Imitation Learning over Heterogeneous Agents with Restraining Bolts

Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi
DIAG - Università di Roma “La Sapienza”, Italy
lastname@diag.uniroma1.it

Abstract

A common problem in Reinforcement Learning (RL) is that the reward function is hard to express. This can be overcome by resorting to Inverse Reinforcement Learning (IRL), which consists in first obtaining a reward function from a set of execution traces generated by an expert agent, and then making the learning agent learn the expert's behavior –this is known as Imitation Learning (IL). Typical IRL solutions rely on a numerical representation of the reward function, which raises problems related to the adopted optimization procedures.

We describe an IL method where the execution traces generated by the expert agent, possibly via planning, are used to produce a logical (as opposed to numerical) specification of the reward function, to be incorporated into a device known as *Restraining Bolt* (RB). The RB can be attached to the learning agent to drive the learning process and ultimately make it imitate the expert. We show that IL can be applied to *heterogeneous* agents, with the expert, the learner and the RB using different representations of the environment's actions and states, without specifying mappings among their representations.

Introduction

Inverse Reinforcement Learning (IRL) consists in estimating a reward function from a set of traces captured during the execution of an agent's policy. IRL can be used in many application domains to implement forms of Imitation Learning (IL). In IL, an expert agent executes a possibly optimal policy, generating a set of execution traces which are exploited by a learning agent to reconstruct the reward function, in turn used to learn a (possibly optimal) policy that imitates the expert behavior. Providing examples of the (optimal) policy is a very convenient way to specify goals for the learning agent, in contrast to defining a reward function, which is typically cumbersome. Interestingly, expert and learner may be different kinds of agents, e.g., human and robot, executing tasks in different ways, i.e., with different action and perception abilities. Unfortunately, this prevents an off-the-shelf application of the IRL approach as, in the classical IRL setting, the expert and the learner must share the representation space (e.g., states and actions).

In this paper, expert and learner may have different capabilities representations of states and actions; thus, the learner cannot interpret the traces generated by the expert. To deal with this, we exploit the idea of Restraining Bolt (RB) (De Giacomo et al. 2019): a device, with its own sensors, that can be attached to a Reinforcement Learning (RL) agent, to constrain its behavior and make it fulfill desired temporal high-level goals. Such goals are expressed as formulas of linear-time temporal logic over finite traces, LDL_f (De Giacomo and Vardi 2013), over a set of fluents, generally different from the features used by the RL agent.

We consider a setting where the expert agent executes its policy, producing desired and undesired traces at its own representation level. From these traces, we obtain a deterministic finite-state automaton (DFA) accepting all the positive (desired) traces and rejecting all the negative (undesired) ones. The DFA thus represents (an approximation of) the expert's behavior. Then, based on a well-known equivalence between LDL_f and DFAs (De Giacomo and Vardi 2013), the DFA is incorporated into a RB attached to the RL agent, to make it learn a (possibly optimal) policy that imitates the expert's behavior.

The ability of the RB to guide learning for a RL agent, when both use different representations and with no explicit mapping between them, has been proven in (De Giacomo et al. 2019). Here, we exploit that result to implement an IL procedure where the specification of the transferred behavior is provided at a higher level wrt the states of the MDP. In other words, the low-level traces generated by the expert are transformed into high-level traces from the RB sensors. Once the high-level behavior is learned, this can be transferred to an agent with different capabilities. This process can be seen as IRL at the RB representation level: instead of estimating the reward function, we reconstruct the DFA associated with the goal formula and then use it for learning.

The main advantage of this setting is a higher modularity, as agents and RBs can be combined to form complex IL systems with minimal effort. Indeed, as discussed in (De Giacomo et al. 2019), a RL agent can be extended to receive signals from a RB in a domain-independent way, by simply extending its state with an integer variable to store (an encoding of) the current state of the RB's DFA.

Summarizing, our contribution is an IL technique for agents with completely different state-action representations. The technique is based on the use of a RB to specify a high-level behavior and does not require any explicit mapping between the different representations.

