Apprenticeship Learning via Soft Local Homomorphisms

Abdeslam Boularias and Brahim Chaib-draa
Computer Science and Software Engineering Department, Laval University, Quebec, G1V 0A6 Canada.
{boularias,chaib}@damas.ift.ulaval.ca

Abstract— **We consider the problem of apprenticeship learning when the expert's demonstration covers only a small part of a large state space. Inverse Reinforcement Learning (IRL) provides an efficient solution to this problem based on the assumption that the expert is optimally acting in a Markov Decision Process (MDP). However, past work on IRL requires an accurate estimate of the frequency of encountering each feature of the states when the robot follows the expert's policy. Given that the complete policy of the expert is unknown, the features frequencies can only be empirically estimated from the demonstrated trajectories. In this paper, we propose to use a transfer method, known as soft homomorphism, in order to generalize the expert's policy to unvisited regions of the state space. The generalized policy can be used either as the robot's final policy, or to calculate the features frequencies within an IRL algorithm. Empirical results show that our approach is able to learn good policies from a small number of demonstrations.**

I. Introduction

Modern robots are designed to perform complicated planning and control tasks, such as manipulating objects, navigating in outdoor environments, and driving in urban settings. Unfortunately, manually programming these tasks is almost infeasible in practice due to the high number of related states. Markov Decision Processes (MDPs) provide efficient mathematical tools to handle such tasks with a little help from an expert. The expert's help consists in simply specifying a reward function. However, in many practical problems, even specifying a reward function is not easy. In fact, it is often easier to demonstrate examples of a desired behavior than to define a reward function (Ng & Russell, 2000).

Learning policies from demonstrated examples, a.k.a. apprenticeship learning, is a technique that has been widely used in robotics. One can generally distinguish between direct and undirect apprenticeship approaches (Ratliff et al., 2009). In direct methods, the robot learns a function that maps state features into actions by using a supervised learning technique (Atkeson & Schaal, 1997). The best known example of a system built on this paradigm is ALVINN (Pomerleau, 1989), where a neural network was trained to learn a mapping between a road image and a vehicle steering action. Despite the remarkable success of the ALVINN system and others, direct methods suffer from a serious drawback: they can learn only reactive policies, where the optimal action of a state depends only on its features, regardless of the future states of the system.

To overcome this drawback, Ng and Russell (2000) introduced a new approach of undirect apprenticeship learning known as Inverse Reinforcement Learning (IRL). The aim of IRL is to recover a reward function under which the expert's policy is optimal, rather than to directly mimic the actions of the expert. The learned reward function is then used to find an optimal policy. Contrary to direct methods, IRL takes into account the fact that the different states of the system are related by transition and value functions. Consequently, the expert's actions can be predicted in states that are different from the states appearing in the demonstration.

Unfortunately, as already pointed by (Abbeel & Ng, 2004), recovering a reward function is an ill-posed problem. In fact, the expert's policy can be optimal under an infinite number of reward functions. Abbeel and Ng (2004) proposed to rather minimize the worst-case loss in value of the learned policy compared to the expert's one. Their algorithm relies on the assumption that the reward function is a linear combination of state features, and the frequency of encountering each feature can be accurately estimated from the demonstration. This assumption is considered in most of apprenticeship learning methods, despite the fact that the features frequencies might be poorly estimated when the number of demonstrations is small, as we will show in our experiments.

In this paper, we propose to use a transfer learning technique, known as soft homomorphism (Sorg & Singh, 2009), 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 frequencies of the features. Contrary to previous direct methods, homomorphisms take into account the long term dependency between different states. We will show that combining this transfer method with other apprenticeship algorithms provides a significant improvement in the quality of the learned policies.

