Multi-Agent Inverse Reinforcement Learning

Sriraam Natarajan*, Gautam Kunapuli*, Kshitij Judah†, Prasad Tadepalli†,
Kristian Kersting‡ and Jude Shavlik*
*University of Wisconsin-Madison, †Oregon State University and ‡Fraunhofer IAIS

Abstract

Learning the reward function of an agent by observing its behavior is termed inverse reinforcement learning and has applications in learning from demonstration or apprenticeship learning. We introduce the problem of multi-agent inverse reinforcement learning, where reward functions of multiple agents are learned by observing their uncoordinated behavior. A centralized controller then learns to coordinate their behavior by optimizing a weighted sum of reward functions of all the agents. We evaluate our approach on a traffic-routing domain, in which a controller coordinates actions of multiple traffic signals to regulate traffic density. We show that the learner is not only able to match but even significantly outperform the expert.

I. Introduction

Traditional Reinforcement Learning (RL) [1] techniques aim to optimize some notion of long-term reward. The goal of RL is to find a policy that maps from states of the world to the actions executed by the agent. The key assumption for RL is that the reward function being optimized is accessible to the agent. However, there are several cases in which the reward might not be easily specifiable [2]. This naturally occurs in the case where an agent observes an expert and tries to learn from the expert; this is called *apprenticeship learning,* [3]. Consider, for example, expert human operators who monitor different roads and control signals to regulate the traffic. While humans may have a "reward function" or optimization criterion in mind, it may not be explicit. It would be desirable to have an automated system that can observe the human agents, learn their reward functions, and optimize them automatically.

Inverse reinforcement learning (IRL) [2], [3] aims to learn precisely in such situations. The goal of IRL is to observe an agent acting in the environment and determine the reward function that the agent is optimizing. The observations include the agent's behavior over time, the measurements of the sensory inputs to the agent, and the model of the environment. In this setting, IRL was studied by Ng and Russell [2], who developed algorithms based on linear programming (LP) for finite state spaces, and Monte-Carlo simulation for infinite state spaces. Abeel and Ng [3] extended the framework to the task of apprenticeship learning where the goal is to use observations of an expert's actions to decide the behavior of the agent. More recently Neu and Szepesvari developed a unified framework for the analysis and evaluation of recent IRL algorithms [4].

So far, IRL methods have been studied and employed in the context of a single agent. The assumption is that a single agent optimizes some criteria and the task is to observe the actions of the agent to learn its optimization function. Though this remains an interesting problem and has deservedly received attention in recent times, there are several real-world scenarios in which *multiple* agents will act *independently* to achieve a common goal.

Consider the example presented in Figure 1. In this scenario, there are 4 agents ($\langle S_1, S_2, S_3, S_4 \rangle$) that control the signals at 4 intersections. The intersections controlled by agents S_1 and S_4 are near a highway and their preferences are different compared to those of S_2 and S_3. While all agents act in a locally optimal manner (i.e., each of them individually optimizes traffic at its own intersection), there is a necessity for co-ordination. It is conceivable, then, that there is a centralized controller that co-ordinates the actions of different agents so that they optimize the traffic over all the intersections.

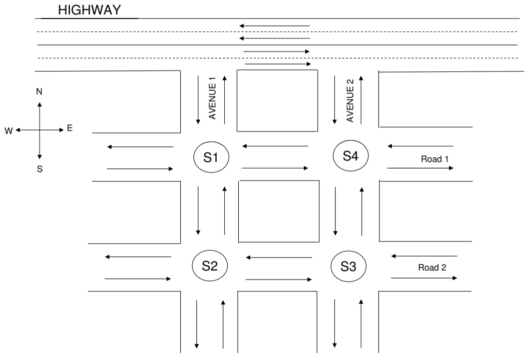

Figure 1. The traffic-routing domain: there are 4 agent-controlled intersections. Agents S_1 and S_4 are near the highway and have to be optimized differently compared to the other two.

We consider learning from situations similar to the scenario discussed above, which poses a significant challenge in the IRL setting. To see this, consider first, a straightforward solution: observe each agent individually and learn the reward functions for each independently. This is not always a good solution as the optimal behavior of one agent may be suboptimal for another. Yet another challenge is that we may never be able to observe the actions of the centralized controller directly but only the actions of the individual agents. Finally, considering the cross-product of the state and action spaces of the individual agents can lead to a prohibitively large space.

