Giving Advice to Agents with Hidden Goals

Benjamin Rosman[1] and Subramanian Ramamoorthy[2]

***Abstract*— This paper considers the problem of providing advice to an autonomous agent when neither the behavioural policy nor the goals of that agent are known to the advisor. We present an approach based on building a model of "common sense" behaviour in the domain, from an aggregation of different users performing various tasks, modelled as MDPs, in the same domain. From this model, we estimate the normalcy of the trajectory given by a new agent in the domain, and provide behavioural advice based on an approximation of the trade-off in utility between potential benefits to the exploring agent and the costs incurred in giving this advice. This model is evaluated on a maze world domain by providing advice to different types of agents, and we show that this leads to a considerable and unanimous improvement in the completion rate of their tasks.**

I. Introduction

Consider an agent, the *explorer*, moving about a large hospital, with the intended destination of a particular ward, but no idea where to find that ward. Now consider a second agent, an autonomous *advisor* installed throughout the "intelligent building", was watching the progress of the explorer, and noticed that it seemed to be lost. In this case, it would be desirable to provide some advice on likely directions in which to move. Without knowing the intended target of the explorer, but by observing its behaviour, the advisor should make suggestions based on the most commonly used paths through the hospital – the "common sense" knowledge of the building. This could guide the explorer towards major wards and waiting areas, and away from service regions. This paper examines this question of how and when to provide advice to an agent with unknown goals.

Furthermore, the advisor does not know how much advice is needed by the explorer. If the explorer truly has no idea of what it is doing, then more information should be provided. On the other hand, especially if the advice is given via a mobile robot component, this is a costly resource that could be moved around to be shared between people – so, constant advice really is something to be minimised. Additionally, we assume that the advisor does not know if the agent is human or artificial, nor if it is acting randomly, with purpose, or learning. This situation is caricatured in Figure 1.

This is an important problem for both fixed-policy (single episode) and learning agents. With the help of advice, a fixed-policy agent can complete a task sufficiently faster than it would have otherwise. Similarly, advice can be a boost to a learning agent in terms of both speed and safety, in that exploration can be guided away from dangerous or useless regions of the domain.

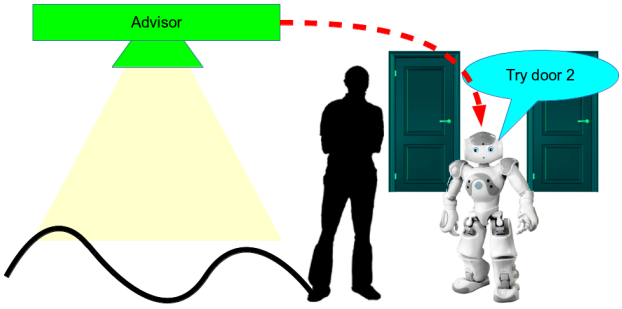

Fig. 1. The envisaged scene: an agent (the human) is exploring some environment. Its path is monitored by an autonomous advisor, which can then deploy some resource (a robot) to offer advice to the agent.

We propose to address this problem by learning a model of "common sense" behaviour over the entire domain, and marginalising over the different tasks. Offline, we learn a model of typical users in this domain in the form of distributions over action selection, from expert agents carrying out *different* tasks in the *same* domain. Online, a new user enters the domain, such that both the behavioural policy and the goal of this user are unknown *a priori*. The advisor continually estimates the performance of the user by comparing behaviour to the common sense model, and then based on this, provides advice in states where it is estimated to be needed. In this way, we regard advice as a data-driven policy selection mechanism for guiding an agent locally [1].

Basing advice on the behaviour of experts in the same space has limitations, in that it depends on the experts visiting the same locations and performing the same actions as would be required by the explorer. However, we wish to advise based on the way in which the domain is used – its *functional semantics*. If a particular task has not been seen before, there is nothing we can do regarding advising about that task, but there is still no need for the agent to collide with walls or enter dead ends. We consequently reason at the level of action distributions, rather than tasks, as we can leverage the fact that there are overlaps between the behaviours needed for different tasks.