II. Preliminaries

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 function ($T(s,a,s') = Pr(s_{t+1} = s'|s_t = s, a_t = a), s, s' \in \mathcal{S}, a \in \mathcal{A}$), R is a reward function ($R(s,a)$ is the reward associated to executing action a in state s), α is the initial state distribution, and γ is a discount factor used to weigh less rewards received further in the future. We denote by MDP\R an MDP without a reward function. We assume that there exists a vector of k features $\phi_i : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, and the reward is a linear function of these features with positive weights w_i:

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

The robot decides which action to execute according to its policy π, defined as $\pi(s,a) = Pr(a_t = a|s_t = s)$. The value $V(\pi)$ of a policy π is the expected sum of rewards that the robot will receive if its actions are sampled according to π.

$$V(\pi) = E[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)|\alpha,\pi,T]$$

An optimal policy π^* is one that satisfies $\pi^* = \arg\max_\pi V(\pi)$. The occupancy measure x^π of a policy π is defined as:

$$x^\pi(s,a) = E[\sum_{t=0}^{\infty}\gamma^t \delta_{s_t,s}\delta_{a_t,a}|\alpha,\pi,T]$$

where δ is the Kronecker delta. We also define V_i, the expected frequency of a feature ϕ_i, as follows:

$$V_i(\pi) = E[\sum_{t=0}^{\infty}\gamma^t\phi_i(s_t,a_t)|\alpha,\pi,T] = \sum_{s\in\mathcal{S},a\in\mathcal{A}} x^\pi(s,a)\phi_i(s,a)$$

Using this definition, the value function of a policy is given by $V(\pi) = \sum_{i=0}^k w_i V_i(\pi)$. Therefore, the value is completely determined by the expected frequencies of the features ϕ_i.

III. Apprenticeship Learning

The aim of apprenticeship learning is to find a policy π that is at least as good as the expert's policy π^E, i.e. $V(\pi) \geq V(\pi^E)$. The value functions of π and π^E cannot be compared directly, given that the true reward function is unknown. As a first solution to this problem, Ng and Russell (2000) proposed to first learn a reward function, assuming that the expert's policy is optimal, and then use it to find a policy π. However, the assumption that the expert's policy is optimal cannot be guaranteed in practice. Abbeel and Ng (2004) did not consider this assumption, their algorithm returns a policy π with a bounded loss in the value function, i.e. $\| V(\pi) - V(\pi^E) \| \leq \varepsilon$. However, this algorithm iteratively calls an MDP planner as a subroutine, which considerably affects its computational efficiency. In this work, we will adopt a faster algorithm proposed by Syed et al. (2008), known as LPAL (Linear Programming Apprenticeship Learning).

LPAL algorithm is based on the following observation: if for some policy π we have $v^* = \min_{i=0,\dots,k-1} V_i(\pi) - V_i(\pi^E)$ then $V(\pi) \geq V(\pi^E) + v^*$. This is a direct consequence of the assumption that the weights w_i are positive. LPAL consists in maximizing the margin v^*, aiming to find policies that might outperform the expert's one. The maximal value of v^* is found by solving the following linear program:

$$\max_{v,x^\pi} \quad v$$

such that

$$\forall i \in \{0,\dots,k-1\}:$$
$$v \leq \sum_{s\in\mathcal{S},a\in\mathcal{A}} x^\pi(s,a)\phi_i(s,a) - V_i(\pi^E) \qquad (2)$$
$$\forall s\in\mathcal{S}, a\in\mathcal{A}:$$
$$\sum_{a\in\mathcal{A}} x^\pi(s,a) = \alpha(s) + \gamma\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}} x^\pi(s',a)T(s',a,s) \qquad (3)$$
$$x^\pi(s,a) \geq 0 \qquad (4)$$