The goal of this work is observe the individual agents, learn the reward functions of all the agents, and then control the agents' policies in such a way as to optimize their *joint reward*. We consider weighting these agents, where some agent's policies are better optimized than others. For example, signals S_1 and S_4 are near a highway and the traffic entering (leaving) these two signals from (to) the highway needs to optimized, keeping into account that traffic densities on highways are higher than surface streets. Our framework provides a natural way of incorporating such differences as weights of individual agents.

This paper makes two key contributions: first, we consider the IRL problem in the presence of multiple agents that co-ordinate to achieve a common goal. More precisely, we assume that it is possible to observe multiple agents for a significant period of time and that there exists a centralized mediator who controls the policies of the individual agents, so that the (weighted) sum of the individual rewards are maximized. The goal is to determine the individual reward functions of the agents thus learning the reward function of the centralized controller.

The second contribution is the evaluation of the algorithms on a transportation domain where there are multiple traffic signals that co-ordinate through a centralized mediator. Given trajectories of the policies of different agents, we demonstrate that our algorithm learns a reward function that can imitate and improve upon the expert policy greatly. A minor yet novel contribution is the consideration of average-reward setting [5] for IRL. We formalize the derivation of IRL algorithm when the agents aim to maximize the average reward. It has been shown that average reward RL is more effective in many tasks where discounting can yield to myopic policies and hence a formal algorithm for inverse average-reward RL is crucial in solving several problems.

II. Average-Reward RL

An MDP is described by a set of discrete states *S*, a set of actions *A*, a reward function $r_s(a)$ that describes the expected immediate reward of action *a* in state *s*, and a state transition function $p^a_{ss'}$ that describes the transition probability from state s to state s' under action *a*. A policy, π, is defined as a mapping from states to actions, and specifies what action to execute in each state. An optimal solution in the average reward setting is the policy that maximizes the expected long-term average reward per step from every state. Unlike in discounted learning, the utility of the reward here is the same for an agent regardless of when it is received.

The Bellman equation for average reward reinforcement learning, for a fixed policy $\pi : S \rightarrow A$ is:

$$V^{\pi}(s) = r_s(\pi(s)) + \sum_{s'} P_{ss'}^{\pi(s)}\, V^{\pi}(s') - \rho, \qquad (1)$$

where ρ is the average reward per time step of the policy π. Under reasonable conditions on the MDP structure and the policy, ρ is constant over the entire state-space. The value function specifies that if the agent moves from the state *s* to the next state s' by executing an action *a*, it has gained an immediate reward of $r_s(a)$ instead of the average reward ρ. The difference between $r_s(a)$ and ρ is called the *average-adjusted reward* of action a in state s. $V^{\pi}(s)$ is called the bias or the value function of state s for policy π and represents the limit of the expected value of the total average-adjusted reward over the infinite horizon for starting from s and following π.

We use an average-reward version of Adaptive Real-Time Dynamic Programming (ARTDP) [6] called H-Learning for average-reward reinforcement learning [7]. The optimal policy chooses actions that maximize the right hand side of (1). Hence, H-learning also chooses greedy actions, which maximize the right hand side, substituting the current value function for the optimal one. It then updates the current value function as follows:

$$h(s) \longleftarrow \max_{a} \left\{ r_s(a) - \rho + \sum_{s'=1}^{n} p_{ss'}(a)\, h(s') \right\}. \qquad (2)$$

The state-transition models p and immediate rewards r are learned by updating their running averages. The average reward ρ is updated using the following equation over the greedy steps, where α is a tunable parameter.

$$\rho \longleftarrow \rho\,(1-\alpha) \;+\; \alpha\;(\,r_s(a) - h(s) + h(s')\,)\,. \qquad (3)$$

