finally, the values ν_{i,π^E} are a function of both a policy and the transition probabilities, Eq. (13) does not take advantage of the fact that the transition probabilities are known.

4. Empirical error analysis

4.1. A Hoeffding bound on the empirical estimation error

In this subsection, we derive an upper bound on the error in estimating the feature averages $\hat{\nu}_{i,\pi^E}$ using the Monte Carlo estimator given by Eq. (13).

Theorem 1. *Let M be the number of trajectories in the demonstration, H be the length of each trajectory, $f_i^- = \min_{s \in \mathcal{S}, a \in \mathcal{A}} f_i(s,a)$, $f_i^+ = \max_{s \in \mathcal{S}, a \in \mathcal{A}} f_i(s,a)$ and $\epsilon \in \mathbb{R}$, then*

$$\Pr(|\hat{\nu}_{i,\pi^E} - \nu_{i,\pi^E}| \geq \epsilon) \leq 2 \exp\left(-\frac{2M((1-\gamma)\epsilon - \gamma^H f_i^+)^2}{(1-\gamma^H)^2 (f_i^+ - f_i^-)^2}\right). \tag{14}$$

Proof. Let $d_{\pi^E,t}$ be the distribution on the states and the actions at time-step t induced by the policy π^E and the transition matrix. Let us denote $|\hat{\nu}_{i,\pi^E} - \nu_{i,\pi^E}|$ by δ. From Eq. (13), we have

$$\delta \leq \sum_{t=1}^{H} \gamma^t \underbrace{\left|\frac{1}{M} \sum_{m=1}^{M} f_i(s_t^m, a_t^m) - \mathbb{E}_{d_{\pi^E,t}} f_i(s,a)\right|}_{\Delta_t} + \frac{\gamma^H}{1-\gamma} f_i^+.$$

From Hoeffding's inequality, we have $\forall \xi \in \mathbb{R}$,

$$\Pr(\Delta_t \geq \xi) \leq 2 \exp\left(-\frac{2M\xi^2}{(f_i^+ - f_i^-)^2}\right).$$

Since the errors Δ_t for different time-steps t are independent given the state–action distributions $d_{\pi^E,t}$, then

$$\Pr\left(\delta \geq \sum_{t=1}^{H} \gamma^t \xi + \frac{\gamma^H}{1-\gamma} f_i^+\right) \leq 2 \exp\left(-\frac{2M\xi^2}{(f_i^+ - f_i^-)^2}\right).$$

The bound is given by setting $\xi = ((1-\gamma)\epsilon - \gamma^H f_i^+)/(1-\gamma^H)$. □

This bound indicates that the probability of making an error within a given threshold increases as the range $(f_i^+ - f_i^-)$ of the estimated feature gets larger.

4.2. An analysis of the reward error in maximum margin planning

To show the effect of the empirical feature estimation error on the quality of the learned rewards, we present an analysis of the distance between the vector of reward weights $\hat{w}$ returned by MMP with empirical feature counts $\hat{\nu}_{\pi^E}$, calculated from the examples by solving Eq. (13), and the vector w^* returned by MMP with accurate values ν_{π^E}, calculated by using Eq. (5). We adopt the following notations: $\Delta\nu_\pi = \hat{\nu}_{\pi^E} - \nu_{\pi^E}$, $\Delta w = \hat{w} - w^*$, and $V_l(w) = \max_{\mu \in \mathcal{G}}(w^T F + l)\mu$, and we consider $q=1$ in Eq. (8). The following theorem shows how the reward error Δw is related to the feature count error $\Delta\nu_\pi$. Due to the fact that the loss function of the MMP algorithm is piecewise defined, one cannot find a simple relation between Δw and $\Delta\nu_\pi$. However, we show that for any $\hat{w} \in \mathbb{R}^k$, there is an upper bound function f such that for any $\epsilon \in \mathbb{R}$, if $\|\Delta\nu_\pi\|_2 < f(\hat{w},\epsilon)$ then $\|\Delta w\|_2 \leq \epsilon$.

Theorem 2. *Let $\epsilon \in \mathbb{R}$, if the following condition is verified*:

$$\|\Delta\nu_\pi\|_2 < \min_{w, \|w-\hat{w}\|_2 = \epsilon} \frac{V_l(w) - V_l(\hat{w}) + (\hat{w}-w)^T \hat{\nu}_{\pi^E} + \frac{\lambda}{2}(\|w\|_2 - \|\hat{w}\|_2)}{\epsilon} \tag{15}$$

then $\|\Delta w\|_2 \leq \epsilon$.

Proof. The condition stated in this theorem implies

$$\|\hat{w}-w\|_2 \|\Delta\nu_\pi\|_2 < V_l(w) - V_l(\hat{w}) + (\hat{w}-w)^T \hat{\nu}_{\pi^E} + \frac{\lambda(\|w\|_2 - \|\hat{w}\|_2)}{2}.$$

From Hölder's inequality, we have

$$(\hat{w}-w)^T \Delta\nu_\pi < V_l(w) - V_l(\hat{w}) + (\hat{w}-w)^T \hat{\nu}_{\pi^E} + \frac{\lambda(\|w\|_2 - \|\hat{w}\|_2)}{2}.$$

Then

$$V_l(\hat{w}) - \left(\hat{w}^T \nu_{\pi^E} - \frac{\lambda}{2}\|\hat{w}\|_2\right) < V_l(w) - \left(w^T \nu_{\pi^E} - \frac{\lambda}{2}\|w\|_2\right).$$

In other terms, the point $(\hat{w}^T \nu_{\pi^E} - (\lambda/2)\|\hat{w}\|_2)$ is closer to V_l than any other point $(w^T \nu_{\pi^E} - (\lambda/2)\|w\|_2)$, where w is a point on the sphere centered around $\hat{w}$ with a radius of ϵ. Since V_l is convex and $(w^{*T} \nu_{\pi^E} - (\lambda/2)\|w^*\|_2)$ is by definition the closest point to the set of hyperplans V_l, then w^* should be inside the ball centered around $\hat{w}$ with a radius of ϵ. Therefore, $\|w^* - \hat{w}\|_2 \leq \epsilon$ and thus $\|\Delta w\|_2 \leq \epsilon$. □

Consequently, the reward error $\|\Delta w\|_2$ approaches zero as the error of the estimated feature values $\|\Delta\nu_\pi\|_2$ approaches zero. A simpler bound can be easily derived given admissible heuristics of V_l.

Corollary. *Let $\underline{V_l}$ and $\overline{V_l}$ be, respectively, a lower and an upper bound on V_l, then Theorem 2 holds if $V_l(w) - V_l(\hat{w})$ is replaced by $\underline{V_l}(w) - \overline{V_l}(\hat{w})$.*

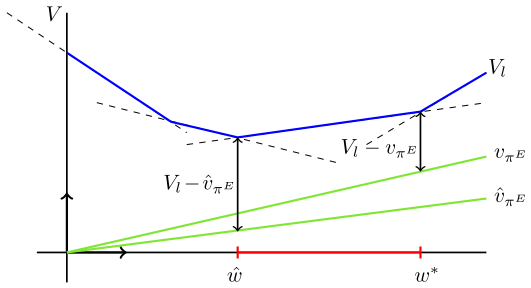

Fig. 1. Reward error in the MMP algorithm using approximate feature counts $\hat{\nu}_{\pi^E}$. We indicate by ν_{π^E} (resp. $\hat{\nu}_{\pi^E}$) the linear function defined by the vector ν_{π^E} (resp. $\hat{\nu}_{\pi^E}$), w^* denotes the true reward weights and $\hat{w}$ denotes the learned ones.

Fig. 1 illustrates the divergence from the optimal reward weights w^* when approximate feature values are used. The error is not a continuous function of $\Delta\nu_\pi$ when the cost function is not regularized, because the vector returned by MMP is always a fringe point. Informally, the error is proportional to the maximum subgradient of $V_l - \nu_{\pi^E}$ at the fringe point w^*.

Finally, note that Theorem 2 can be used along with the Hoeffding bound given in Theorem 1 to derive a probabilistic bound that links the reward error to the number of examples, as shown in the following theorem.

Theorem 3. *Let M be the number of trajectories in the demonstration, H be the length of each trajectory, $f_i^- = \min_{s \in \mathcal{S}, a \in \mathcal{A}} f_i(s,a)$, $f_i^+ = \max_{s \in \mathcal{S}, a \in \mathcal{A}} f_i(s,a)$ and $\epsilon \in \mathbb{R}$, and $\xi = \min_{w, \|w-\hat{w}\|_2 = \epsilon}($ $V_l(w) - V_l(\hat{w}) + (w-w)^T \hat{\nu}_{\pi^E} + (\lambda/2)(\|w\|_2 - \|\hat{w}\|_2))/\epsilon$, then*

$$\Pr(\|\Delta w\|_2 \geq \epsilon) \leq 2 \exp\left(-\min_i \frac{2M((1-\gamma)\xi - \gamma^H f_i^+)^2}{(1-\gamma^H)^2 (f_i^+ - f_i^-)^2}\right). \tag{16}$$

Proof. From Theorem 2, we have $\Pr(\|\Delta w\|_2 \geq \epsilon) \leq \Pr(\|\Delta\nu_\pi\|_2 \geq \xi)$ (in general, $(A \Rightarrow B) \Rightarrow (\Pr(A) \leq \Pr(B))$). Note that $\|\Delta\nu_\pi\|_2 \leq \max_i |\hat{\nu}_{i,\pi^E} - \nu_{i,\pi^E}|$. Therefore, $\Pr(\|\Delta w\|_2 \geq \epsilon) \leq \max_i \Pr(|\hat{\nu}_{i,\pi^E} - \nu_{i,\pi^E}| \geq \xi)$. From the bound given in Theorem 1, we have $\max_i \Pr(|\hat{\nu}_{i,\pi^E} - \nu_{i,\pi^E}| \geq \xi) \leq 2\exp(-\min_i(2M((1-\gamma)\xi - \gamma^H f_i^+)^2/(1-\gamma^H)^2 (f_i^+ - f_i^-)^2))$. □

Such a bound will not be of a practical use since there is no straightforward easy way for calculating the right term in Eq. (16). However, this bound clearly shows that the probability of an error $\|\Delta w\|_2$ higher than a given ϵ increases with the margins $f_i^+ - f_i^-$. Therefore, one might need more training examples when the features vary a lot from a state to another.