Another example of this problem is that of building an autonomous advisor to teach humans or other autonomous agents to play complicated video games where game-play is not linear, and there are many different subgoals. Many modern games offer environments where players may progress through the game in any of a number of different ways,

[1]Benjamin Rosman is with the Institute of Perception, Action and Behaviour in the School of Informatics, University of Edinburgh, UK, and Mobile Intelligent Autonomous Systems (MIAS), CSIR, South Africa benjros@gmail.com

[2]Subramanian Ramamoorthy is with the Institute of Perception, Action and Behaviour in the School of Informatics, University of Edinburgh, UK s.ramamoorthy@ed.ac.uk

or even combinations thereof. Learning the combination of all these play styles and subgoals would not be feasible, particularly if the data is sparsely provided through very few instances of each. Consequently, learning a model of common sense behaviour provides the advisor with a compact description of reasonable courses of action.

After defining preliminaries in Section II, we discuss action priors as a model of common sense domain behaviour in Section III. Our general formulation and proposed approach are presented in Section IV, and experimental results are shown in Section V. Related work is discussed in Section VI, and concluding remarks are given in Section VII.

II. Preliminaries

In order to define a common sense model of agent behaviour, we assume that a number of agents perform various different tasks, all within the same domain. This domain is constant and is defined as $\mathcal{D} = (S, A, T, \gamma)$, and each different task is a Markov Decision Process (MDP) defined by $\tau = (\mathcal{D}, R, S_0)$. S is a finite set of states, A is a finite set of actions available to the agent, $T : S \times A \times S \to [0, 1]$ is the state transition function where $T(s, a, s')$ defines the probability of transitioning from state s to state s' after taking action a, $R : S \times A \times S \to \mathcal{R}$ is the reward function, where $R(s, a, s')$ is the reward received by the agent when transitioning from state s to s' with action a, and $\gamma \in [0, 1]$ is a discount factor.

A policy $\pi : S \times A \to [0, 1]$ for solving an MDP is a distribution over state-action space. The return, or utility, generated from an episode of running the policy π is the accumulated discounted reward $U^\pi = \sum_k \gamma^k r_k$, for the reward r_k received at step k. An optimal policy $\pi^* = \arg\max_\pi U^\pi$ is one which maximises the total expected return of an MDP.

III. Priors over Action Selection

A set of tasks, defined as MDPs in Section II, rely on access to a common domain $\mathcal{D}$, providing the infrastructure for the tasks to be situated within the same domain. This decomposes the environment such that S, A and T are fixed for all tasks, while each varies only in reward function R.

In order to advise the explorer, the advisor learns a model of "normal" behaviour in the domain. This is acquired by learning the *action priors* [2] of the domain. For an unknown but arbitrary set of tasks $\mathcal{T} = \{\tau\}$, each with a hidden goal, and corresponding expert policies $\Pi = \{\pi_\tau\}$, we learn for each state $s \in S$ a distribution $\theta_s(A)$ over the action set A. This distribution is the action prior, and represents the probability of each action in A being used by a solution policy traversing through state s, aggregated over tasks $\mathcal{T}$.

From a set of policies, each selecting actions in the same state s according to different distributions, a state-based model is learnt of typical behaviour marginalised over all known tasks in the domain, without requiring explicit knowledge or identification of those tasks themselves. Thus, given a state s, if one action is favoured by all trajectories through s, then that action should be preferred by any new trajectory exploring through s. Conversely, any action which is not selected by any trajectory passing through s is likely to have negative consequences, and should be avoided. By studying the policies from multiple tasks, a model of the structure of the underlying domain can thus be learnt, in terms of identifying the set of behaviours which are invariant across all tasks in the domain.

In general, different types of agents may have different distributions over their expected behaviours, e.g. hospital staff vs visitors. This can be taken into account in our framework by augmenting the state space with some description of the agent, if that sensing information is available, e.g. if they are wearing a uniform or name tag.

Although the mechanism of action priors is defined as learning from policies Π, we consider trajectories as samples thereof, and use these to update the action prior model. We define Π as a set of *expert* policies, where an expert is considered to be an agent trained at a particular task. This includes, but is not limited to, optimal policies for the task MDPs. These need not be optimal, and trajectories generated by any other systematic choice model, including those not explicitly optimising a utility function, are equally acceptable. The training data should provide a set of systematic biases which can be formalised as action priors, in order to see performance benefits from this advice.