For the multi-agent case, we use vector-based reinforcement learning [8], [9]. Each agent is assumed to be a component and the central controller picks actions according to a weighted sum of the individual rewards. The rewards are divided into M types, where M is the number of agents. We associate a weight with each type, which represents the importance of that reward. Given a weight vector $\mathbf{w}$ and the MDP defined earlier, a new "weighted MDP" can be defined where each reward $r^i_s(a)$ of type i is multiplied by the corresponding weight w_i. We call the average-reward per time step of the new weighted MDP for a policy, its *weighted gain*. The goal of multi-agent RL is to find a policy for the central controller that optimizes the weighted gain. Since the transition probability models do not depend on the weights, they are not vectorized in H-Learning. The update equation for vector-based H-Learning is:

$$\mathbf{h}(s) \longleftarrow \mathbf{r}^a(s) + \sum_{s'=1}^{n} p_{ss'}(a)\,\mathbf{h}(s') - \boldsymbol{\rho}, \tag{4}$$

where

$$a = \arg\max_a \left\{ \mathbf{w} \cdot \left(\mathbf{r}^a(s) + \sum_{s'=1}^{n} p_{ss'}(a)\,\mathbf{h}(s') \right) \right\}, \tag{5}$$

and $\boldsymbol{\rho}$ is updated using

$$\boldsymbol{\rho} \longleftarrow \boldsymbol{\rho}\,(1-\alpha) + \alpha\,(\,\mathbf{r}^a(a) - \mathbf{h}(s) + \mathbf{h}(s')\,) \tag{6}$$

III. Multi-Agent Inverse Average Reward RL

The goal of inverse RL is to find a reward function that faithfully explains the observed behavior of the agent, or more specifically by observing a few trajectories. The inverse problem of learning from trajectories can most typically be setup using the Bellman equation (1) to obtain an optimization problem which can be solved for the reward function. As observed by Neu and Szepesvari [4], most IRL methods can be understood as minimizing a measure of distance between the reward function and the observed trajectories i.e., the goal is to determine a reward function such that the trajectories generated using this new reward function will be similar to the observed trajectories. We derive the algorithm in a manner similar to [2], but for the multi-agent case (vector-based inverse RL).

The Bellman equation for the value of a state, in ARL formalism, is given by (1) and, in the multi-agent case, becomes:

$$\mathbf{v}^{\pi}_s = \mathbf{r}^{\pi}(s) + \sum_{s'} P^{\pi(s)}_{ss'}\,\mathbf{v}^{\pi}_{s'} - \boldsymbol{\rho}^{\pi}, \tag{7}$$

where $\mathbf{v}^{\pi}_s$ is the value vector of executing an action $\pi(s)$ in the current state s. The central controller chooses the best action according to $a' = \arg\max_a \{\mathbf{w} \cdot \mathbf{q}^a_s\}$, where $\mathbf{q}^a_s$ is the value vector of executing an action a in state s and is equal to $\mathbf{v}^{\pi}_s$ when $\pi(s) = a$. Note that the action a we denote here is the joint action over all the agents and is composed of individual actions. In the rest of the paper, we denote the action of the central controller as a and refers to the joint action over all the agents (which can be different individually). Let P_a represent the transition matrix corresponding to $p^a_{ss'}\ \forall s, s'$. Let $\pi(s) = a_1$. Rewriting (7) more compactly as $(I - P_{a_1})\,\mathbf{v}^{\pi}_s = \mathbf{r}^{\pi}(s) - \boldsymbol{\rho}^{\pi}$, and we have:

$$\mathbf{v}^{\pi}_s = (I - P_{a_1})^{-1}\,(\mathbf{r}^{\pi}(s) - \boldsymbol{\rho}^{\pi}). \tag{8}$$

In the equation above, I is the identity matrix. For the expert to choose action a_1 over all other actions, the value of executing this action must be higher than the values of executing all the other actions. Consequently, we have, $\forall a \in A \setminus a_1$,

$$\mathbf{r}^{\pi}(s) + \sum_{s'} P^{a_1}_{ss'}\,\mathbf{v}^{\pi}_{s'} - \boldsymbol{\rho}^{\pi} \;\; \geq \;\; \mathbf{r}^{\pi}(s) + \sum_{s'} P^{a}_{ss'}\,\mathbf{v}^{\pi}_{s'} - \boldsymbol{\rho}^{\pi} \tag{9}$$