5. Analytical bootstrapping of apprenticeship learning

In this section, we show how one can avoid the use of the empirical feature values in the loss functions of MMP and LPAL by adopting modified loss functions.

5.1. Assumptions

We consider the standard assumptions of inverse reinforcement learning: (i) the dynamics of the system is known, and (ii) the expert is near-optimal [14]. The first assumption is fundamental since most of the algorithms in the literature are model-based (exceptions include [7]). If the dynamics is unknown, then one can apply system identification techniques to construct a model by using the trajectories provided by the expert, or by directly interacting with the system and collecting additional trajectories [2]. However, inaccurate models will have an effect on the learned reward that is comparable to using a Monte Carlo

estimator as discussed in the previous section. In this paper, we only consider the case where the dynamics model is accurate.

The assumption regarding the optimality of the expert's policy is ill-defined, since any policy is optimal with respect to a certain reward function. For instance, all the policies are optimal with respect to a zero (or a constant) reward function. In fact, only the reward features f_i in Eq. (1) have a physical interpretation, the reward weights w_i reflect subjective preferences for certain features. Therefore, the reward function cannot be measured by using system identification methods. This problem can be addressed only by incorporating prior knowledge in the loss function. For example, the MMP algorithm returns a reward function that maximizes the distance between the value of the expert's policy and the second best policy.

A better interpretation of the optimality assumption would be that the expert is *rational*, i.e. the expert's actions are chosen according to Eqs. (1)–(3) for some unknown weights w_i. Consequently, the reward weights that explain the choice of actions in some states can be used to predict the actions in other states. Inverse reinforcement learning consists in learning weights w_i that make the expert's policy optimal. If the expert's choice of actions is sub-optimal with respect to some ideal reward function, then the generalized policy also will be sub-optimal.

5.2. Bootstrapping maximum margin planning

The empirical feature error $\Delta\nu_\pi$ can be significantly reduced by using the known transition function for calculating $\hat{\nu}_{\pi^E}$ and solving the flow equation (5), instead of the Monte Carlo estimator (Eq. (13)). However, this cannot be done unless the complete expert's policy π^E is provided.

Assuming that the expert's policy π^E is optimal, the value $w^TF\mu_{\pi^E}$ in Eq. (8) can be replaced by $\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$, the value of the optimal policy, according to the hypothesized reward weight w, that selects the same actions as the expert in all the states that occurred in the demonstration. The loss function of the bootstrapped maximum margin planning becomes

$$c_q(w)=\left(\max_{\mu\in\mathcal{G}}(w^TF+l)\mu-\max_{\mu'\in\mathcal{G}_{\pi^E}} w^TF\mu'\right)^q+\frac{\lambda}{2}\|w\|^2, \tag{17}$$

where $\mathcal{G}_{\pi^E}$ is the set of vectors μ_π, subject to the following modified Bellman-flow constraints

$$\begin{cases}\mu_\pi(s)=\alpha(s)+\gamma\sum_{s'\in\mathcal{S}^E}\mu_\pi(s')\sum_{a\in\mathcal{A}}\pi^E(s',a)T^a(s',s)\\ \quad+\gamma\sum_{s'\in\mathcal{S}\setminus\mathcal{S}^E}\sum_{a\in\mathcal{A}}\mu_\pi(s',a)T^a(s',s),\\ \mu_\pi(s)=\sum_{a\in\mathcal{A}}\mu_\pi(s,a),\\ \mu_\pi(s,a)\geq 0.\end{cases} \tag{18}$$

$\mathcal{S}^E$ is the set of states where the expert's policy is known, i.e. the set of states that appeared in the training examples.

Unfortunately, the new loss function (Eq. (17)) is not necessarily convex. In fact, it corresponds to a margin between two convex functions: the value of the bootstrapped expert's policy $\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$ and the value of the best alternative policy $\max_{\mu\in\mathcal{G}}(w^TF+l)\mu$. Yet, a local optimal solution of this modified loss function can be found by using the same subgradient as in Eq. (10), and replacing μ_{π^E} by $\arg\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$. In practice, as we will show in the experimental analysis, the solution returned by the bootstrapped MMP outperforms the solution of MMP wherein the feature values of the expert's policy are calculated without taking into account the known transition probabilities. This improvement is particularly pronounced in highly stochastic environments. The computational cost of minimizing this modified loss function is twice the one of MMP, since two optimal policies are found at each iteration. The computational complexity of calculating a subgradient of Eq. (17) is dominated by the complexity of finding the vectors μ and μ' that maximize the corresponding reward functions, which can be casted as a linear program and solved in $\mathcal{O}((|\mathcal{S}||\mathcal{A}|)^{3.5})$. Therefore, the complexity of calculating a subgradient of Eq. (17) is $\mathcal{O}(2(|\mathcal{S}||\mathcal{A}|)^{3.5})$. One should consider the trade-off between the quality of the learned reward and the computational resources that are available in order to choose between MMP and the bootstrapped MMP.

In the remainder of this subsection, we provide a theoretical analysis of the new loss function (Eq. (17)). For the sack of simplicity, we consider $q=1$ and $\lambda=0$.

Theorem 4. *The loss function c_q, defined by Eq. (17), has at most $|\mathcal{A}|^{|\mathcal{S}|}/|\mathcal{A}|^{|\mathcal{S}^E|}$ different local minima.*

Proof. If $q=1$ and $\lambda=0$, then the cost $c_q(w)$ corresponds to a distance between the convex and piecewise linear functions $\max_{\mu\in\mathcal{G}}(w^TF+l)\mu$ and $\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$. Therefore, for any vector $\mu'\in\mathcal{G}_{\pi^E}$, the function c_q is monotone in the interval of w where μ' is optimal, i.e. where $w^TF\mu'=\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$. Consequently, the number of local minima of the function c_q is at most equal to the number of optimal vectors μ in $\mathcal{G}_{\pi^E}$, which is upper bounded by the number of deterministic policies defined on $S\setminus\mathcal{S}^E$, i.e. by $|\mathcal{A}|^{|\mathcal{S}|-|\mathcal{S}^E|}$. □

Consequently, the number of different local minima of the function c_q decreases as the number of states covered by the demonstration increases. Ultimately, the function c_q becomes convex when the examples cover all the possible states.

Theorem 5. *If there exists a reward weight vector $w^*\in\mathbb{R}^k$, such that the expert's policy π^E is the only optimal policy with w^*, i.e.* $\arg\max_{\mu\in\mathcal{G}} w^{*T}F\mu=\{\mu_{\pi^E}\}$, *then there exists $\alpha>0$ such that*: (i) *the expert's policy π^E is the only optimal policy with αw^*, and* (ii) *$c_q(\alpha w^*)$ is a local minimum of the function c_q, defined in Eq. (17).*

Proof. The subgradients of the function c_q at a point $w\in\mathbb{R}^k$ correspond to the vectors $F\mu'-F\mu''$, with $\mu'\in\arg\max_{\mu\in\mathcal{G}}(w^TF+l)\mu, \mu''\in\arg\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$. In order that $c_q(w)$ will be a local minimum, it suffices to ensure that $\vec{0}\in\nabla_w c_q(w)$, i.e. $\exists\ \mu'\in\arg\max_{\mu\in\mathcal{G}}(w^TF+l)\mu, \exists\ \mu''\in\arg\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$ such that $F\mu'=F\mu''$. Let $w^*\in\mathbb{R}^k$ be a reward weight vector such that π^E is the only optimal policy, and let $\epsilon=w^{*T}F\mu_{\pi^E}-w^{*T}F\mu'$ where $\mu'\in\arg\max_{\mu\in\mathcal{G}-\{\mu_{\pi^E}\}} w^{*T}F\mu$. Then, $2|\mathcal{S}^E|/(1-\gamma)=\alpha w^{*T}F\mu_{\pi^E}-\alpha w^{*T}F\mu'$, where $\alpha=2|\mathcal{S}^E|/\epsilon(1-\gamma)$. Notice that by multiplying w^* by $\alpha>0$, π^E remains the only optimal policy, i.e. $\arg\max_{\mu\in\mathcal{G}}\alpha w^{*T}F\mu=\{\mu_{\pi^E}\}$, and $\mu'\in\arg\max_{\mu\in\mathcal{G}-\{\mu_{\pi^E}\}}\alpha w^{*T}F\mu$. Therefore, it suffices to show that $\mu_{\pi^E}\in\arg\max_{\mu\in\mathcal{G}}(\alpha w^{*T}F+l)\mu$. Indeed, $\max_{\mu\in\mathcal{G}-\{\pi^E\}}(\alpha w^{*T}F+l)\mu\leq\max_{\mu\in\mathcal{G}-\{\pi^E\}}\alpha w^{*T}F\mu+\max_{\mu\in\mathcal{G}-\{\pi^E\}} l\mu\leq(\alpha w^{*T}F\mu_{\pi^E}-2|\mathcal{S}^E|/(1-\gamma))+|\mathcal{S}^E|/(1-\gamma)\leq\alpha w^{*T}F\mu_{\pi^E}-|\mathcal{S}^E|/(1-\gamma)$. Therefore, $\mu_{\pi^E}\in\arg\max_{\mu\in\mathcal{G}}(\alpha w^{*T}F+l)\mu$. □

Theorem 6. *If F is an identity matrix, then all the local minima of the cost function defined by Eq. (17) are equal to*0.