Related work

Most IRL solutions model the reward function in parametric form (e.g., a weighted sum of reward features) and use some optimization or regression method to optimize the parameter values (see (Arora and Doshi 2018) for a recent survey of IRL solutions). The main issue with these approaches is that the optimization problem is essentially ill-posed, as many reward functions exist (including that with all null values) that can explain the observations, and defining metrics for their comparison is difficult (Ng and Russell 2000). E.g., two reward functions differing only in one state-action pair may produce considerably different behaviors. Although these solutions solve several issues in IRL, estimating numerical reward functions from execution traces remains an open problem. This paper aims, instead, at synthesizing the reward function at a logical level (the RB's representation level), avoiding numerical optimization and regression, thus overcoming their respective limitations. The proposed approach allows also for dealing with non-Markovian rewards.

Two broad classes of approaches, *passive* and *active*, exist to learn a (temporal) formula/DFA from sets of positive and negative traces. In passive approaches the formula/DFA is learned from a fixed set of positive and negative traces. Examples include (Camacho and McIlraith 2019) and (Heule and Verwer 2010). In the former, an LTL_f formula satisfied by all positive traces and by no negative trace is generated. In the latter, the problem is compiled into SAT and then, by exploiting the SAT technology, a minimum-size DFA (wrt number of states) accepting all positive and no negative traces is obtained. In *active learning*, the set of traces is produced as the result of an interaction between the learner and the expert. The distinctive feature of this approaches is the fact that the expert knows the target formula/automaton (which is not the case in this work). Angluin's L^* algorithm (Angluin 1987) (and later extensions) offers an example of this. While our work is agnostic to the learning technique, which is used in a black-box fashion, we resort to active learning, specifically Angluin's technique, using some care to overcome the fact that the expert does not know the target formula/automaton.

Problem definition

A Restraining Bolt (RB) is a tuple $RB = \langle \mathcal{L}, \{(\varphi_i, r_i)\}_{i=1}^m \rangle$ where each φ_i is an LDL_f formula over a set of fluents $\mathcal{L}$ and each r_i is a reward value. Fig. 1 illustrates the basic RB setting. This is a standard RL scenario, with the environment, the RL agent, its features and the reward function, extended with the RB, i.e., a device that observes the environment and, based on its own *fluents*, offers rewards to the agent. Fluents constitute the RB's representation of the environment state and need not match the RL agent features (and typically they do not). Formulas φ_i specify the behaviors that should be rewarded, each with its respective r_i. As known (De Giacomo and Vardi 2013) LDL_f formulas can be equivalently represented as DFAs. We use this representation. RBs were introduced in (De Giacomo et al. 2019), to constrain an agent's behavior to fulfill high-level (i.e., fluent-based) goals. We refer to that work for further details.

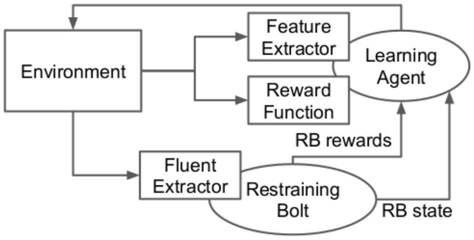

Figure 1: The RB setting

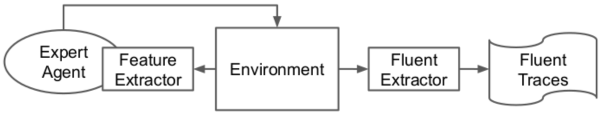

Figure 2: Trace generation for RB-IRL

We use RBs to address the problem of transferring a task from an expert to a learner agent. The task is represented by a formula φ in a RB or, more precisely, by the corresponding DFA Q. As a result, we consider RBs of the form $\langle \mathcal{L}, Q, r \rangle$, where Q is a DFA representing an LTL_f/LDL_f formula and r is a reward value associated with the accepting states of Q.