Rewriting (9) as $P_{a_1}\mathbf{v}^{\pi}_{s'} \;\geq\; P_a\mathbf{v}^{\pi}_{s'}$,

$$(P_{a_1} \,-\, P_a)\,\mathbf{v}^{\pi}_{s'} \geq 0. \tag{10}$$

Equation (10) gives the optimality conditions for the current action a_1. Combining (8) and (10) we get

$$(P_{a_1} - P_a)\,(I - P_{a_1})^{-1}\,(\mathbf{r} - \boldsymbol{\rho}) \geq 0, \quad \forall a \in A \setminus a_1, \tag{11}$$

where, $\mathbf{r} - \boldsymbol{\rho}$ is the average adjusted vector. In the case where there are multiple agents (say M agents), we can replace this term with a weighted sum,

$$\boldsymbol{\theta} = \sum_{i=1}^{M} (\mathbf{r}_i - \boldsymbol{\rho}_i)\,w_i = \sum_{i=1}^{M} \boldsymbol{\theta}_i w_i = \Theta\,\mathbf{w}, \tag{12}$$

where $\boldsymbol{\theta}_i$ denotes the average-adjusted reward due to the agents $i = 1, \ldots, M$, and the matrix Θ collects all individual agent rewards as columns. The adjusted reward of the central controller is then, a weighted sum of the adjusted rewards of the individual agents indicated by the dot product. We now arrive at the following equation,

$$(P_{a_1} - P_a)\,(I - P_{a_1})^{-1}\Theta\,\mathbf{w} \geq 0 \quad \forall a \in A \setminus a_1. \tag{13}$$

Equation (13) is very similar to the discounted-reward condition derived in [2], the key differences being: first, in the our setting, there is no notion of a discount factor (γ), and second, the multi-agent case is vector based. We can now formalize a theorem for the multi-agent average reward case similar to the one presented in [2].

Theorem 3.1: Given a finite state space S, a set of actions $a_1, ..., a_n$ and the transition probabilities P_a, an action a_i is optimal for the current state if and only if, $\forall a \in A \setminus a_i$, the average adjusted reward vector θ, defined in (12), satisfies

$$(P_{a_i} - P_a)\,(I - P_{a_i})^{-1}\Theta\,\mathbf{w} \geq 0 \quad \forall a \in A \setminus a_i. \tag{14}$$

Note that we can obtain empirical estimates of P_a for every action and ρ from observing the agent demonstrations i.e., the sample trajectories. From (13), it is clear that $\boldsymbol{\theta}_i = \mathbf{0}$ is a possible solution to the problem which corresponds to estimating the reward of each agent by setting $\boldsymbol{\theta}_i = \mathbf{r}_i - \boldsymbol{\rho}_i = \mathbf{0}$. This trivial solution is theoretically feasible but far from ideal as it does not allow the separation of one policy from another; this is due to the fact that all the policies can become optimal and all the actions in a particular state become equally important. From a practical point of view, the degenerate solution is not a useful solution in many domains. We now proceed to outline the formulation.

Since we are interested in discovering the reward function corresponding to the optimal policy, it would be reasonable to search for the reward function that satisfies (13) and maximizes

$$\sum_{s\in S} \left(\mathbf{Q}^{\pi}(s,\, a_1) - \mathbf{Q}^{\pi}(s,\, a) \right),\ \forall a \in A \setminus a_1. \qquad (15)$$

where $\mathbf{Q}^{\pi}(s,\, a)$ is the value vector corresponding to executing the action a in state s. The above equation seeks to maximize the difference between the value of executing the optimal and all the other actions over all the agents; a better objective would be to maximize the difference between the value of choosing the optimal and value of choosing the second-best action:

$$\sum_{s\in S} \left(\mathbf{Q}^{\pi}(s, a_1) - \max_{a\in A\setminus a_1} \mathbf{Q}^{\pi}(s, a) \right). \qquad (16)$$

It is fairly straightforward to turn (16) into an optimization problem constrained by the characterization of optimal policies from Theorem 1. Like most inverse problems, the resulting optimization might be ill-posed resulting in many "optimal" solutions. Consequently, in addition to maximizing (16), we also minimize a scaled regularization term, typically some norm of the reward, $\lambda \left\|\theta\right\|$. The resulting problem is to

$$\begin{aligned}
\max_{\boldsymbol{\theta},\, \boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_M} \ & -\lambda \ \left\| \boldsymbol{\theta} \right\| + \\
& \sum_{i=1}^{N} \min_{a\in A\setminus a_1} \left(P_{a_1}(i) - P_a(i)\right) \left(I - P_{a_1}\right)^{-1} \boldsymbol{\theta} \\
\text{s.t.} \quad & \left(P_{a_1} - P_a\right) \left(I - P_{a_1}\right)^{-1} \boldsymbol{\theta} \geq 0, \forall a \in A \setminus a_1, (17) \\
& \boldsymbol{\theta} = \sum_{i=1}^{M} \boldsymbol{\theta}_i \, w_i, \\
& |\theta_i^j| < \theta_{\text{max}}, \forall i = 1,\ldots,M,\, j = 1,\ldots,N
\end{aligned}$$