A. Learning Action Priors

Consider a setting in which multiple agents have each solved potentially different tasks in $\mathcal{D}$, and have solution policies. For each state $s \in S$, model the action priors $\theta_s(a)$, $\forall a \in A$ using a Dirichlet distribution, by maintaining a count $\alpha_s(a)$ for each action $a \in A$. The initial values of $\alpha_s(a) = \alpha_s^0(a)$ are the hyperprior, initialised to any value. In this application, the hyperprior is used if the advisor is unsure whether or not it has received a sampling of trajectories which accurately approximate the underlying task distribution in the domain. Setting this hyperprior to a larger value will then prevent the action priors from over-generalising from incomplete data, and provide the agent with a possibility of finding a goal, even if that goal has not been witnessed before by the advisor.

The αs are learnt by the advisor from previous behaviours [2], and updated for each (s, a) for a new policy $\hat{\pi}$:

$$\alpha_s^{new}(a) \quad \longleftarrow \quad \begin{cases} \alpha_s(a) + 1 & \text{if } \hat{\pi}(s, a) > \delta_\theta \\ \alpha_s(a) & \text{otherwise,} \end{cases} \tag{1}$$

where $0 \leq \delta_\theta \leq 1$ is a threshold. $\alpha_s(a)$ reflects the number of times a was considered a good choice of action (as determined by δ_θ) in state s by *any* policy, added to the hyperprior priors $\alpha_s^0(a)$. Although δ_θ is domain dependent, in our application it is sufficient to set $\delta_\theta = 0$ if the policies are near-optimal.

The closed form of (1) is thus the number of known policies in which a is taken in s with probability greater than δ_θ:

$$\alpha_s(a) \quad = \quad \|\{\pi \in \Pi | \pi(s, a) > \delta_\theta\}\| + \alpha_s^0(a). \tag{2}$$

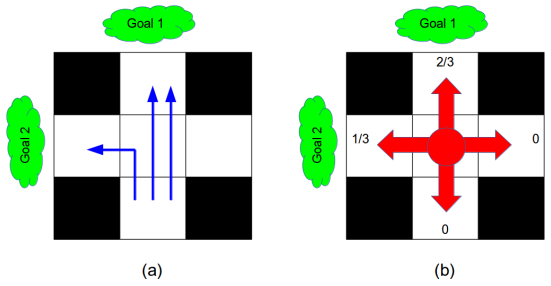

Fig. 2. An illustration of using the action priors for advice in a 3×3 region of a larger domain. (a) Assume three expert trajectories (shown in blue) have passed through this region. (b) When providing advice to an agent situated in the centre cell, the suggested probabilities (shown in each cell) of taking each direction are computed from the behaviour of the previous trajectories.

To obtain the action priors $\theta_s(a)$, sample from the Dirichlet distribution: $\theta_s(a) \sim \text{Dir}(\alpha_s)$. As $\theta_s(a)$ is a probability distribution over A: $\sum_a \theta_s(a) = 1$, $\forall s \in S$.

In (1) the α counts are incremented by 1, rather than the probability $\hat{\pi}(s, a)$. This is because counts of the actions used by the different policies are accumulated, rather than simply averaging the individual policies. A simple average could result in the agent being drawn towards a local optimum in state space, by one policy dominating others, and subsequently being unable to escape from it. Instead each action is weighted by the number of tasks which require the selection of that particular action, which is then used as the prior of that action choice in that state over all tasks in the domain. Our key assumption is thus that common sense is modelled as prioritising actions lower if they are less commonly used.

IV. Advice Giving using Action Priors

To provide advice, we do not need to assume that the advisor has the same action set as the explorer. Instead, the advisor can provide advice over the *outcomes* of the actions [3]. In this way, we require that all agents must have state and action sets mappable to a common set used by the advisor. The implication of this is that the state and action representations used by each agent can differ, but effects of these actions should be common to all agents. For instance, each agent may have a different mechanism for moving, but they should all have actions with outcomes of movement in the same directions. In what follows, we assume the action sets are the same for all agents. The advisor also needs to be able to observe the outcomes of the actions taken by every agent. Learning the model of typical domain behaviours is not possible using observations of state alone as, for example, the advisor would be unable to distinguish between an agent repeatedly running into a wall, or standing still. An illustration of providing advice based on the action priors which are computed from previous trajectories through the space is shown in Figure 2.

The advisor's goal is to estimate how much help the agent needs by comparing behaviour to the action prior, representing common sense behaviour, and then based on this, to provide advice to the agent in critical locations.