The Bellman flow constraints (3) and (4) define the feasible set of x^π. The learned policy π is given by:

$$\pi(s,a) = \frac{x^\pi(s,a)}{\sum_{a'\in\mathcal{A}} x^\pi(s,a')} \qquad (5)$$

As many other algorithms, LPAL requires the knowledge of the features frequencies $V_i(\pi^E)$ (in equation (2)). These frequencies can be analytically calculated only when a complete expert policy is provided. However, the expert provides only a sequence of M demonstration trajectories $t_m = (s_1^m, a_1^m, \dots, s_H^m, a_H^m,)$. The estimated frequencies $\hat{V}_i(\pi^E)$, which are used in lieu of $V_i(\pi^E)$ in LPAL, are given by:

$$\hat{V}_i(\pi^E) = \frac{1}{M}\sum_{m=1}^{M}\sum_{t=1}^{H}\gamma^t\phi_i(s_t^m,a_t^m) \qquad (6)$$

There are nevertheless many problems related to this approach. First, the estimated frequencies $\hat{V}_i(\pi^E)$ can be too different from the true ones when the demonstration trajectories are few. Second, the frequencies $\hat{V}_i(\pi^E)$ are estimated for a finite horizon H, whereas the frequencies $V_i(\pi)$, given by $\sum_{s\in\mathcal{S},a\in\mathcal{A}} x^\pi(s,a)\phi_i(s,a)$ and used in the objective function (equation (2)), are calculated for an infinite horizon (equations (3) and (4)). In practice, these two values are very different and cannot be compared as done in equation (2). Finally, a frequency $V_i(\pi^E)$ cannot even be estimated if the feature ϕ_i takes non null values only in states that did not appear in the demonstration.

To solve these problems, we propose a new approach based on transferring the demonstrated actions to the complete state space. The generalized policy $\hat{\pi}^E$ will be used for calculating the frequencies $V_i(\hat{\pi}^E)$ by solving Bellman flow equations (3) and (4), and $V_i(\hat{\pi}^E)$ will be used as our estimator of $V_i(\pi^E)$, i.e. $\hat{V}_i(\pi^E) = V_i(\hat{\pi}^E)$.

IV. Transfer Learning

Transfer learning refers to the problem of using the policy learned for performing some task in order to perform a related, but different, task. The related task may be defined on a new domain, or on the same domain but in a different region of the state space. This problem has been widely studied in the context of reinforcement learning, and due to the lack of space, we cannot give an overview of the literature. The interested reader might find an extended overview in (Taylor & Stone, 2009).

In this paper, we focus on a transfer method known as MDP homomorphism (Ravindran, 2004), and more particularly, on soft MDP homomorphism (Sorg & Singh, 2009). The core idea of this latter approach consists in finding a function f, called the transfer function, that maps each state of an MDP model $\mathcal{M} = (\mathcal{S},\mathcal{A},T,R,\alpha,\gamma)$ to a probability distribution over 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:

$$f: \mathcal{S}\times\mathcal{S}' \mapsto [0,1]$$
$$\forall s\in\mathcal{S}, s'\in\mathcal{S}', \forall a\in\mathcal{A}:$$
$$\sum_{s''\in\mathcal{S}} T(s,a,s'')f(s'',s') = \sum_{s''\in\mathcal{S}'} f(s,s'')T'(s'',a,s') \qquad (7)$$

The reward function also should be preserved, but we will not consider this constraint since, in the context of apprenticeship learning, the reward function is unknown.

Sorg and Singh (2009) showed that soft homomorphisms can be used to transfer the values of the policies from an MDP to another. In the next section, we will show how one can use soft homomorphisms in order to transfer actions from a subset of a state space to another subset of the same space.

V. Apprenticeship Learning 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 actions if it is stochastic. We will 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\R $\mathcal{M}^E = (\mathcal{S}^E, \mathcal{A}, T^E, \alpha, \gamma)$, where the transition function T^E is defined as:

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

This function ensures that all the transitions remain within the states of $\mathcal{S}^E$ by assuming that any transition 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 loss function corresponds to the error in preserving the transition probabilities according to equation (7). The transfer function f of this homomorphism is found by solving the following linear program:

$$\min_f \quad e \tag{8}$$

such that

$$\forall s \in \mathcal{S}, s' \in \mathcal{S}^E, \forall a \in \mathcal{A}:$$
$$|\sum_{s'' \in \mathcal{S}} T(s,a,s'') f(s'',s') - \sum_{s'' \in \mathcal{S}^E} f(s,s'') T^E(s'',a,s')| \leq e$$
$$f(s,s') \geq 0, \sum_{s' \in \mathcal{S}^E} f(s,s') = 1$$