Consider now an expert agent defined on an MDP $\mathcal{M}_e = \langle S_e, A_e, Tr_e, R_e \rangle$. The agent can execute optimal policies of a given target task represented by a DFA Q, but cannot make the corresponding reward function explicit; in other words, the agent knows how to accomplish the task but cannot describe it. As the agent executes the policy, some traces are produced, some of which are desirable (posiive) and some other are not. The expert can correctly classify the traces as positive or negative, based on its own state representation.

On the other hand, the traces can also be seen from the RB perspective, through the RB sensors. Thus, from each state, the fluents can be extracted to produce the corresponding representation in the RB space. Notice the expert does not know anything about fluents, in particular, it cannot interpret them, as belonging to a different representation space. In fact, the expert is not even aware of the RB. This scenario is illustrated in Figure 2. Let $\mathcal{T}$ be a set of fluent traces collected while observing the behavior of the expert. The problem we address in this paper is that of reconstructing a DFA $Q_\mathcal{T}$ that is consistent with $\mathcal{T}$, i.e., that accepts all of its positive traces and none of its negative. The approach proposed in this paper allows for generating a new RB $RB = \langle \mathcal{L}, Q_\mathcal{T}, r \rangle$, where $Q_\mathcal{T}$ is the DFA built from $\mathcal{T}$, and r is a reward value associated with the accepting states of $Q_\mathcal{T}$. After the training phase, the generated RB can be placed on a learner agent to drive the learning process of a

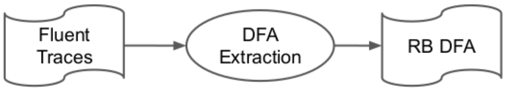

Figure 3: RB's DFA learning setting

behaviour imitating the expert's.

Consider a learner agent defined on $\mathcal{M}_l = \langle S_l, A_l, Tr_l, R_l \rangle$, with Tr_l and R_l unknown, equipped with the RB that encodes the behavior of the expert agent in performing the given task. The system $M_l^{RB} = \langle M_l, RB \rangle$ can be used to learn an optimal policy driven by RB, as explained in (De Giacomo et al. 2019). In this way, the behavior of the learner agent imitates that of the expert, when considering the evolution at the RB level.

Notice again that the state representations of S_e and S_l, as well as the set of actions A_e and A_l, may be completely different (e.g., states may come from different sets of state variables), as long as they allow to solve the task. Moreover, no explicit mapping between them is required.

Solution method

The core problem of our approach is extracting the DFA (or the formula) from the set $\mathcal{T}$ of (positive and negative) traces, to be incorporated in the RB used by the learning agent to learn the expert's behavior. This is illustrated in Fig. 3. Notice that the target DFA is unknown, even to the expert. As a result, the best we can hope for is to come up with an approximation. For this reason, we search for a DFA that accepts all positive and no negative traces, according to $\mathcal{T}$.

Since we aim at generalizing over the data $\mathcal{T}$, the obtained DFA must accept more traces than exactly the positive ones, and possibly reject more than the negative ones. As typical in learning, in order to guarantee a certain degree of generalization, some bias must be introduced. One reasonable approach is to check smaller DFAs (in terms of number of states) first. This is motivated by the intuition that smaller DFAs are less selective and tend to accept more traces than those with a large number of states. As a result, we expect them to generalize better than large ones.

As discussed, several approaches exist for extracting a DFA from a set of labelled traces. In our case, any is a reasonable candidate. For simplicity, we have selected L^* (Angluin 1987). The choice was due to the following reasons. Firstly, the algorithm returns a DFA (not a formula) that can be used as-is when executing the RB –in the case of a formula, instead, this should be translated into its equivalent DFA representation first. Secondly, the algorithm produces increasingly larger DFAs, thus satisfying the generalization requirement discussed above –though it is not guaranteed to return the minimal DFA. Thirdly, the technique has been implemented in a reliable software framework, *LearnLib*, actively tested and maintained, which has proven extremely convenient for the implementation step[1]. We remark that our approach is agnostic to the specific DFA/formula extraction technique.