where N is the number of states, θ_{max} is an upper bound on the value of the adjusted reward and $P(i)$ is the i-th row of the probability matrix P. In (17), we do not specify the norm. As we show in our experiments, various reward regularizers yield optimal rewards with different properties. For instance, if the L_1-penalty is used, it enforces sparsity and only some of the $\boldsymbol{\theta}$ components will be non-zero resulting in the optimal reward being expressible by a sparse set of states.

If the number of components are very large, then we can sample a few states and ensure that the constraints are satisfied on those states. For instance, in [2], k Monte-Carlo trajectories under the policy π were created, and for each trajectory, the values were the average empirical estimates under π; this allows us to maximize the difference between the observed value function and the true value function. But, in our domains, we were able to handle large state spaces without the need for sampling.

The formulation (17) contains the quadratic term $\Theta\, \mathbf{w}$, and in its full general setting can lead to an optimization problem with quadratic constraints that can become highly intractable. We assume that each agent is weighted equally ($w_i = 1/M$) and hence (17) becomes either an *linear program* (LP) or a *quadratic program* (QP) depending on the regularization. Furthermore, if we assume an agent-wise decomposition of the state-space (as is the case in our traffic-signal control domain), with a slightly different choice of objective, (17) decomposes into M separate optimization programs as shown below:

$$\begin{aligned}
\max_{\boldsymbol{\theta}} \quad & -\lambda \left\|\boldsymbol{\theta}\right\| + \\
& \sum_{i=1}^{N} \min_{a\in A\setminus a_1} \ \left(P_{a_1}(i) - P_a(i)\right) \left(I - P_{a_1}\right)^{-1} \boldsymbol{\theta} \\
\text{s. t.} \quad & (P_{a_1} - P_a)(I - P_{a_1})^{-1} \boldsymbol{\theta} \geq 0,\ \ \forall\, a \in A \setminus a_1, \\
& |\theta_i| \leq \theta_{\text{max}},\ \ \forall\, i = 1,\ldots,N.
\end{aligned} \qquad (18)$$

An expert reader would deduce correctly that the single agent average reward inverse RL problem is a sub-case of our formulation. Note that while we can decompose the problem into several subproblems and solve them independently, there is no inherent assumption that the states and actions of the individual agents must be exactly similar. The only assumption is that the central controller's action can be decomposed into individual agent actions.

Some of the key features of the formulation:

(1) The formulation is similar to the one derived in [2]. This is a very nice property because it provides justification to the average reward setting by drawing similarities with the discounted setting. As we have mentioned earlier, while the discounted methods are widely popular due to their strong theoretical properties, average reward models have been shown useful in practice and reflects the fact that this work formulates the average-reward IRL problem based on the discounted setting.

(2) The formulation (18) is solved to obtain the adjusted reward, θ. Given trajectories, it is possible to estimate the average reward $\hat{\rho}$ and then compute r, or just use the adjusted rewards θ directly after learning to act optimally. Note that the probabilities can be estimated from the trajectories as well using maximum likelihood estimation over state-action values.

(3) Regularization allows us to control the properties of the learned rewards. Using an L_1 regularizer allows us to look for highly sparse rewards. L_∞ on the other hand allows for bounding the maximum adjusted rewards and forces the learner to look for discriminating rewards. Using both these norms in (18) leads to a LP. It is also possible to use an L_2 regularizer, which leads to a QP.

(4) For an N state problem, P_{a_1} is a $N \times N$ probability matrix and each row sums to 1. Thus, the matrix $I - P_{a_1}$ has rank of at most $N - 1$, and is *never* invertible. One solution is to use unnormalized counts rather than probabilities but the matrix may become ill-conditioned.

Thus, in practice, we consider the matrix $I - (1-\epsilon)P$, for some small ϵ to improve conditioning. Note that $1-\epsilon$ should not be confused with the discount parameter (γ) used in the discounted reward setting. It is introduced to make the ill-posed inverse problem well-posed. The difference is further highlighted by the fact that ϵ is *not used for action selection* once the rewards are learned.