Proof. If $c_q(w)$ is a local minimum, then $\vec{0}\in\nabla_w c_q(w)$, where $\nabla_w c_q(w)$ denotes the set of subgradients of the cost function at point w, and $\vec{0}\in\mathbb{R}^k$ is the zero vector. The subgradients correspond to the vectors $F\mu'-F\mu''$, with $\mu'\in\arg\max_{\mu\in\mathcal{G}}(w^TF+l)\mu, \mu''\in\arg\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$. The condition $\vec{0}\in\nabla_w c_q(w)$ implies that $\exists\mu'\in\arg\max_{\mu\in\mathcal{G}}(w^TF+l)\mu$, $\exists\mu''\in\arg\max_{\mu\in\mathcal{G}_{\pi^E}} w^TF\mu$ such that $F\mu'=F\mu''$. Since F is an identity, then $F\mu'=\mu'$ and $F\mu''=\mu''$, therefore $\mu'=\mu''$. Notice that since $\mu'\in\mathcal{G}_{\pi^E}$ then $l\mu'=0$ given that l takes nonzero values only in the state–actions that are different from π^E. Consequently, $(w^TF+l)\mu'=(w^TF)\mu'$ and $c_q(w)=0$. □

This last theorem shows that the nonconvexity of the modified loss function is actually not a problem when there is a distinctive feature for each state–action. In that case, all the local minima are equal. However, this leads to a larger uncertainty regarding the reward function used by the expert, since a large number of reward vectors w will minimize the modified loss function.

5.3. Bootstrapping linear programming apprenticeship learning

As for MMP, the feature counts v_{i,π^E} in LPAL can be analytically calculated only when a complete policy π^E of the expert is provided. Alternatively, the same error bound $V(\pi) \geq V(\pi^E)+\nu$ can be guaranteed by ensuring that $\nu = \min_{i=0,\ldots,k-1} \min_{\pi' \in \Pi^E} [v_{i,\pi} - v_{i,\pi'}]$, where Π^E denotes the set of all the policies that select the same actions as the expert in all the states that occurred in the demonstration (in LPAL, the expert's policy is not necessarily optimal). Instead of enumerating all the policies of the set Π^E in the constraints, note that

$$\min_{i=0,\ldots,k-1} \min_{\pi' \in \Pi^E} [v_{i,\pi} - v_{i,\pi'}] \leq \min_{i=0,\ldots,k-1} [v_{i,\pi} - \max_{\pi' \in \Pi^E} v_{i,\pi'}]. \tag{19}$$

Therefore, LPAL can be reformulated as maximizing $\min_{i=0,\ldots,k-1}[v_{i,\pi} - v_i^E]$, where $v_i^E = \max_{\pi' \in \Pi^E} v_{i,\pi'}$ for each feature i. The maximal margin is found by solving the following linear program:

$$\max_{\nu,\mu_\pi} \nu$$

subject to $\forall i \in \{0,\ldots,k-1\}$:

$$\nu \leq \underbrace{\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu_\pi(s,a) f_i(s,a)}_{v_{i,\pi}} - \underbrace{\sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu_{i,\pi'}(s,a) f_i(s,a)}_{v_i^E},$$

$$\mu_\pi(s) = \alpha(s) + \gamma \sum_{s'\in\mathcal{S}} \sum_{a\in\mathcal{A}} \mu_\pi(s',a) T(s',a,s),$$

$$\sum_{a\in\mathcal{A}} \mu_\pi(s,a) = \mu_\pi(s), \mu_\pi(s,a) \geq 0, \tag{20}$$

where the margins v_i^E are found by solving k separate optimization problems (k is the number of features). For each feature i, v_i^E is the value of the optimal policy in the set Π^E under the reward weight w defined as $w_i=1$ and $w_j=0$, $\forall j \neq i$.

The analytical bootstrapping methods that we described in this section consist in modifying the loss function of two typical apprenticeship learning algorithms. The expected feature values of a hypothesized optimal policy are compared to the ones of the best policy that chooses the same actions as the expert in all the states that appeared in the examples. In the next section, we describe a more explicit scheme for generalizing the examples by using graph homomorphism.

6. Apprenticeship learning with graph homomorphism

In this section, we propose another solution for overcoming the problems resulting from the empirical errors Δv_π. We use a transfer learning technique, known as soft homomorphism [21], in order to generalize the expert's actions to unvisited regions of the state space. The generalized policy can then be used to analytically calculate the expected values of the features.

This type of imitation learning techniques, called *behavioral cloning*, does not make any assumption regarding the optimality, or even the rationality, of the expert. If the expert's examples are sub-optimal, then the generalized policy will also be sub-optimal as well.

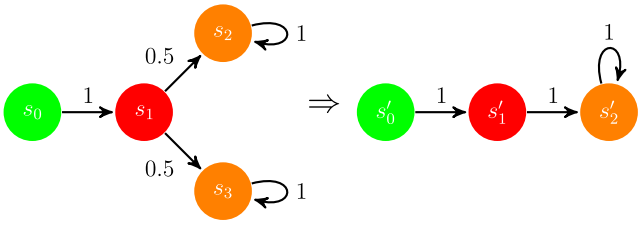

Fig. 2. Example of a homomorphism of a Markov decision process.

6.1. Transfer learning through MDP homomorphism

Transfer learning refers to the problem of using a policy learned for performing some task in order to perform a different, but related, task. The related task may be defined in a new domain, or in the same domain but in a different region of the state space. This problem has been widely studied in the context of reinforcement learning. An overview of the literature on transfer learning is out of the scope of this paper, the interested reader might find an extended overview in [24].

We will focus on a transfer method known as MDP homomorphism [19]. A homomorphism from an MDP $\mathcal{M}=(\mathcal{S},\mathcal{A},T,R,\alpha,\gamma)$ to another MDP $\mathcal{M}'=(\mathcal{S}',\mathcal{A},T',R',\alpha,\gamma)$ is a surjective function f, called the transfer function, defined as

$$f: \mathcal{S} \to \mathcal{S}'$$
$$\forall s\in\mathcal{S}, s'\in\mathcal{S}', \forall a\in\mathcal{A}: T'(f(s),a,s') = \sum_{\substack{s''\in\mathcal{S} \\ f(s'')=s'}} T(s,a,s''). \tag{21}$$

In other terms, a homomorphism can be seen as a transformation of an MDP that preserves its dynamics. Fig. 2 shows an example of a homomorphism f from an MDP with four states $\{s_0,s_1,s_2,s_3\}$ to another one with three states $\{s'_0,s'_1,s'_2\}$, where $f(s_0)=s'_0$, $f(s_1)=s'_1$, and $f(s_2)=f(s_3)=s'_2$. In order to make the example simpler, we considered only one action a in this example. Note that $T(s'_0,a,s'_1)=T(s_0,a,s_1)$, $T(s'_1,a,s'_2)=T(s_1,a,s_2)+T(s_1,a,s_3)$ (both s_2 and s_3 have as image s'_2), and $T(s'_2,a,s'_2)=T(s_2,a,s_2)+\underbrace{T(s_2,a,s_3)}_{0}=T(s_3,a,s_3)+\underbrace{T(s_3,a,s_2)}_{0}$.

Unfortunately, finding a homomorphism is an NP-complete combinatory search problem. In this section, we will consider another variant of this approach known as Soft MDP homomorphism [21]. The core idea of this latter method consists in finding a transfer function f that maps each state of an MDP model $\mathcal{M}=(\mathcal{S},\mathcal{A},T,R,\alpha,\gamma)$ to a probability distribution on the states of another MDP model $\mathcal{M}'=(\mathcal{S}',\mathcal{A},T',R',\alpha,\gamma)$. Additionally, the mapping between the states of $\mathcal{S}$ and $\mathcal{S}'$ should preserve the transition probabilities,

$$f: \mathcal{S}\times\mathcal{S}' \to [0,1], \forall s\in\mathcal{S}, s'\in\mathcal{S}', \forall a\in\mathcal{A}:$$
$$\sum_{s''\in\mathcal{S}} T(s,a,s'') f(s'',s') = \sum_{s''\in\mathcal{S}'} f(s,s'') T'(s'',a,s'). \tag{22}$$

The reward function also should be preserved, but we will not consider this constraint since, in the context of this paper, the reward function is unknown. Sorg and Singh [21] showed that soft homomorphisms can be used to transfer the values of policies from an MDP model to another. In the next section, we show how to use soft homomorphisms in order to transfer actions from a subset of a state space to another subset of the same space.