The algorithm works as follows. The learner poses *membership* queries ("is this a positive trace?") and *equivalence* queries ("is this the target formula/automaton?") to the expert, which answers respectively with a "yes/no" and a "yes" or a counterexample ("no, because this trace should be/not be accepted"). The learner uses membership queries to produce a candidate DFA. Once done with this, the learner asks the expert whether the candidate solution is the target DFA. If the expert answers "yes", the DFA has been found and no other work is required; if the expert answers "no", it also returns a counterexample which is used by the learner, together with possible additional membership queries, to produce a new candidate solution to be checked for equivalence, and so on. The algorithm is shown to terminate and find the target DFA in polynomial time wrt both the size of the minimal DFA equivalent to the target DFA and the maximum length of any returned counterexample (Angluin 1987).

Unfortunately, in our case, the expert executes the policy offline, thus cannot be asked whether a certain trace is positive or negative, and, more critically, does not know the target DFA, thus cannot perform the equivalence check. As a consequence, the expert cannot act as the oracle required by L^* to answer the queries. Nonetheless, using the dataset $\mathcal{T}$, the oracle can be simulated in such a way that L^* generates a suitable approximation of the target DFA. This is done as follows: when a membership query is posed, the (simulated) oracle answers "no", if the input trace is classified as negative in $\mathcal{T}$, otherwise it answers "yes"; when an equivalence check is to be performed, the oracle answers "yes " if the candidate DFA accepts all positive traces and no negative trace from $\mathcal{T}$, and "no" otherwise, returning also one of the traces that made the test fail.

As it can be easily seen, with this trick we can simulate the required oracle and thus apply the algorithm. Also, by the choices above, the returned DFA is an approximation of the (unknown) target DFA, in the sense discussed above. Notice that we are potentially accepting all traces that are not explicitly forbidden by $\mathcal{T}$. Obviously, this is not the only option and other are possible (e.g., classify randomly the traces not in $\mathcal{T}$), yet it is a possible way to achieve a generalization wrt the data set.

Case studies

We showcase our approach in three scenarios: BREAKOUT (De Giacomo et al. 2019), SAPIENTINO (De Giacomo et al. 2019) and MINECRAFT (Icarte et al. 2018) (see Figure 4). For each scenario, we proceeded as follows.

Firstly, we fixed a restraining bolt in LTL$_f$/LDL$_f$, representing the target task, and we played a simplified version the game (that we call variant A), recording its traces (projected on fluents only) and labeling the good or bad according to the satisfaction of the formula. In this way, we generate an "expert behavior" that can be used later for assessing the quality of the policy learned. Then, we used the collected traces to generate a DFA that captures the expert's behavior, using the *LearnLib* implementation of Angluin's algorithm, as described in the previous section. Such a DFA typically is not the same as the LTL$_f$/LDL$_f$ formula but it is

[1] https://learnlib.de

Figure 4: Experimental scenarios: BREAKOUT, SAPIENTINO, MINECRAFT

close enough. Next, the learner learns a policy in the more complex game (variant B) using the learned DFA as the restraining bolt, using the same approach described in (De Giacomo et al. 2019). Finally, to assess policy learned we simulate its execution together with the original LTL_f/LDL_f DFA checking when we reach its final states.

Notice that for each scenario, the features and the actions in variant A and variant B are different. In particular, in variant A actions are stronger making the game easier. Notice that there must be a relationship (but not an isomorphism) between the actions in the two variants for making the approach effective in practice, but such a relationship can be quite loose and can remain unexpressed. Note also that actions are not used in the alphabet to progress the DFA and hence are not part of the reward given by the restraining bolt. This allows us to have different actions in the two variants. What is crucial for our approach to work is to have enough good and bad high-level traces to learn an accurate DFA. Once we get DFA, we assign a reward to the final states, and apply the restraining bolt techniques (De Giacomo et al. 2019).