IV. Experiments

We designed and implemented a simulator for a traffic signal domain (Figure 1) in which there 4 intersections and each is an agent. Each agent controls the signals at that intersection. Each agent has 4 actions corresponding to the direction of the traffic that is allowed to move (that has the green). The possible directions are $N - S$ (and $S - N$), $E - W$ (and $W - E$), and 2 kinds of left turns : $N - W$ and $S - E$ turn green simultaneously while $W - S$ and $E - N$ turn green together. These signals are assumed to be mutually exclusive and the agent chooses 1 of these 4 configurations; the signal at that configuration remains green until the agent changes its action.

Cars are generated at random at different locations with random destinations. Each car is assumed to move at a constant speed along the shortest possible route to its destination. In this case, the car's shortest path is the one that minimizes the Manhattan distance between the source and the destination. The agents are present at the 4 intersections and control the signals as specified above. A centralized controller controls the actions taken by the different agents considering their requirements. The action of the centralized controller is observed by observing the actions of the different agents.

The state space of each agent is the density of cars $\{$*low*, *medium*, *high*$\}$ that will be served if each of the above configuration turns green. Hence the size of the state space 3^4 for each agent. The number of actions for each agent is 4 corresponding to the 4 configurations. It is clear that it is not possible to solve a single LP for all the agents together as the state space is exponential in the number of agents ($3^{4^4} \approx 4 \times 10^7$). Fortunately, we can consider each agent separately and then learn the reward function for each of them, finally combining them in the central controller.

The expert policy is coded as a decision-list for each agent. An example (partial) policy is:

```
If Config1 = H, action = 1
else if Config4 = H, action = 4
else if Config1 = M, action = 1
else if Config2 = H, action = 2...
```

The policies were designed based on the traffic requirements. As can be seen from Figure 1, it is important for agents S_1 and S_4 to optimize the traffic to and from the highway. Hence for these two signals, `Config1` (which corresponds to $N - S$ or $S - N$ direction) is of the utmost priority. With such policies, we generated about 15000 state-action pairs for each agent to solve the LP.

Results. We used different regularizers to solve the optimization problems. In addition, once the rewards were learned, we use the Boltzmann distribution over the values to obtain a smoother action selection function.

As can be expected, L_1 regularization results in sparse reward functions. L_∞, on the other hand, tries to maximize the rewards for highly visited states and drives the negative weights lower. Thus, rewards computed using L_∞ were very discriminative between the states. L_2 aims to derive a smooth non-sparse function. Hence, most states end up having non-zero values and the difference between the rewards in different states is not as high when compared to other regularizers. The difference in reward functions for a few selected states are presented in Figure 2. As can been seen, L_∞ exhibits the maximum range of values and hence is much more discriminative while L_2 due to the property of having a dense yet smooth function, has the smallest range of reward values.

Figure 3 presents the fraction of states in which the learner's policy matches the expert's policy as a function of the number of training examples (number of $\langle state, action, nextState \rangle$ tuples). As the number of examples increases, the behavior of the learner becomes increasingly similar to the expert. But when the number of examples increase beyond 8000, the learner deviates away from the expert policy due to the fact that the learner actually **improves** upon the expert policy.

The overall goal is to minimize the traffic at intersections. Hence, we measured the traffic densities (averaged over 20 time-steps) and present the results in Figure 4. We used 16000 training examples as input and used L_1 regularization for learning the reward functions. Once the rewards are learned, we used H-learning [7] with Boltzmann action selection mechanism. Initially the learner is very similar to the expert but it quickly learns to act optimally and minimizes the traffic congestion at the intersections. The expert being a decision-list always aims to optimize one intersection and hence allows the traffic to accumulate at the other signals. The learner, on the other hand, uses a distribution based on the values of different states and rotates the signals. As time increases, the number of vehicles at different signals increases drastically for the expert while remaining nearly constant for the learner. This experiment proves conclusively that the learner can not only imitate the expert, but also improve upon it.