6.2. Generalizing policies with local homomorphisms

Given an MDP model without reward $\mathcal{M}=(\mathcal{S},\mathcal{A},T,\alpha,\gamma)$ and a set of M trajectories provided by an expert, the state space $\mathcal{S}$ can be divided into two subsets: $\mathcal{S}^E$, the set of states that appear in the

provided trajectories, and $\mathcal{S}\backslash\mathcal{S}^E$. For the states of $\mathcal{S}^E$, the expert's policy π^E can be directly inferred from the trajectories if it is deterministic, or estimated by calculating the frequencies of the actions if it is stochastic. We consider the general case and use $\hat{\pi}^E$ to denote the estimated expert's policy.

In order to generalize the policy $\hat{\pi}^E$ to $\mathcal{S}\backslash\mathcal{S}^E$, we first create a restrained MDP without a reward function $\mathcal{M}^E=(\mathcal{S}^E,\mathcal{A},T^E,\alpha,\gamma)$, where the transition function T^E is defined as

$$\forall s,s'\in\mathcal{S}^E\backslash\{s\},\forall a\in\mathcal{A}:\begin{cases}T^E(s,a,s')=T(s,a,s'),\\ T^E(s,a,s)=T(s,a,s)+\sum_{s''\in\mathcal{S}\backslash\mathcal{S}^E}T(s,a,s'').\end{cases}$$

This function ensures that all the transitions remain within the states of $\mathcal{S}^E$ by assuming that any action that leads to a state outside of $\mathcal{S}^E$ has no effect.

The next step consists in finding a lossy soft homomorphism between $\mathcal{M}$ and $\mathcal{M}^E$, where the objective function corresponds to the error in preserving the transition probabilities. The transfer function f of this homomorphism is found by solving a linear program. Function f can be seen as a measure of similarity between two states. One can use this measure in order to define the generalized policy $\hat{\pi}^E$ as follows:

$$\forall s\in\mathcal{S}\backslash\mathcal{S}^E,\forall a\in\mathcal{A}:\hat{\pi}^E(s,a)=\sum_{s'\in\mathcal{S}^E}f(s,s')\hat{\pi}^E(s',a). \tag{23}$$

However, this method scales up poorly with respect to the number of states visited by the expert and the number of states in the corresponding domain. This is due to the fact that $|\mathcal{S}^E|\times|\mathcal{S}|$ variables are used in the linear program. To improve the computational efficiency of this approach, we redefine the function f as a measure of local similarity between two states. We denote by s^d the set of states that can be reached from state s within a distance of d time-steps, and by $\mathcal{M}^{s,d}(s^d,\mathcal{A},T^d_s,\alpha,\gamma)$ the MDP| R defined on these states. The transition function T^d_s is then defined as

$$\forall s,s'\in s^d\backslash\{s\},\forall a\in\mathcal{A}:\begin{cases}T^d_s(s,a,s')=T(s,a,s'),\\ T^d_s(s,a,s)=T(s,a,s)+\sum_{s''\in\mathcal{S}\backslash s^d}T(s,a,s'').\end{cases}$$

Given a distance d and a threshold ϵ, two states s and s' are considered as locally similar if there exists a soft homomorphism between $\mathcal{M}^{s,d}$ and $\mathcal{M}^{s',d}$ with a transfer error not greater than ϵ. This property is checked by solving the following linear program:

$$\min_f e$$

subject to $\forall s_i\in s^d,s_k\in s'^d,\forall a\in\mathcal{A}$:

$$\left|\sum_{s_j\in s^d}T^d_s(s_i,a,s_j)f(s_j,s_k)-\sum_{s_j\in s'^d}f(s_i,s_j)T^d_{s'}(s_j,a,s_k)\right|\le e,$$

$$f(s_i,s_k)\ge 0,\ \sum_{s_k\in s'^d}f(s_i,s_k)=1. \tag{24}$$

The principal steps of our approach are summarized in Algorithm 1, with a high-level view given in the diagram of Fig. 3. For every state $s\in\mathcal{S}\backslash\mathcal{S}^E$, we create the list s^t of neighbor states that can be reached from s within t steps (steps 1–6). The distance t is gradually increased until we find a state $s'\in s^t\cap\mathcal{S}^E$ that is locally similar to s (step 7). If $s^t=s^{t-1}$, i.e. all the states that can be reached from s are already contained in s^{t-1}, and no one is locally similar to s (step 8), then we set $\hat{\pi}^E(s,a)$ to a uniform distribution (step 9), and a stopping boolean variable c is set to true (step 21). Otherwise, for each action a, $\hat{\pi}^E(s,a)$ is proportional to the weighted votes for a of the states that are locally similar to state s (steps 16 and 23). The generalized policy $\hat{\pi}^E$ can be either considered as the robot's policy or used to calculate the feature values $\hat{v}_{i,\pi^E}$ for an apprenticeship learning algorithm.

Algorithm 1. Bootstrapping Apprenticeship Learning via Soft Local Homomorphisms.

Input: An MDP model without reward $(\mathcal{S},\mathcal{A},T,\alpha,\gamma)$, a set of demonstration trajectories, an error threshold ϵ, and a similarity distance d;

1 Let $\mathcal{S}^E$ be the set of states contained in the demonstration trajectories;

2 Use the demonstration trajectories to estimate the policy $\hat{\pi}^E$ for the states of $\mathcal{S}^E$;

3 Let s^t be the set of states that can be reached from a state s within t steps, *votes* a vector containing the number of votes per action, and c the stopping condition;

4 **foreach** $s\in\mathcal{S}\backslash\mathcal{S}^E$ **do**

5 $t\leftarrow 0, s^0\leftarrow\{s\}, votes\leftarrow(0,\ldots,0), c\leftarrow$ false;

6 **repeat**

7 $t\leftarrow t+1$;

8 **if** $s^t=s^{t-1}$ **then**

9 $c\leftarrow$ true, $votes\leftarrow(1,\ldots,1)$;

10 **else**

11 **foreach** $s'\in s^t\cap\mathcal{S}^E$ **do**

12 Let e be the error returned by the linear program (24) on $(\mathcal{M}^{s,d},\mathcal{M}^{s',d})$

13 **if** $e\le\epsilon$ **then**

14 $c\leftarrow$ true;

15 **foreach** $a\in\mathcal{A}$ **do**

16 $votes(a)\leftarrow votes(a)+\hat{\pi}^E(s',a)$

17 **end**

18 **end**

19 **end**

20 **end**

21 **until** $c=true$;

22 **foreach** $a\in\mathcal{A}$ **do**

23 $\hat{\pi}^E(s,a)=\frac{votes(a)}{\sum_{a'\in\mathcal{A}}votes(a')}$;

24 **end**

25 **end**

Output A generalized policy $\hat{\pi}^E$;

Note that the proposed scheme is generic and other transfer|-classification methods can also be used for generalizing the provided examples. In the next section, we will show that combining this particular transfer method with other apprenticeship algorithms leads to a significant improvement in the quality of the learned policies.

7. Experimental results

To validate the proposed techniques, we performed experiments on two simulated navigation domains. The first one is a gridworld problem. While this is not meant to be a challenging task, it allows us to compare our approach to other methods of generalizing the expert's policy when the number of demonstrations is small. The second domain corresponds to a racetrack.

7.1. Gridworld setting

We consider multiple $x\times x$ gridworld domains taken from [1], with x taking the following values: 16, 24, 32, and 48. The state of the robot corresponds to its position on the grid, therefore, the

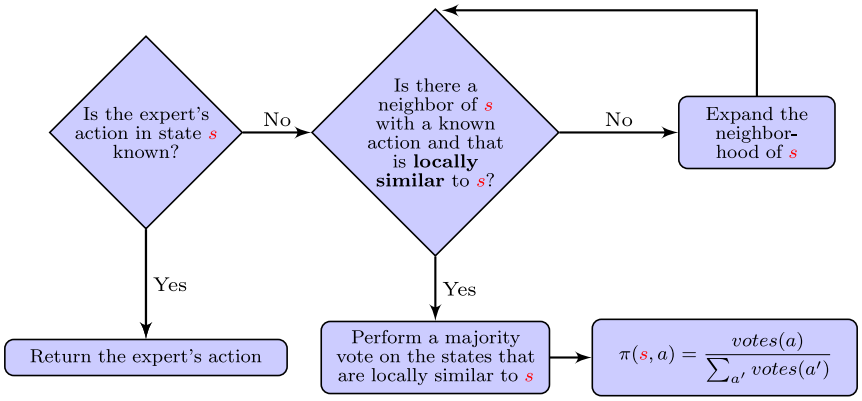

Fig. 3. Bootstrapping apprenticeship learning via soft local homomorphisms. The resulting policy is used as a substitute to the demonstration.

dimension $|\mathcal{S}|$ of the state space takes the values 256, 576, 1024, and 2304. The robot has four actions for moving in one of the four directions of the compass, but with a probability of 0.3 the action fails and result is a random move. The initial state corresponds to the position (0,0), and the discount factor γ is set to 0.99. The gridworld is divided into non-overlapping regions, and the reward function varies depending on the region in which the robot is located. For each region i, there is one feature f_i, where $f_i(s)$ indicates whether state s is in region i. The robot knows the features f_i, but not the weights w_i defining the reward function of the expert. The weights w_i are randomly generated. They are set to 0 with probability 0.9, and to a random value between 0 and 1 with probability 0.1.