The transfer function f corresponds to a measure of similarity between two states. One can use this measure in order to define the generalized policy $\hat{\pi}^E$ as follows:

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

Unfortunately, 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 this 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 steps, and by $\mathcal{M}^{s,d}(s^d, \mathcal{A}, T^{s,d}, \alpha, \gamma)$ the MDP\R model defined on these states. The transition function $T^{s,d}$ is then defined as:

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

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:

$$\min_f \quad e \tag{9}$$

such that

$$\forall s_i \in s^d, s_k \in s'^d, \forall a \in \mathcal{A}:$$
$$|\sum_{s_j \in s^d} T^{s,d}(s_i,a,s_j) f(s_j,s_k) - \sum_{s_j \in s'^d} f(s_i,s_j) T^{s',d}(s_j,a,s_k)| \leq e$$
$$f(s_i,s_k) \geq 0, \sum_{s_k \in s'^d} f(s_i,s_k) = 1$$

The principal steps of our approach are summarized in Algorithm 1. 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. 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. If $s^t = s^{t-1}$, i.e. all the

```
Input: An MDP model without reward $(\mathcal{S}, \mathcal{A}, T, \alpha, \gamma)$, a set of
       demonstration trajectories, an error threshold $\varepsilon$, and a
       similarity distance $d$;
Let $\mathcal{S}^E$ be the set of states contained in the demonstration
trajectories;
Use the demonstration trajectories to estimate the policy $\hat{\pi}^E$
for the states of $\mathcal{S}^E$;
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;
foreach $s \in \mathcal{S}\backslash\mathcal{S}^E$ do
    $t \leftarrow 0, s^0 \leftarrow \{s\}, votes \leftarrow (0,\dots,0), c \leftarrow$ false;
    repeat
        $t \leftarrow t+1$;
        if $s^t = s^{t-1}$ then
            $c \leftarrow$ true, $votes \leftarrow (1,\dots,1)$;
        else
            foreach $s' \in s^t \cap \mathcal{S}^E$ do
                Let $e$ be the error returned by the linear
                program (9) on $(\mathcal{M}^{s,d}, \mathcal{M}^{s',d})$ ;
                if $e \leq \varepsilon$ then
                    $c \leftarrow$ true;
                    foreach $a \in \mathcal{A}$ do
                        $votes(a) \leftarrow votes(a) + \hat{\pi}^E(s',a)$
    until $c = true$ ;
    foreach $a \in \mathcal{A}$ do
        $\hat{\pi}^E(s,a) = \frac{votes(a)}{\sum_{a' \in \mathcal{A}} votes(a')}$;
Output: A generalized policy $\hat{\pi}^E$;
```

Algorithm 1: Apprenticeship Learning via Soft Local Homomorphisms

Gridworld Size	Number of Regions	**Expert policy**	**Full 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 I
GRIDWORLD RESULTS

states that can be reached from s are already contained in s^{t-1}, and no one is locally similar to s, then we set $\hat{\pi}^E(s,a)$ to a uniform distribution. 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 s.

The generalized policy $\hat{\pi}^E$ can be either considered as the robot's policy, or used to calculate the features frequencies $\hat{V}_i(\pi^E)$ for another algorithm, as LPAL.

VI. Experiments

To validate our approach, we experimented on two simulated navigation domains. The first one is a gridword problem taken from (Abbeel & Ng, 2004). 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.