A. What Advice to Give

When advice is given to the explorer, we assume it is given as the action prior at the state it currently occupies. In this way, we follow a model of providing *action advice* [4] as a method of instruction which does not place restrictive assumptions on the differences between the advisor and the explorer. The advice is thus a probability distribution over the action space, as obtained from the prior. If the agent is currently in state s, the advisor provides the advice $\theta_s(A)$.

The proposed protocol for using the offered advice is that the explorer should select the next action according to this advised distribution. This protocol for the advice to be provided as a distribution rather than recommending a single action allows the explorer to incorporate its own beliefs about the next action it should choose. In this way, the explorer may thus alternatively use this provided action distribution to update its own beliefs from which to select an action.

Note that if the explorer uses the proposed protocol of sampling an action from the offered advice, then the advisor could equivalently have sampled from $\theta_s(A)$ and offered a single action. The choice of protocol here would depend on the mechanism used to convey the advice to the agent.

As described in Section III, the action priors describe a model of behaviours in this domain, marginalised over multiple tasks. As such, by providing a distribution of actions in accordance with this model, we inform the instantaneous motion of the agent such that it matches this common behaviour. This information will guide the agent towards regions of the domain in proportion to their importance for reaching the set of known domain goal locations.

B. When to Give Advice

There are a number of reasons not to provide advice to the explorer at every time step. Costs may be incurred by the advisor in terms of the resources required to display the advice to the explorer. In the hospital example, these could be ground lighting pointing the way, electronic signage, or dispatching a robot to the location of the agent. Additionally, interpreting this advice may cost the explorer time, and anecdotal evidence suggests that humans are easily annoyed by an artificial agent continually providing advice where it is not wanted.

Consequently, the advisor should provide advice only as it is needed. It thus computes an estimate of the amount of advice required, from the probability of the agent's trajectory under the action prior at time t. This probability is a measure of the agent's similarity to population behavioural normalcy. Because the action prior models normalcy in the domain, deviation from this corresponds to fault detection in the absence of task-specific knowledge.

Ideally advice should only be given if its benefit outweighs the cost incurred by giving advice. We assume a constant penalty of advising, κ. This could, e.g., be the average cost of dispatching a robot, or an estimate of user annoyance or likelihood of ignoring the advice. The benefit of advising at state s is the utility gain $\Delta U(s)$ from using the advice, rather than taking another action. Because we do not know

the actual values of any states, having learnt from expert trajectories only with no access to reward functions, we approximate this utility gain as a function of the difference between the probabilities of action selection under the action prior, and the expected action of the agent. This gives

$$\Delta U(s) \simeq KL\Big[\theta_s(A), P(A|s,H)\Big], \tag{3}$$

where $KL[\cdot,\cdot]$ is the KL-divergence. $P(A|s,H)$ is the expected action selection probabilities of the explorer in state s, having followed the state-action history H, computed by

$$P(A|s,H) \quad = \quad \int_M P(A|s,M)P(M|H)dM, \tag{4}$$

with $P(A|s,M)$ being the action selection probabilities in state s under a model M. $P(M|H)$ is the probability that the agent is selecting actions according to M, given the state-action history H, and is computed according to

$$P(M|H) \quad = \quad \frac{P(H|M)P(M)}{\int_M P(H|M)P(M)dM}. \tag{5}$$

Combining these forms a decision rule for giving advice. The explorer is advised if the following condition holds:

$$KL\Big[\theta_s(A), \int_M P(A|s,M)P(M|H)dM\Big] \geq \kappa. \tag{6}$$

Condition (6) requires a set of different behavioural models. The action priors provide a model of normal behaviour, but other models must be defined which describe different classes of behaviour. Herein we assume two models M:

1) M_1: Normalcy, modelled by action priors, empirically estimates the probability of selecting actions from the behaviour of expert agents performing various tasks.
2) M_2: Uniformity, which models the explorer not knowing what it is doing or where it is going, involves selecting actions with uniform probability.

Without further information, we assume that these two models have equal prior probability, and so $P(M_1) = P(M_2) = 0.5$. Also note that $P(A|s,M_1) = \theta_s(A)$. We choose to model a lost agent through M_2 as one that does not know what to do at all at the level of single actions. This could equally well be modified to work at the level of macro-actions for temporally extended behaviours, taking into account the fact that action distributions at adjacent states may not be independent.