The key reason why the learner outperforms the expert is that it explores using average reward RL to act in the environment. This exploration makes the policy optimize the expected returns, given the immediate rewards. Once the rewards are learned, the learner can try to find the best policy that the exploration allows, which can be better than

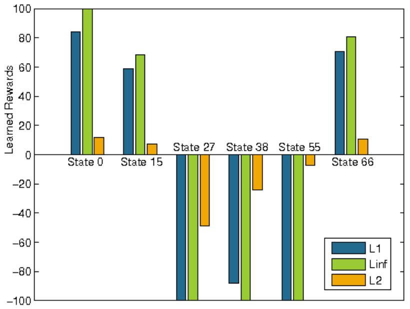

Figure 2. Rewards for different regularizations for a few selected states

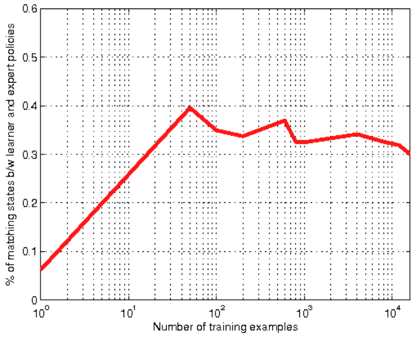

Figure 3. Fraction of states for which learner matches the expert vs. # examples

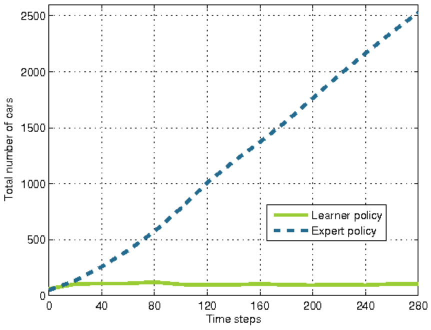

Figure 4. Total number of vehicles at the intersection (averaged over 20 time steps)

the expert.The expert on the other hand does not perform exploration. These results clearly prove that the use of RL makes it possible to design a learner that performs optimally even when learning from a sub-optimal expert.

V. Conclusions

Ng and Russell derived IRL algorithms in the discounted setting for a single agent [2]. They derived an LP formulation for finite state spaces and observed that the number of constraints can become infinite in the presence of infinite states and hence developed a Monte-Carlo based algorithm for IRL in infinite spaces. Abeel and Ng [3] extended [2] to imitate the expert's behavior. The inverse problem was posed as a quadratic program and solved using a SVM solver. Ratliff et al. [10] used IRL for imitation learning in robotics domain. They posed the problem of learning from an expert as a series of planning problems and use the max margin planning algorithm of [11] to learn the objective function. More recently, Neu and Szepesvari [**?**] considered the problem of training parsers as IRL. They consider PCFG parsing as a sequential decision making process and compared several IRL algorithms that learned the parsers from training data (similar to parser training). All these methods are closely related in that they derive linear programs or quadratic programs with linear constraints. Our work is motivated by all these methods but focuses on multi-agent average reward setting and the problem of traffic signal optimization.

IRL has not been explored in the multi-agent setting and we have derived and outlined an algorithm for multi-agent average reward IRL. Our experiments conclusively prove that the learner learns the correct reward function and uses RL to improve upon the expert. One of the assumptions was that the state space was completely observed. It is an interesting future direction of research to consider the problem of partial observability in IRL algorithms [12] and extend them to the multi-agent setting. It is also important to relax the assumption of weights being observed and jointly determine the weights and the reward functions that leads to a non-convex program. Yet another research problem is to combine prior knowledge about the domain with sample trajectories in learning the reward function. Finally, it is interesting to combine approaches that learn the rewards with the ones that explicitly learn the user policy such as policy matching [4]. This will allow the learner to imitate the expert as much as possible while allowing for exploration of unseen states thus improving upon the expert.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[2] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000.

[3] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004.

[4] G. Neu and C. Szepesvari, "Training parsers by inverse reinforcement learning," *Mach. Learn.*, vol. 77, pp. 303–337, 2009.

[5] S. Mahadevan and L. Kaelbling, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," 1996.

[6] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1–2, pp. 81–138, 1995, computational research on interaction and agency, part 1.

[7] P. Tadepalli and D. Ok, "Model-based average reward reinforcement learning," *AI Journal*, vol. 100, no. 1-2, pp. 177–223, 1998.

[8] Z. Gabor, Z. Kalmar, and C. Szepesvari, "Multi-criteria reinforcement learning," in *ICML*, 1998.

[9] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *ICML*, 2005.

[10] N. Ratliff, J. Bagnell, and M. Zinkevich, "Maximum margin planning," in *ICML*, 2006.

[11] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *NIPS*, 2003.

[12] J. Choi and K. Kim, "Inverse reinforcement learning in partially observable environments," 2009.