The expert's policy π^E corresponds to the optimal deterministic policy found by value iteration. Although π^E was known for every state in this problem, we limited the number of demonstration trajectories to only 10, which is a small number compared to other methods (e.g., [13]). In fact, we are primarily interested in simulating apprenticeship learning scenarios wherein the examples are provided by an expert human. The provided examples are therefore costly and limited. The demonstration trajectories were sampled using the expert's policy, the length of each trajectory is 50 for the 16×16 and 24×24 grids, 100 for the 32×32 grid, and 200 for the 48×48 grid. The same trajectories were used for training with different approaches.

7.2. Evaluating the homomorphism bootstrapping approach on the gridworld problem

The robot is trained by using the LPAL algorithm. However, as mentioned before, this algorithm requires the knowledge of the feature counts ν_{i,π^E}, which is not the case in our experiments since the demonstration covers only a small number of states. Instead, we used the following methods for learning a generalized policy $\hat{\pi}^E$, and provided the feature values of $\hat{\pi}^E$ to LPAL. Except for Monte Carlo where the feature values are empirically calculated using Eq. (13), all the following methods used the flow equation (5) to calculate the feature values of $\hat{\pi}^E$.

- *Complete policy*: The complete expert's policy π^E is provided to LPAL. The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|)$.
- *Maximum entropy*: The generalized policy $\hat{\pi}^E$ is set to a uniform distribution in the states that did not appear in the demonstration. The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|)$.
- *Soft local homomorphism*: The generalized policy $\hat{\pi}^E$ is learned by Algorithm 1, the threshold ϵ is set to 0 and the distance d is set to 1. The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|^2 n^{3.5})$, where n is the maximum number of next states with a nonzero probability. This corresponds to a worst-case scenario where $|\mathcal{S}|^2$ linear programs are solved for measuring the similarity between every couple of states. The computational complexity of each linear program is $\mathcal{O}(n^{3.5})$, using Karmarkar's algorithm for instance [11]. In our case, there are four possible next states for every action, thus $n=4$.
- *Euclidean k-NN*: The generalized policy $\hat{\pi}^E$ is learned by the k-nearest neighbors algorithm using the Euclidean distance. The distance k is gradually increased until encountering at least one state that appears in the demonstration trajectories, as done in Algorithm 1. The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|^2)$.
- *Manhattan k-NN*: k-NN with a Manhattan distance, which counts the minimum number of transitions between two states. The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|^2)$.
- *Nonlinear regression*: The occupancy measure $\mu_{\hat{\pi}}$ is a linear function of a polynomial kernel defined on the horizontal and vertical coordinates of the robot's position. In other terms, for a state $s=(s_i,s_j)$, we have

$$\sum_{a\in\mathcal{A}} \mu_\pi(s,a) = \alpha_0+\alpha_1 s_i+\alpha_2 s_j+\alpha_3 s_i^2+\alpha_4 s_j^2+\alpha_5 s_i s_j+\epsilon(s).$$

  We use a linear program to minimize the sum of the errors $\sum_s|\epsilon(s)|$ under the Bellman flow constraints (Eq. (5)), while the states that appear in the demonstration are constrained to have the same action as the expert. Finally, $\hat{\pi}^E$ is extracted from μ_π according to Eq. (12). The computational complexity of this method is $\mathcal{O}(|\mathcal{S}|^{3.5})$.
- *Monte Carlo*: This is the method usually used in the literature, the counts $\hat{\nu}_{i,\pi^E}$ are estimated directly from the trajectories, according to Eq. (13). The computational complexity of this method is $\mathcal{O}(|\mathcal{S}^E|)$.

Table 1 shows the average reward per time-step of the robot's policy, averaged over 10^3 independent trials of the same length as the demonstration trajectories. Each trial starts with the same initial state (0,0), the actions are sampled according to the policy learned by one the methods described above. A different random seed is used for each trial.

Our first observation is that the LPAL algorithm learned policies just as good as the expert's policy when the feature values are calculated by using the expert's complete policy, but remarkably failed to do so when the feature values are learned from the demonstration by using a Monte Carlo estimator. This is due to the fact that we used a very small number of demonstrations compared to the size of these problems. Second, LPAL returns better policies when the values are analytically calculated

Table 1
Gridworld average rewards in the first experiment. The expert policy results are the optimal ones, the other results are found by combining different bootstrapping methods with the LPAL algorithm.

Gridworld size	Number of regions	Expert policy	Complete policy	Soft local homomorphism	Monte Carlo	Maximum entropy	Euclidian k-NN	Regression	Manhattan k-NN
16×16	16	0.4672	0.4692	**0.4663**	0.0380	0.3825	0.4672	0.4370	0.4635
	64	0.5281	0.5310	**0.5210**	0.0255	0.4607	0.5218	0.5038	0.5198
	256	0.3988	0.4029	**0.4053**	0.0555	0.3672	0.3915	0.3180	0.4062
24×24	64	0.6407	0.6386	**0.6394**	0.0149	0.5855	0.6394	0.5530	0.6334
	144	0.5916	0.5892	**0.5827**	0.0400	0.5206	0.5890	0.5069	0.5876
	576	0.3568	0.3553	**0.3489**	0.0439	0.2814	0.3114	0.2701	0.2814
32×32	64	0.6204	0.6179	**0.6188**	0.0145	0.5694	0.6198	0.5735	0.6177
	256	0.5773	0.5779	**0.5726**	0.0556	0.5118	0.5730	0.4372	0.5729
	1024	0.4756	0.4778	**0.4751**	0.0394	0.4482	0.4751	0.4090	0.4706
48×48	64	0.6751	0.6751	**0.6732**	0.0141	0.6234	0.6732	0.6052	0.6653
	256	0.6992	0.7006	**0.6909**	0.0603	0.6587	0.6999	0.6437	0.6997
	2304	0.4950	0.4972	**0.4876**	0.0528	0.4640	0.4913	0.4437	0.4330

Table 2
Gridworld average rewards in the second experiment.

Gridworld size	Number of regions	Expert policy	Euclidian k-NN	Monte Carlo MMP	Analytically bootstrapped MMP	Monte Carlo LPAL	Analytically bootstrapped LPAL
16×16	16	0.4672	0.4635	0.0000	**0.4678**	0.0380	**0.1572**
16×16	64	0.5281	0.5198	0.0000	**0.5252**	0.0255	**0.4351**
16×16	256	0.3988	0.4062	0.0537	**0.3828**	0.0555	**0.1706**
24×24	64	0.5210	0.6334	0.0000	**0.5217**	0.0149	**0.2767**
24×24	144	0.5916	0.5876	0.0122	**0.5252**	0.0400	**0.4432**
24×24	576	0.3102	0.2814	0.0974	**0.0514**	0.0439	**0.0349**

by using the maximum entropy principle than when they are estimated by Monte Carlo. This is because Monte Carlo estimates the values for a finite horizon. Given that the expert's actions cannot be explained by only the vertical and horizontal coordinates, the regression method also failed to outperform the maximum entropy method. We also remark that Euclidian and Manhattan k-NN performed similarly due to the similarity between these two distances in the context of flat grids. They both succeeded in learning policies with values close to the optimal value.

The performance of the bootstrapping approach with soft local homomorphisms is akin to k-NN in this experiment. In fact, the bootstrapping approach uses the same principle of performing a majority vote among neighboring states as in the k-NN method (Fig. 3). The main difference between the two methods is the metric used for measuring the similarity between states. The bootstrapping approach takes into account the dynamics of the system and uses the transfer error in Eq. (24) for deciding if two states are locally similar, whereas k-NN considers any neighboring states as similar. In this particular experiment, the dynamics of the system is nearly the same in most of the states since there are no obstacles on the grid. Consequently, local homomorphisms do not provide further information, except for the few states juxtaposing a wall.

Given that k-NN and local homomorphisms achieve the same performance in gridworlds, one should consider the computational time for deciding which method to use. The local homomorphism method has a higher computational complexity, $\mathcal{O}(|\mathcal{S}|^2 n^{3.5})$ compared to $\mathcal{O}(|\mathcal{S}|^2)$ for k-NN.

Table 3 shows the running times, in seconds, of the different algorithms. As expected, the bootstrapping advantage over Monte Carlo estimators comes with a high computational cost. This is caused by the linear programs solved at each state for generalizing the expert's policy. However, these algorithms are used typically offline, in simulation. Therefore, they are not limited by the real-time constraints of a physical system.

7.3. Evaluating the analytical bootstrapping on the gridworld problem

As in the previous experiment, we used only 10 demonstration trajectories. Table 2 shows the average reward per step of the agent's policy, averaged over 10^3 independent trials of the same length as the demonstration trajectories. Our first observation is that the analytically bootstrapped MMP learned policies just as good as the expert's policy, while both MMP and LPAL using the Monte Carlo (MC) estimator remarkably failed to collect any reward. This is due to the fact that we used a very small number of demonstrations (10×50 time-steps) compared to the size of these domains. Note that this problem is not specific to MMP or LPAL. In fact, any other algorithm using the same approximation method would produce similar results. The second observation is that the values of the policies learned by the analytically bootstrapped LPAL were between the values of LPAL with Monte Carlo and the optimal ones. In fact, the policy learned by the bootstrapped LPAL is the one that minimizes the difference between the expected value of a feature using this policy and the maximal one among all the policies that resemble to the expert's policy. Therefore, the learned policy maximizes the value of a feature that is not necessary a good one (with a high reward weight). We also notice that the performance of all the tested algorithms was low when 576 features were used. In this case, every feature takes a nonnull weight in one state only. Therefore, the demonstrations did not provide enough information about the rewards of the

Table 3
Gridworld learning times (in seconds) in the first experiment.