We next describe each, scenario together with the corresponding variants and the target task. In all cases, the generated DFA was consistent with the target task and the learner was able to learn the task. The code can be found at https://github.com/whitemech/Imitation-Learning-over-Heterogeneous-Agents-with-Restraining-Bolts.

Breakout. This is the popular Atari game where a brick wall must be destroyed. In the original version, bricks can be removed by hitting them with a ball driven by a paddle placed at the bottom of the screen, that can move only horizontally. As **variant** A, we considered the game where there is no ball and the paddle can fire bullets to break the bricks. In this case, the state representation of the expert consists in the paddle position only. Notice that the brick configuration is not accessible to the agent (and neither is, consequently, the configuration of columns). As **variant** B, instead, we used the original version. In this variant, the learner can hit the ball with the paddle (but cannot fire) and can access the ball velocity and position. The **target task** is: *all bricks must be removed, completing the columns from left to right*, i.e., all the bricks in column i must be removed before completing any other column $j > i$. This task can be expressed with an LDL_f formula, using a fluent to represent the state of each column (completed or not).

Sapientino. SAPIENTINO is an educational game for 5-8 y.o. children where a small mobile robot must be programmed to visit specific cells in a 5x7 grid. Some cells contain concepts that must be matched by the children (e.g., a colored animal, a color, the first letter of the animal's name), while other cells are empty. The robot executes sequences of actions given in input by children with a keyboard on the robot's top side. During execution, the robot moves on the grid and executes an action (actually a *beep*) to announce that the current cell has been reached (this is called a *visit* of a cell).

As **variant** A, we took the scenario where the (expert) robot can move omni-directionally (actions: up, down, left, right). As **variant** B, we took the scenario where the (learner) robot can move differentially (actions: forward, backward, turn left, turn right). In both variants, the robot cannot see its position on the grid, nor can sense colors and/or concepts on the cells. The target task is: *visit a sequence of colors in a given order without bad beeps between the visits*.

Minecraft. In this scenario, an agent has to accomplish a task consisting in a sequence of *get resource* and *use tool* actions. A requirement to get resources and use tools is that the robot be on the cell associated with the resource or tool.

In **variant** A, the expert is endowed with "teleporting" capabilities (e.g. "go to a resource/tool"). In **variant** B, the learner can move only on the grid using differential drive, similarly variant B of SAPIENTINO.

Results of learning variant B tasks are similar to the ones presented in (De Giacomo et al. 2019) and are thus not reported here.

Conclusions

We have shown an approach based on the use of Restraining Bolts to perform Imitation Learning in a scenario where *heterogeneous* agents are involved, i.e., where, in particular, the expert and the agent do not share the same state representation nor a mapping between them. We do so by applying an approach based on Inverse Reinforcement Learning, where the behavior of the expert agent is learned and represented as a DFA that is then incorporated in a RB, in turn used by the learner at training time. Interestingly, the DFA constitutes a logical representation of the reward function, thus avoiding a number of problems arising when a numerical representation is adopted. We have performed experiments on several use-cases to show the effectiveness of our approach. Despite the differences in the state-action representation space, in all cases our approach was successful in transferring a task from the expert to the learner.

Future directions include testing different approaches to the generation of the DFA from a set of traces, as well as other approximation criteria.

Acknowledgments

Work supported in part by European Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (No. 834228), by the EU Horizon 2020 research and innovation program under AI4EU project (grant N. 825619), and by the Sapienza Project DRAPE: Data-awaRe Automatic Process Execution.

References

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Inf. Comput.* 75(2):87–106.

Arora, S., and Doshi, P. 2018. A survey of inverse reinforcement learning: Challenges, methods and progress.

Camacho, A., and McIlraith, S. A. 2019. Learning interpretable models expressed in linear temporal logic. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, 621–630.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019.*, 128–136.

Heule, M., and Verwer, S. 2010. Exact DFA identification using SAT solvers. In *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, 66–79.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Teaching multiple tasks to an RL agent using LTL. In *AAMAS*, 452–461.

Ng, A. Y., and Russell, S. J. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, 663–670.