A. Gridworld

We consider multiple x by x gridworld domains, with x taking the following values: 16, 24, 32, and 48. The state of the robot corresponds to its location on the grid, therefore, the 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 actions fail 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 ϕ_i, where $\phi_i(s)$ indicates whether state s is in region i. The robot knows the features ϕ_i, but not the weights w_i defining the reward function of the expert (equation (1)). The weights w_i 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. In all our experiments on gridworlds, we used only 10 demonstration trajectories, which is a significantly small number compared to other methods ((Neu & Szepesvári, 2007) for example). The length of the trajectories are 50 for the 16 by 16 and 24 by 24 grids, 100 for the 32 by 32 grid, and 200 for the 48 by 48 grid.

The robot is trained by using LPAL algorithm. However, as already mentioned, this algorithm requires the knowledge of the frequencies $V_i(\pi^E)$, 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 features frequencies of $\hat{\pi}^E$ to LPAL. Except for Monte Carlo, the frequencies $V_i(\hat{\pi}^E)$ are calculated by solving the flow equations (3) and (4).

Full policy: the complete expert's policy π^E is provided to LPAL.

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.

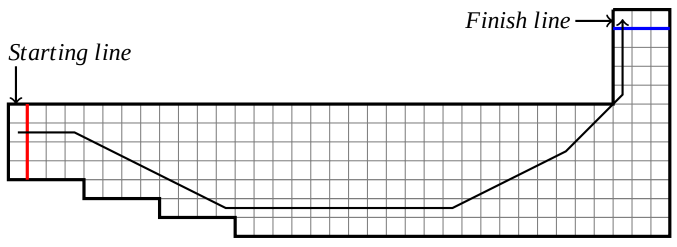
Maximum entropy: The generalized policy $\hat{\pi}^E$ is set to a uniform distribution on the states that did not appear in the demonstration.

Euclidian *k*-NN: The generalized policy $\hat{\pi}^E$ is learned by the k-nearest neighbors algorithm using the Euclidian distance. The distance k is gradually increased until encountering at least one state that appears in the demonstration trajectories.

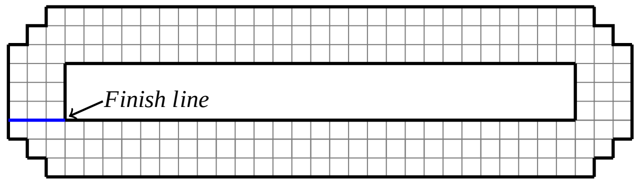
Manhattan *k*-NN: the Manhattan distance from state s_i to state s_j is the number of states contained in the shortest path from state s_i to state s_j on the MDP graph.

Nonlinear regression: The occupancy measure $x^{\hat{\pi}^E}$ is considered as a linear function of a polynomial kernel defined on the horizontal and vertical coordinates of the robot's position. In other terms, for each state $s = (s_i, s_j)$ we have $\sum_{a \in \mathcal{A}} x^{\hat{\pi}^E}(s,a) = \alpha_0 + \alpha_1 s_i + \alpha_2 s_j + \alpha_3 s_i^2 + \alpha_4 s_j^2 + \alpha_5 s_i s_j + \varepsilon(s)$. We use a linear program to minimize $\sum_s |\varepsilon(s)|$ under Bellman flow constraints, the states that appear in the demonstration are constrained to have the same action as the expert. Finally, $\hat{\pi}^E$ is extracted from $x^{\hat{\pi}^E}$ according to equation (5).

Monte Carlo: This is the method used in the literature, the frequencies $\hat{V}_i(\pi^E)$ are estimated directly from the trajectories, according to equation (6).



(a)



(b)

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

Table I shows the average reward per step of the robot's policy, averaged over 1000 independent trials of the same length as the demonstration trajectories. Our first observation is that LPAL algorithm learned policies just as good as the expert's policy when the features frequencies are calculated by using the expert's full policy, but remarkably failed to do so when the frequencies 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 frequencies are analytically calculated by using maximum entropy technique than when they are estimated by Monte Carlo. This is because Monte Carlo estimates the frequencies 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 to learn policies with values close to the optimal value. Finally, our approach performed just as *k*-NN in this experiment. In fact, since there are no obstacles on the grid, most of the states are locally similar, even for neighbors of a long distance.