Gridworld size	Number of regions	Complete policy	Soft local homomorphism	Monte Carlo	Maximum entropy	Euclidian k-NN	Regression	Manhattan k-NN
16×16	16	0.26	33.65	0.3	0.44	0.15	0.17	0.16
	64	0.46	30.09	0.53	0.67	0.26	0.3	0.29
	256	0.73	32.61	0.74	0.85	0.62	0.66	0.62
24×24	64	2.11	107.27	1.79	2.61	0.64	0.87	0.63
	144	3.56	111.18	1.79	3.59	1.34	1.63	1.37
	576	3.36	108.71	3.48	4.26	2.88	3.47	2.91
32×32	64	1.27	214.28	1.66	3.19	1.65	2.18	1.73
	256	2.96	225.24	3.28	4.86	3.3	4.19	3.35
	1024	11.97	226.46	11.84	13.7	13.15	12.55	11.97
48×48	64	3.24	577.67	6.21	27.55	6.21	7.63	6.3
	256	9.45	590.41	13.06	58.89	12.37	13.7	12.39
	2304	89.3	671.9	93.36	183.47	92.08	93.62	91.82

Table 4
Gridworld learning times (in seconds) in the second experiment.

Gridworld size	Number of regions	Euclidian k-NN	Monte Carlo MMP	Analytically bootstrapped MMP	Monte Carlo LPAL	Analytically bootstrapped LPAL
16×16	16	0.15	15.7	44.89	0.31	13.35
16×16	64	0.26	18.75	40.48	0.54	35.8
16×16	256	0.62	21.74	41.51	0.75	127.37
24×24	16	0.22	38.04	102.47	2.55	45.72
24×24	64	0.64	46.71	100.72	1.81	168.16
24×24	256	0.22	51.52	98.05	3.13	455.05

states that were not visited by the expert. Finally, we remark that k-NN performed as an expert in this experiment. In fact, since there are no obstacles on the grid, neighboring states often tend have similar optimal actions.

Table 4 shows the running times, in seconds, of the different algorithms. Note that computational cost of the analytically bootstrapped MMP is about only twice that of the Monte Carlo MMP, since two optimal policies are found at each gradient-descent step (Eq. (17)). The learning times of the bootstrapped LPAL is, however, significantly higher than that of the Monte Carlo LPAL. Arguably, this high computational effort is justified by the tremendous performance improvement of bootstrapped LPAL over Monte Carlo LPAL.

7.4. Racetrack setting

We implemented a simplified care race simulator, the corresponding racetracks are showed in Fig. 4. The states correspond to the position of the care in the racetrack and its velocity. We considered two discretized velocities, low and high, in each direction of the vertical and horizontal axes, in addition to the zero velocity in each axis, leading to a total of 25 possible combinations of velocities, 5900 states for racetrack (1), and 5100 states for racetrack (2). The car controller can accelerate or decelerate in each axis, or do nothing. The controller cannot, however, combine a horizontal and a vertical action, the number of actions is then 5. When the velocity is low, acceleration|deceleration actions succeed with probability 0.9, and fail with probability 0.1, leaving the velocity unchanged.

The success probability of the actions falls down to 0.5 when the velocity is high, making the vehicle harder to control. When the vehicle tries to move off-road, it remains in the same position and its velocity is reset to zero. The car controller receives a reward of 5 for each time-step except for off-roads, where it receives 0, and for reaching the finish line, where the reward is 200. A discount factor of 0.99 is used in order to favor shorter trajectories.

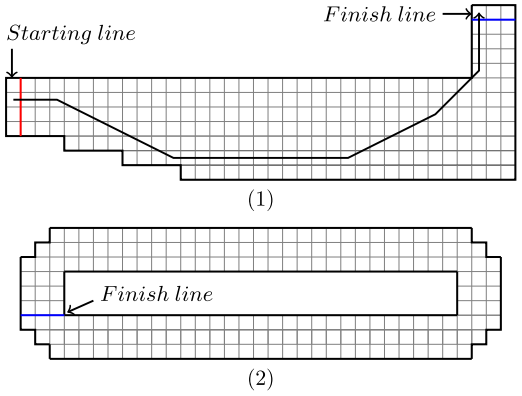

Fig. 4. Racetrack configurations and a demonstration of the expert's policy. In racetrack (2), the car starts at a random position.

7.5. Evaluating the homomorphism bootstrapping approach on the racetrack problem

In this experiment, we compared only the methods that performed well in the gridworld domain, which are LPAL using a complete policy, LPAL using soft local homomorphisms, and LPAL with k-NN using the Manhattan distance, since the Euclidean distance does not take into account the velocity of the vehicle. We also compared k-NN and soft local homomorphisms without LPAL.

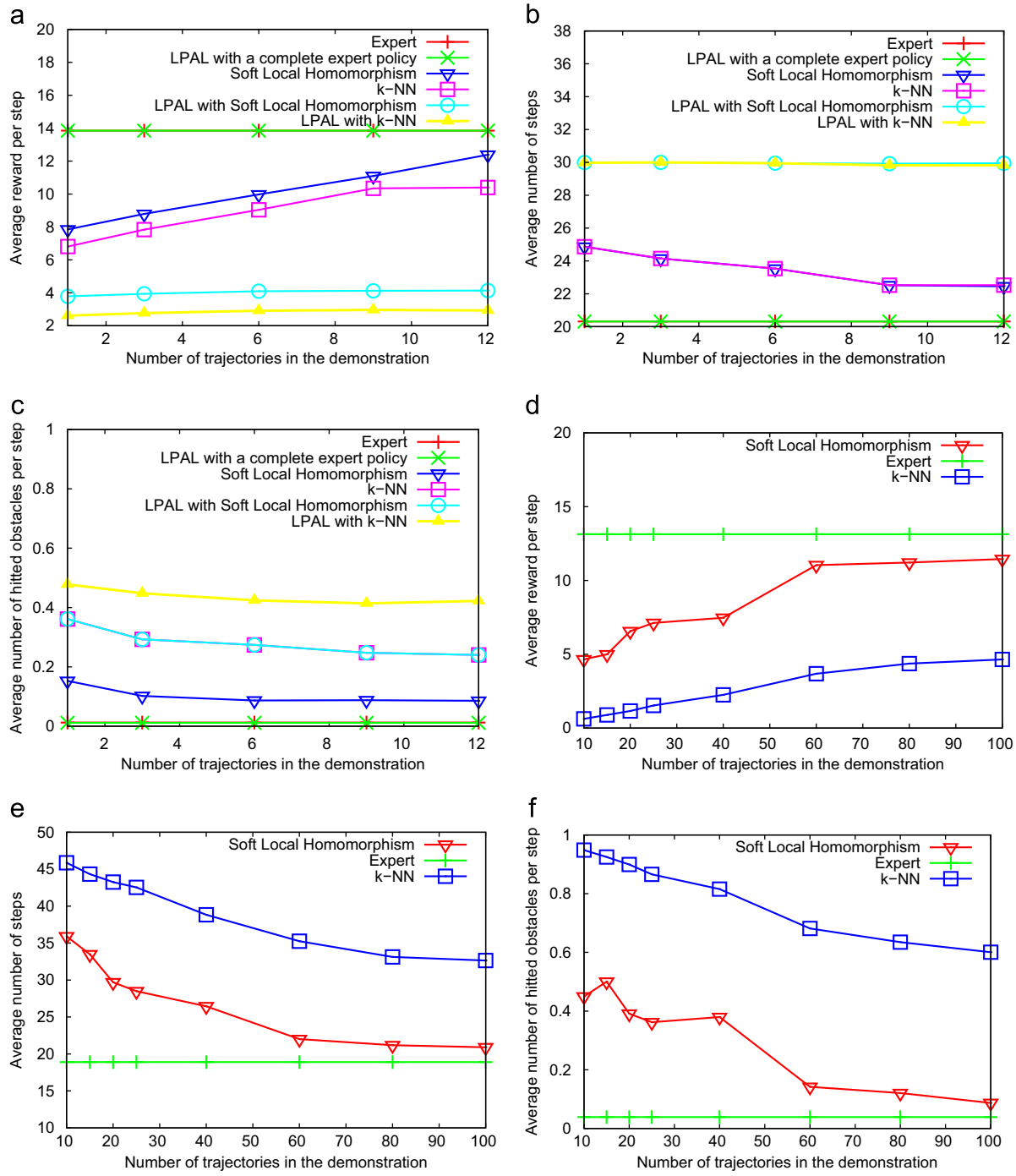

Fig. 5. Racetrack results of the homomorphism bootstrapping approach: (a) average reward in racetrack 1, (b) average number of steps in racetrack 1, (c) average number of off-roads, racetrack 1, (d) average reward in racetrack 2, (e) average number of steps in racetrack 2, (f) average number of off-roads, racetrack 2.