C. Complete Procedure

The full procedure consists of an offline training phase, and an online advising phase. These two phases may run concurrently, but whereas the online phase may last for only a single task, the training phase happens over a considerably longer duration, consisting of numerous agent interactions with the domain. The full system is shown in Figure 3.

- Offline: train the advisor by collecting trajectories from many different agents carrying out various tasks in the same domain. From these, the advisor learns the action priors as a model of typical behaviour (Section III).

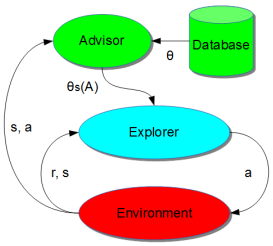

Fig. 3. The proposed system. The explorer interacts with the environment by executing an action a, and receiving a new state s and possibly a reward r. The advisor observes the agent, by receiving a copy of s and a. It evaluates the trajectory taken by the explorer, by comparing to the action prior θ obtained from a database of previous behaviours. If advice must be given, it is provided to the explorer in the form of the distribution $\theta_s(A)$.

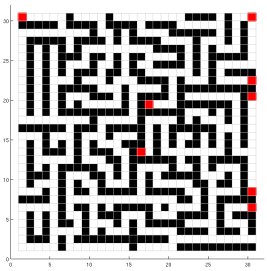

Fig. 4. The maze used in these experiments. White cells denote free space, black cells are obstacles, goal locations are the eight cells marked in red.

- Online: a new agent, the explorer, begins execution of some unknown task. The advisor continually evaluates the trajectory of the explorer, and if condition (6) is satisfied, the advisor provides advice to the explorer in the form of the action prior for the current state.

V. Experiments

A. Maze Domain

For our experiments we use a maze domain, being the 30×30 cell grid world shown in Figure 4, through which an agent navigates by moving at each time step in one of four cardinal directions. There are eight goals, selected randomly from corners and ends of corridors. Each agent starts at the lower left corner and must reach a randomly chosen goal. This is a delayed reward problem, with the agent receiving a reward of 1000 for reaching its goal, and 0 otherwise.

B. Agents

The offline training data which is provided to the advisor is a set of expert trajectories from agents moving towards each of the eight goals. From these the action prior model M_1 is learnt, to model "normal" behaviour in the domain.

Online experiments involve different types of explorers entering the maze, each with a randomly selected goal location. As a baseline, we consider the performance of an agent following an optimal trajectory. We do not know, *a priori*, how people behave and do not want to assume that people can easily zero in on a goal and execute an optimal policy. We thus experiment with various behaviour

Agent	Without Advice	With Advice
Random motion	50	900
Random with obstacle avoidance	50	750
Always turn left	150	750
Goal seeking bug	100	800
Q-learning	150	1000
Optimal trajectories	*1000*	*1000*

TABLE I

PERFORMANCE OF VARIOUS AGENTS ON THE MAZE DOMAIN

patterns to clarify how the algorithm behaves, although not all accurately model how humans and animals search. We consider the effects of advice on the following user models:

- Completely random: at every time step pick a cardinal direction at random and attempt to move in that direction.
- Random with obstacle avoidance: select actions at random, from those which will not result in a collision.
- Always turn left: use the common heuristic for navigating mazes: always take the leftmost path at an intersection.
- Goal seeking bug: heuristically move in the direction which will most reduce the difference in x- and y-coordinates between the agent and its goal.
- Q-learning: this non-stationary agent learns using ϵ-greedy Q-learning [5].

What is common to all of the fixed-policy agents is that when offered advice, they will all select an action according to the offered distribution. The Q-learning agent uses the advice as an exploration policy with probability ϵ [2].

Each fixed-policy agent performs 20 random *one-shot* tasks of up to 5,000 steps, with and without advice. Note that each agent has only one episode to complete each task. The learning agents learn for 5,000 episodes, where each episode is limited to only 200 steps, making this a difficult learning task.

C. Results

The results for the various agents with and without advice are shown in Table I. Performance curves and probabilities that the agent is acting according to M_1 are shown in Figure 5 for the fixed-policy agents, and reward curves in Figure 6 for the learning agents. The probabilities in condition (6) are over a moving window of the most recent 300 time steps of the transition history H of the explorer, to avoid underflow.

Figure 5 shows the performance improvement when advice is given to the different types of agents, over performance without advice. Although the agents all follow different action selection procedures, injection of advice results in them resembling normal behaviour in the domain more closely, and subsequently improving performance significantly.