B. Racetrack

We implemented a simplified car race simulator, the corresponding racetracks are showed in Fig. 2. The states correspond to the position of the car in the racetrack and its speed. We considered two discretized speeds, low and high, in each direction of the vertical and horizontal axis, in addition to the zero speed in each axis, leading to a total of 25 possible combinations of speeds, 5900 states for racetrack (a), and 5100 states for racetrack (b). The 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 then is five. When the speed is low, acceleration\deceleration actions succed with probability 0.9, and fail with probability 0.1, leaving the speed unchanged. The success probability falls down to 0.5 when the speed is high, making the vehicle harder to control. When the vehicle tries to move off-road, it remains in the same position and its speed is reset to zero. The controller receives a reward of 5 for each 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 favour shorter trajectories.

In this experiment, we compared only the methods that performed well in the gridworld domain, which are LPAL with a full policy, LPAL with soft local homomorphisms, and LPAL with *k*-NN using the Manhattan distance, since the Euclidian distance considers only the position of the vehicle and not its speed. We also compared *k*-NN and soft local homomorphisms without LPAL.

Figures 2 (a-f) show the average reward per step of the controller's policy, the average number of off-roads per step, and the average number of steps before reaching the finish line, as a function of the number of trajectories in the demonstration. For racetrack (a), the car always starts from the same initial position, and the length of each demonstration trajectory is 20. For racetrack (b) 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 (a) and 50 for racetrack (b).

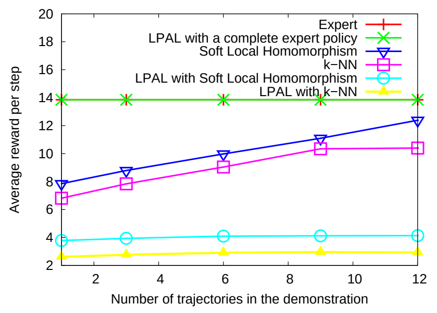
Contrary to the gridworld experiments, LPAL achieved good performances only when the features are calculated by using the complete policy of the expert. For clarity, we removed from Figures 2 (d-f) the results of LPAL with *k*-NN and with soft local homomorphisms, which were below the performances of 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 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 car 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 optimal action in these states, which is to decelerate, can then be efficiently transferred.

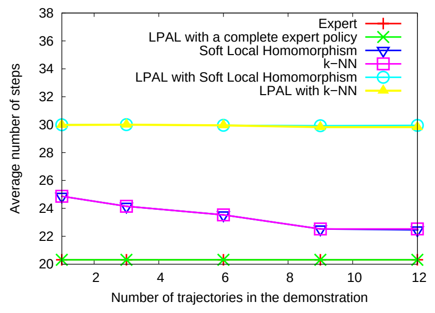
VII. Conclusion

The main question of apprenticeship learning is how to generalize the expert's policy to states that have not been encountered during the demonstration. Previous works have attempted to solve this problem by considering the state as a vector of features of a smaller dimension, and classifying the states accordingly. However, an expert's policy is a much more complicated function than it can be explained by only immediate features.

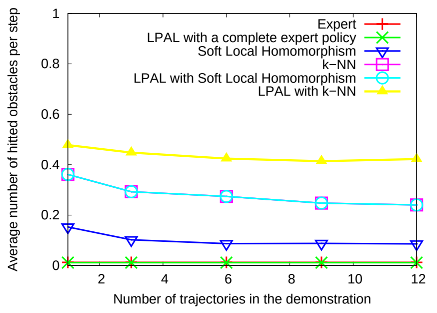
Inspired by the intuition that the states that are locally similar have the same optimal action in general, we introduced a new technique for generalizing the expert's policy



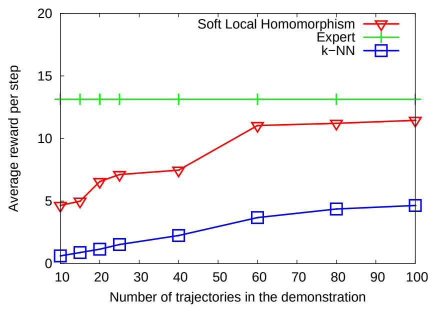
(a) Average reward per step in racetrack (a).