Fig. 5(a)–(f) shows the average reward per step of the robot's policy, the average number of off-roads per step, and the average number of time-steps before reaching the finish line, as a function of the number of trajectories in the demonstration. For racetrack (1), the car always starts from the same initial position, and the length of each demonstration trajectory is 20. For racetrack (2), however, the car starts at a random position, and the length of each trajectory is 40. The results are averaged over 1000 independent trials of length 30 for racetrack (1) and 50 for racetrack (2). Contrary to the gridworld experiments, LPAL achieved good performances only when the features are calculated by using the complete policy of the expert. For clarity, we removed the results of LPAL with k-NN and with soft local homomorphisms, which were below the performances of the other methods.

As expected, we notice the significant improvement of our algorithm over k-NN in terms of average reward, average number of off-roads per step, and average number of time-steps to the finish line. This is due to the fact that, contrary to k-NN, homomorphisms do take into account the dynamics of the system. For example, when the care faces an obstacle, the local MDP defined around its current position is similar to all the local MDPs defined around the positions of facing an obstacle, the deceleration actions can then be efficiently transferred, as depicted in the example of Fig. 6.

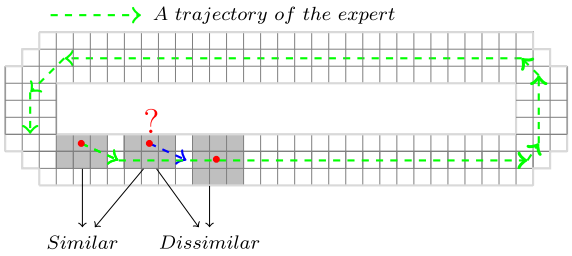

Fig. 6. An example of two states that are locally similar by homomorphism.

Table 5
Racetrack learning times (in seconds) in the first experiment.

Method	Problem	
	Racetrack 1	Racetrack 2
LPAL with complete policy	3.98	2.3
Local homomorphism	161.61	183.45
LPAL with homomorphism	163.2	188.05
k-NN	0.79	1.21
LPAL with *k*-NN	4.09	2.33

Table 5 shows the running times, in seconds, of the different algorithms. Here again, we note the high computational cost of the homomorphism techniques compared to simple classifiers, such as *k*-NN.

7.6. Evaluating the analytical bootstrapping techniques on the racetrack problem

Fig. 7(a)–(f) shows the average reward per step of the learned policies, the average proportion of off-road steps, and the average number of steps before reaching the finish line, as a function of the number of trajectories in the demonstration. We notice that the LPAL algorithm with both the Monte Carlo estimation of the expected feature values and the analytical bootstrapping failed to achieve good results in racetracks (1) and (2). This is due to the fact that LPAL tries to maximize the expected value of features that are not necessarily associated to a high reward, such as hitting obstacles. We also notice the nearly optimal performance of the bootstrapped MMP algorithm, in both racetracks (1) and (2). Table 6 shows the learning times in seconds. As in the gridworlds experiment, the computational cost of the bootstrapped MMP is only twice the cost of the Monte Carlo MMP.

8. Discussion

From the experimental analysis, we notice that the bootstrapping approach using local homomorphisms achieves better results in the gridworld domain, whereas the analytical bootstrapping achieves better results in the racetrack domain. In general, classification-based methods, a.k.a. direct imitation, perform better than IRL-based bootstrapping methods when a correct similarity measure is provided. In a large gridworld, most of the neighboring states are similar to each other and have the same optimal action. Therefore, *k*-NN methods are expected to outperform IRL-based methods in this type of problems. Our experimental analysis shows that there is no significant difference, in terms of performance, between the different metrics of *k*-NN (Euclidean, Manhattan, Homomorphism). However, the relatively high computational complexity of verifying the existence of a homomorphism between neighboring states leads to preferring simpler metrics over the local homomorphisms.

IRL-based bootstrapping methods achieve a higher performance in more complex goal-oriented problems. In the racetrack problem for example, the analytically bootstrapped method was able to learn an accurate reward function with only one demonstration trajectory. *k*-NN methods achieved lower performances because the Euclidean and the Manhattan distances are not good similarity measures in general MDPs. In fact, one should consider all the possible successor states in order to decide whether two states are similar or not. This problem can be alleviated by using the homomorphism measure, but due to its computational cost, one can only consider the immediate successor states. Nonetheless, the homomorphism measure leads to a better performance compared to the Euclidean or the Manhattan distances.

In both cases, the bootstrapping techniques proposed in this paper significantly outperformed, in terms of average rewards, the standard Monte Carlo estimator used in the apprenticeship learning literature.

However, the improved performance of the proposed bootstrapping techniques comes with a computational cost. This cost corresponds to the time that is needed for learning a generalized expert policy. Our empirical evaluations show that the running time of the analytically bootstrapped MMP is only twice that of the original MMP algorithm. The running times of the other bootstrapping methods are higher by orders of magnitude.

Nevertheless, once a generalized policy is found, the decision for taking an action takes less than a millisecond on average. In these experiments, learning and planning were all performed offline, in simulation, while the tests were performed online. Learning a reward function from examples may take as long as 10 min in some cases, but this is done by using only the dynamics equations without actually executing any actions or interacting with the physical system. Given the learned reward function, a generalized policy is found, again in simulation. The policy is a function that returns an action for every possible state. This policy is then saved in a table, when the state space is discretized, and used for testing on the physical system. During the tests, e.g. the race, an action is found by simply searching for the current state in the policy table and returning the corresponding action, which typically takes less than a millisecond.

Consequently, the best option would be to run the three bootstrapping algorithms in simulation using a dynamical model (transition function), and choose the one that achieves the highest performance. If the computational resources are not sufficient for that, then one can consider only the analytically bootstrapped MMP, which has shown to be a fast algorithm. In any case, the time constraints of a physical system, e.g. the velocity of a car, do not limit the applicability of these approaches, because the system can be slowed down in simulation. The learned policy can always be used in real-time on the real system.

9. Conclusion and future work

The main question addressed in imitation learning is how to generalize the expert's examples to states that have not been encountered during the demonstration. Apprenticeship Learning via Inverse Reinforcement Learning provides a simple and elegant answer which consists in first learning a reward function that explains the observed behavior, and then using it for finding an optimal policy that generalizes the examples.

Apprenticeship learning relies on the assumption that the reward function can be expressed in terms of state and action features. The provided examples are used to estimate the average value of each feature. The distance between these empirical values and the feature values of an optimal policy is minimized by a loss function, where the optimal policy is a function of the

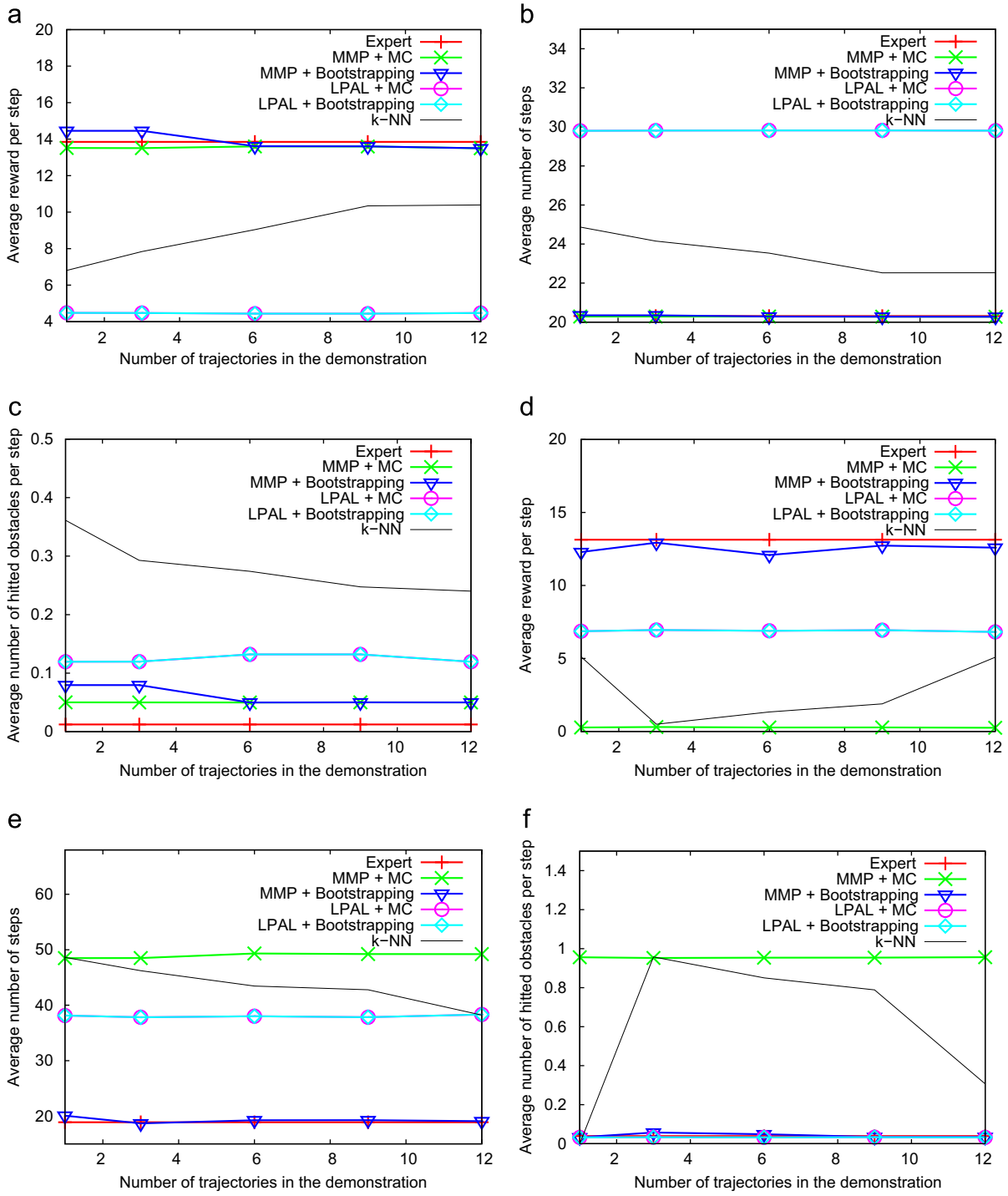

Fig. 7. Racetrack results with the analytical bootstrapping: (a) average reward in racetrack 1, (b) average number of steps in racetrack 1, (c) average number of off-roads, racetrack 1, (d) average reward in racetrack 2, (e) average number of steps in racetrack 2, (f) average number of off-roads, racetrack 2.