In Figure 6, note the similar improvement in the performance of Q-learning by guiding the learning agent towards sensible goal locations. The difficulty with a domain such as that in Figure 4, as seen by the fact that the non-advised agent is far from converged after 5000 episodes, is that

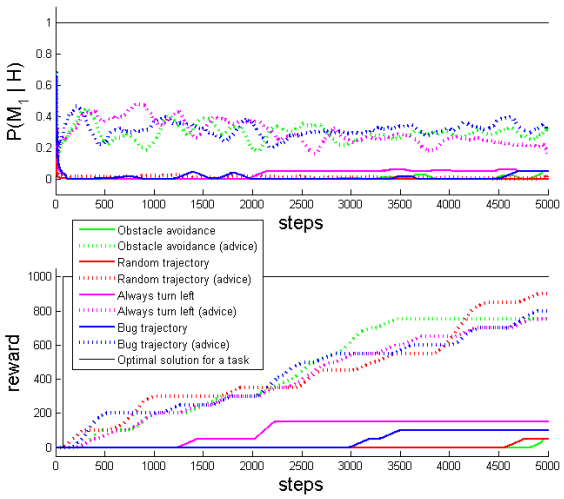

Fig. 5. $P(M_1|H)$ and rewards of different fixed-policy agents, averaged over 20 random tasks with $\kappa = 0.5$ Solid lines show non-advised performance, and dotted lines are advised. Results are smoothed for readability.

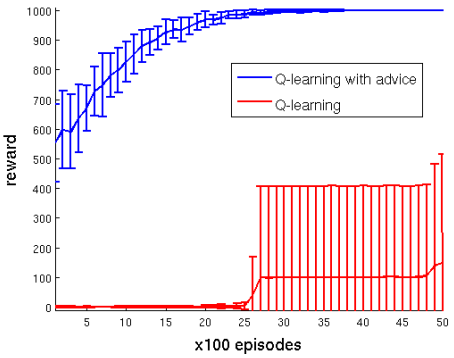

Fig. 6. Performance from advice giving with a learning agent, averaged over 20 tasks with $\kappa = 0.5$

a specific sequence of correct actions is required, and the standard learning agent does not receive information to guide its progress. Using advice, the learning agent can discover the correct goal location within a handful of episodes, something which is not possible otherwise.

Figure 7 shows the amount of advice given to the agents as κ in condition (6) is varied. κ is the cost of giving advice, to both the explorer and the advisor. As κ is increased, the quantity of advice provided decreases. Although not included here, we note that the Q-learning agent behaves initially as the random agent, and after convergence behaves as optimal.

VI. RELATED WORK

A similar problem to our own is that of a learning agent teaching another to perform a particular task [4]. Our problem differs, as our domain supports a number of different tasks, and so the advisor does not know the explorer's goal. Advice is more useful in our case as a distribution over action possibilities. Further, our problem does not explicitly restrict

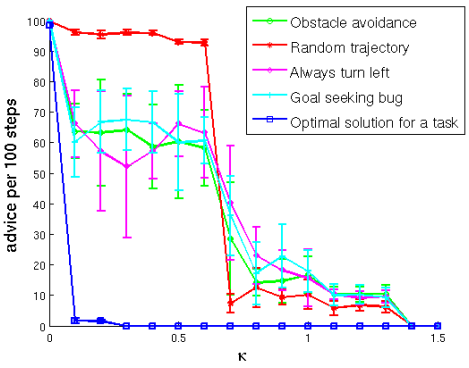

Fig. 7. Amount of advice given to different agents per 100 steps as a function of κ. $\kappa = 0$ implies that advice is always given. 10 trajectories of 1000 time steps were generated for each agent at each value of κ.

the amount of advice to a constant, which seems unnatural, but instead computes the utility trade-off from giving advice.

An alternative approach to learning this common sense model would be learning latent variable models to infer the current activities. One method is to learn the typical behaviours of individual agents (e.g. [6]), but our application assumes many unique users who are not active in the system for long enough to be modelled. Instead one could also learn models of all the different tasks in the domain, using e.g. hidden-goal MDPs. Again, we include the possibility in our problem of there being a large number of goals, and furthermore providing advice from hidden-goal MDPs is PSPACE-complete [7].