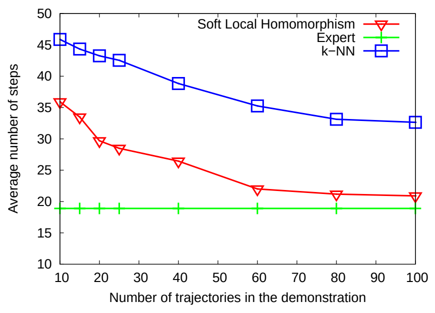
(b) Average number of steps before reaching the finish line in racetrack (a).



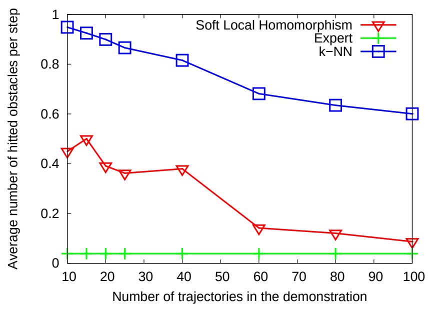
(c) Average number of off-roads per step in racetrack (a).



(d) Average reward per step in racetrack (b).



(e) Average number of steps before reaching the finish line in racetrack (b).



(f) Average number of off-roads per step in racetrack (b).

Fig. 1. Racetrack results

based on soft local homomorphisms. Unlike other methods, our approach considers the long term dependency between different states, rather than just immediate features. We also showed that using homomorphisms leads to a significant improvement in the quality of the learned policies.

However, our approach lacks of a theoretical guarantee beyond the intuition. In fact, control policies are local and reactive in most states, such as avoiding obstacles during a navigation task, but there are always some critic states where the optimal actions cannot be explained by only the local dynamics of the system. Distinguishing between these states is crucial for providing a theoretical guarantee of our approach in a future work. Another interesting research avenue is to consider using graph kernels (S. V. N. Vishwanathan & Schraudolph, 2007) for imitation learning. In fact, local homomorphisms can be seen as a special graph kernel measuring the similarity between two nodes, and other types of graph kernels can be used for the same purpose.

References

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship Learning via Inverse Reinforcement Learning. *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)* (pp. 1–8).

Atkeson, C., & Schaal, S. (1997). Robot Learning From Demonstration. *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*.

Neu, G., & Szepesvári, C. (2007). Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods. *Conference on Uncertainty in Artificial Intelligence (UAI'07)* (pp. 295–302).

Ng, A., & Russell, S. (2000). Algorithms for Inverse Reinforcement Learning. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)* (pp. 663–670).

Pomerleau, D. (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. *Neural Information Processing Systems (NIPS'89)* (pp. 769–776).

Ratliff, N., Silver, D., & Bagnell, A. (2009). Learning to Search: Functional Gradient Techniques for Imitation Learning. *Autonomous Robots*, *27*, 25–53.

Ravindran, B. (2004). *An Algebraic Approach to Abstraction in Reinforcement Learning*. Doctoral dissertation, University of Massachusetts, Amherst MA.

S. V. N. Vishwanathan, K. B., & Schraudolph, N. N. (2007). Fast Computation of Graph Kernels. *Neural Information Processing Systems (NIPS'07)* (pp. 1449–1456).

Sorg, J., & Singh, S. (2009). Transfer via Soft Homomorphisms. *Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)* (pp. 741–748).

Syed, U., Bowling, M., & Schapire, R. E. (2008). Apprenticeship Learning using Linear Programming. *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)* (pp. 1032–1039).

Taylor, M. E., & Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, *10*, 1633–1685.