Table 6
Racetrack learning times (in seconds) in the second experiment.

Method	Problem	
	Racetrack 1	Racetrack 2
Monte Carlo MMP	356.51	306.27
Analytically bootstrapped MMP	694.73	698.4
Monte Carlo LPAL	4.04	2.47
Analytically bootstrapped LPAL	59.57	62.81

unknown reward. Consequently, the learned rewards are sensitive to the empirical error in estimating the average values of the features. This problem becomes more evident when the number of examples is small, as we showed in our experiments.

To overcome this problem, we introduced two bootstrapping techniques for apprenticeship learning. The first one consists in modifying the loss functions of well-known apprenticeship learning algorithms, namely MMP and LPAL, by replacing the empirical terms by the exact feature values of optimal policies that are consistent with the provided examples. We also provided theoretical insights

on the modified loss functions, showing that they admit the expert's true reward as a local optimum, under mild conditions. Although the modified loss functions are nonconvex, an empirical analysis in simulation confirmed the outperformance of bootstrapped apprenticeship learning algorithms, and MMP in particular. These promising results push us to investigate further theoretical properties and interpretations of the modified loss functions.

The second bootstrapping technique makes use of a direct imitation learning method, based on local graph homomorphism, for generalizing the provided examples. The generalized policy is used along with the known system dynamics in order to analytically calculate the average feature values. This technique is inspired from the fact that the states that are locally similar tend to have the same optimal action in general. Unlike other methods, the graph homomorphism approach considers the future possible states when comparing two states, rather than just the immediate features. We also showed that using homomorphisms leads to a significant improvement in the quality of the learned policies. However, this approach lacks a rigorous theoretical guarantee beyond the intuition. In fact, most real-world tasks admit optimal policies that are local and reactive in most states, such as avoiding obstacles during a navigation task. However, there are critical states where the optimal actions cannot be explained only by the local dynamics of the system. Distinguishing between these states is crucial for providing a theoretical guarantee of our approach in a future work.

In addition to solving the issues mentioned above, this work can be extended in many ways. First, the proposed techniques can be used as a general approach for bootstrapping other apprenticeship learning algorithms. Second, our approach requires that the transition model is known, which is a strong assumption shared with most apprenticeship learning algorithms. It would be interesting to consider the case when no transition model is given.

We should mention that our formulation of the generalization problem is limited in the sense that negative examples are not taken into account. This can be a problem when a state can have multiple optimal actions, and only one of them is selected by the expert. This problem can be addressed by explicitly providing sub-optimal actions and considering their votes in the algorithm.

Finally, this paper focuses on some algorithmic and theoretical aspects of bootstrapping apprenticeship learning, an empirical study on a real-world application would be an important extension of this work.

References

[1] P. Abbeel, A. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04), 2004, pp. 1–8.

[2] P. Abbeel, A. Coates, A.Y. Ng, Autonomous helicopter aerobatics through apprenticeship learning, Int. J. Robot. Res. 29 (13) (2010) 1608–1639.

[3] E.I. Barakova, T. Lourens, Mirror neuron framework yields representations for robot interaction, Neurocomputing 72 (4–6) (2009) 895–900.

[4] A. Billard, Imitation: a review, in: Michael A. Arbib (Ed.), The Handbook of Brain Theory and Neural Network, second ed. MIT Press, 2002, pp. 566–569.

[5] A. Boularias, B. Chaib-draa, Apprenticeship learning via soft local homomorphisms, in: 2010 IEEE International Conference on Robotics and Automation (ICRA'10), 2010, pp. 2971–2976.

[6] A. Boularias, B. Chaib-draa, Bootstrapping apprenticeship learning, in: Neural Information Processing Systems 24 (NIPS'10), 2010, pp. 1–9.

[7] A. Boularias, J. Kober, J. Peters, Relative entropy inverse reinforcement learning, J. Mach. Learn. Res. Proceed. Track 15 (2011) 182–189.

[8] A. Boularias, O. Kroemer, J. Peters, Learning robot grasping from 3-D images with Markov random fields, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11), 2011.

[9] E. Hourdakis, H.E. Savaki, P. Trahanias, Computational modeling of cortical pathways involved in action execution and action observation, Neurocomputing 74 (7) (2011) 1135–1155.

[10] N. Jetchev, M. Toussaint, Task space retrieval using inverse feedback control, in: Proceedings of the Twenty-eighth International Conference on Machine Learning (ICML'11), 2011, pp. 449–456.

[11] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, ACM, STOC'84, 1984, pp. 302–311.

[12] Z. Kolter, P. Abbeel, A. Ng, Hierarchical apprenticeship learning with application to quadruped locomotion, in: Neural Information Processing Systems (NIPS'08), 2008, pp. 769–776.

[13] G. Neu, C. Szepesvri, Apprenticeship learning using inverse reinforcement learning and gradient methods, in: Conference on Uncertainty in Artificial Intelligence (UAI'07), 2007, pp 295–302.

[14] A. Ng, S. Russell, Algorithms for inverse reinforcement learning, in: Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00), 2000, pp. 663–670.

[15] J. Peters, S. Schaal, Natural actor critic, Neurocomputing 71 (7–9) (2008) 1180–1190.

[16] D. Ramachandran, E. Amir, Bayesian inverse reinforcement learning, in: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07), 2007, pp. 2586–2591.

[17] N. Ratliff, J. Bagnell, M. Zinkevich, Maximum margin planning, in: Proceedings of the Twenty-third International Conference on Machine Learning (ICML'06), 2006, pp. 729–736.

[18] N. Ratliff, D. Silver, A. Bagnell, Learning to search: functional gradient techniques for imitation learning, Autonomous Robots 27 (1) (2009) 25–53.

[19] B. Ravindran, An Algebraic Approach to Abstraction in Reinforcement Learning, Ph.D. Thesis, University of Massachusetts, Amherst, MA, 2004.

[20] S. Schaal, Is imitation learning the route to humanoid robots? Trends Cognitive Sci. 3 (6) (1999) 233–242.

[21] J. Sorg, S. Singh, Transfer via soft homomorphisms, in: Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), 2009, pp. 741–748.

[22] U. Syed, R. Schapire, A game-theoretic approach to apprenticeship learning, in: Advances in Neural Information Processing Systems 20 (NIPS'08), 2008, pp. 1449–1456.

[23] U. Syed, M. Bowling, R.E. Schapire, Apprenticeship learning using linear programming, in: Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), 2008, pp. 1032–1039.

[24] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: a survey, J. Mach. Learn. Res. 10 (1) (2009) 1633–1685.

[25] B. Ziebart, Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy, Ph.D. Thesis, Carnegie Mellon University, PA, 2010.

Abdeslam Boularias received an Engineering degree in Computer Science from École nationale Supérieure d'Informatique (ESI), Algeria, in October 2004, and a Master degree in the same field from Paris-Sud University (Paris XI) in September 2005. During his studies at Paris XI, he was a research assistant at the INRIA Saclay Institute, where he worked on fault-tolerance in Grid Computing. In January 2006, Abdeslam joined the group of Dr. Brahim Chaib-draa at Laval University, Canada, where he graduated with a Ph.D. degree in July 2010. Since August 2010, he is a research scientist at the Max Planck Institute for Intelligent Systems in Tübingen, in the Machine Learning Department, led by Prof. Bernhard Schölkopf. His main research interests include planning under uncertainty, reinforcement learning, imitation learning, and multi-agent systems.

Brahim Chaib-draa received a Diploma in Computer Engineering from École Supérieure d'Électricité (SUPELEC), Paris, France, in 1978 and a Ph.D. degree in Computer Science from the Université du Hainaut-Cambrésis, Valenciennes, France, in 1990. In 1990, he joined the Department of Computer Science and Software Engineering at Laval University, Quebec, QC, Canada, where he is a Professor and Group Leader of the Decision for Agents and Multi-Agent Systems (DAMAS) Group. His research interests include agent and multiagent technologies, natural language for interaction, formal systems for agents and multiagent systems, distributed practical reasoning, and real-time and distributed systems. He is the author of several technical publications in these areas. He is on the Editorial Boards of IEEE Transactions on SMC, Computational Intelligence and The International Journal of Grids and Multiagent Systems. Dr. Chaib-draa is a member of ACM and AAAI and senior member of the IEEE Computer Society.