Having one agent advise another is a form of transfer learning [8], in that it is a mechanism for simplifying and speeding up the learning of a new task by using the expertise of one agent to bootstrap another. There are many other approaches to transferring knowledge in this way, with two common restrictions: 1) the advisor and learner agree on the task and goal, and 2) the amount of teaching/advice is unlimited. Common inter-agent teaching methods include imitation learning, or learning from demonstration [9], [10], and in particular to the reinforcement learning case, apprenticeship learning [11], [12], [13]. Finally, a teacher may also instruct a learner in a more passive manner by deconstructing the main task into a sequence of simpler tasks of increasing difficulty. This approach is typically referred to as reward shaping or curriculum learning [14], [15], [16], [17].

VII. Conclusion

This paper examines how to teach an agent through giving it advice while it performs unknown tasks in a known domain. This is an important issue in the current trends of collaborative and social robotics, where a human in some building could perform more efficiently when advised by a robotic agent installed building-wide. Having examined different agents performing different tasks in this common domain, we build a model of expected behaviour in the domain. New agents can then be compared to this model, to estimate how much teaching is required. To do so, we derive a condition which approximates the trade-off in utility gained from following the advice rather than continuing with its current course of action, against the utility lost as a result of providing the advice. If advice is required, the agent is provided with the distribution over actions suggested by the domain model at that state. This is shown to boost the performance of different fixed-policy and learning agents on random tasks.

Acknowledgements

The authors gratefully acknowledge the suggestions of the three reviewers. This work has taken place in the Robust Autonomy and Decisions group within the School of Informatics, University of Edinburgh. Research of the RAD Group is supported by the UK Engineering and Physical Sciences Research Council (grant number EP/H012338/1) and the European Commission (TOMSY Grant Agreement 270436, under FP7-ICT-2009.2.1 Call 6, and SmartSociety Grant Agreement 600854, under a FET Proactive Call).

References

[1] O. Flemisch, A. Adams, S. R. Conway, K. H. Goodrich, M. T. Palmer, and P. C. Schutte, "The H-Metaphor as a Guideline for Vehicle Automation and Interaction," NASA, Tech. Rep. NASA/TM2003-212672, 2003.

[2] B. S. Rosman and S. Ramamoorthy, "What good are actions? Accelerating learning using learned action priors," *International Conference on Development and Learning and Epigenetic Robotics*, November 2012.

[3] A. A. Sherstov and P. Stone, "Improving Action Selection in MDP's via Knowledge Transfer," *AAAI*, pp. 1024–1029, 2005.

[4] L. Torrey and M. E. Taylor, "Teaching on a Budget: Agents Advising Agents in Reinforcement Learning," *International Conference on Autonomous Agents and Multiagent Systems*, May 2013.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[6] L. Liao, D. J. Patterson, D. Fox, and H. Kautz, "Learning and Inferring Transportation Routines," *Artificial Intelligence*, vol. 171, pp. 311–331, 2007.

[7] A. Fern and P. Tadepalli, "A Computational Decision Theory for Interactive Assistants," *Advances in Neural Information Processing Systems*, 2010.

[8] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains: A Survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[9] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, May 2009.

[10] A. N. Meltzoff, P. K. Kuhl, J. Movellan, and T. J. Sejnowski, "Foundations for a New Science of Learning," *Science*, vol. 325, no. 5938, pp. 284–288, July 2009.

[11] P. Abbeel and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning," *International Conference on Machine Learning*, 2004.

[12] U. Syed and R. E. Schapire, "A Game-Theoretic Approach to Apprenticeship Learning," *Advances in Neural Information Processing Systems*, 2008.

[13] B. S. Rosman and S. Ramamoorthy, "A Game-Theoretic Procedure for Learning Hierarchically Structured Strategies," *IEEE International Conference on Robotics and Automation*, 2010.

[14] G. D. Konidaris and A. G. Barto, "Autonomous shaping: Knowledge transfer in reinforcement learning," *Proceedings of the 23rd International Conference on Machine Learning*, pp. 489–496, 2006.

[15] T. Erez and W. D. Smart, "What does Shaping Mean for Computational Reinforcement Learning?" *International Conference on Development and Learning*, pp. 215–219, 2008.

[16] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum Learning," *International Conference on Machine Learning*, 2009.

[17] W. B. Knox and P. Stone, "Interactively Shaping Agents via Human Reinforcement: The TAMER Framework," *International Conference on Knowledge Capture*, September 2009.