Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description

Felipe Leno da Silva and Anna Helena Reali Costa
Escola Politécnica of the University of São Paulo, Brazil
{f.leno,anna.reali}@usp.br

ABSTRACT

The successful use of Reinforcement Learning in complex tasks depends on techniques to scale-up classical learning algorithms because they suffer from the curse of dimensionality. Transfer Learning approaches have been used to accelerate learning by reusing knowledge gathered from the solution of previous tasks. However, discovering how different tasks are related is a very complex undertaking if a human is not available (or is unable) to manually establish a mapping between tasks. We here propose an algorithm to autonomously estimate a Probabilistic Inter-TAsk Mapping (PITAM) across tasks described in an object-oriented manner, which requires less domain knowledge than a handcrafted Inter-Task Mapping. We also propose two strategies for Temporal-Difference algorithms to transfer knowledge using learned PITAMs. Our experiments evaluate varied scenarios in which the source and target tasks differ in several aspects, and our proposal presents benefits over both regular learning and *Q-value Reuse* using a detailed Inter-Task Mapping.

Keywords

Transfer Learning, Reinforcement Learning, Temporal-Difference Learning

1. INTRODUCTION

Reinforcement Learning (RL) [14, 25] agents aim at solving tasks with minimal input data. In order to solve a task, the agent has a description of its perceptions of the environment (states) and a set of actions that can be executed. As the state transition function is usually unknown, the agent gathers samples of interactions with the environment to learn how to maximize a *reward* signal, that encodes the agent performance. Even though RL has been used to solve challenging and increasingly complex tasks [16, 17, 24, 35], agents learning from scratch need a prohibitive number of samples to learn a good policy.

In order to alleviate this sampling complexity, Transfer Learning (TL) [32] approaches have been applied in RL to accelerate learning in a new task (target task) through the reuse of previous knowledge. This previous knowledge can be extracted from several sources, such as guidance from a human observer [1], action suggestions from another RL agent [37], previously solved (source) tasks [33], or a library of source tasks [6, 10]. However, in order to successfully transfer knowledge from one source to another, the agent has to reason about [32]:[1]

1. Which task (or set of tasks) among the already solved ones is appropriate to TL?
2. How the source task(s) and target task are related?
3. What to transfer from one task to another?

Even when a source task was already manually selected, finding the correct correspondences between tasks is critical to successfully transfer knowledge, as blindly transferring gathered information from one task to another is likely to cause *negative transfer* (i.e., the learning process is hampered instead of accelerated). If a human is available to map *how* the source and target tasks are related, she/he can define an *Inter-Task Mapping* [33], that specifies how the state-action spaces are related. If a human user is not available (or is unable) to hand-code such relationship, an Inter-Task Mapping can still be autonomously learned through a classification algorithm if the user is able to describe the environment as objects [34], which requires less domain knowledge and is easier to specify than an explicit mapping. However, as it is well known by the supervised learning community, a classifier requires a relatively high number of samples in order to provide an appropriate classification accuracy [15]. For example, Taylor *et al.*'s approach [34] requires a dataset of environment interactions in the target task to learn an Inter-Task Mapping.

We here argue that if tasks are described through *Object-Oriented Markov Decision Processes* [5], which is very similar to the object-based description used in Taylor *et al.*'s proposal [34], the domain knowledge contained in the task description allows to estimate an Inter-Task Mapping without the use of classifiers (and consequently without the need of interacting with the environment in the target task before transfer). We here introduce the Probabilistic Inter-TAsk Mapping (PITAM), a function that maps pairs of states across tasks to a probability value. We also propose a method, named *Zero-Shot Autonomous Mapping Learning*, to autonomously estimate a PITAM using object-oriented task description with no need of samples in the target task.

[1]We here focus on transfer across tasks, but the majority of the discussed issues are valid for all categories of TL.

Appears in: *Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL) at the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), A. Costa, D. Precup, M. Veloso, M. Taylor (chairs), May 8–9, 2017, São Paulo, Brazil.*

Finally, we propose two strategies to autonomously transfer knowledge using the learned PITAM. Our experimental evaluation in the *Predator-Prey* and *Goldmine* domains shows that our proposal has benefits in both jumpstart and performance at the end of the learning process. The evaluated scenarios took into account differences in state space size, reward and transition functions, and transfer across different domains.

The remainder of this paper is organized as follows. Section 2 discusses the basic RL and TL concepts, and presents related TL approaches; Section 3 defines PITAM, while Section 4 proposes two methods to autonomously use learned PITAMs to transfer knowledge across tasks; Section 5 presents our experimental evaluation; and finally Section 6 concludes the paper and points towards further works.

2. BACKGROUND AND RELATED WORKS

In this Section, we first present the basic RL concepts and then discuss the TL concepts and related works.

2.1 Reinforcement Learning

RL problems are usually framed as *Markov Decision Processes* (MDP) [19]. An MDP is composed of $\langle S, A, T, R\rangle$, where S is a (possibly infinite) set of environment states, A is a set of available actions, T is the transition function, and R is the reward function. At each decision step, an agent observes the state s and chooses an action a (among the applicable ones in s). Then the next state s' and reward signal r are defined and can be observed by the agent. This cycle is then repeated until the learning process is finished. Since T and R are unknown to the agent in learning problems, the agent must learn a policy π, that maps states to actions, through the observations of samples of $\langle s, a, s', r\rangle$, where s is a state in which the action a was executed, $s' = T(s, a)$, and $r = R(s, a, s')$. The solution of an MDP is an optimal policy π^*, i.e., a function that chooses an action maximizing future rewards at every state.

Temporal Difference (TD) algorithms [25] iteratively learn a Q-function, that maps each state-action pair to an expected long-term discounted sum of rewards: $Q : S \times A \to \mathbb{R}$. In finite MDPs, the Q-Learning [39] algorithm eventually converges to the true Q-function $Q^*(s, a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, which can be used to define an optimal policy $\pi^*(s) = \arg\max_a Q^*(s, a)$. However, learning Q^* requires a very long time in most domains, because every state-action pair must be visited many times in order to reach convergence. The *Object-Oriented MDP* (OO-MDP) [5] was introduced to abstract the state space and provide generalization opportunities at the cost of some additional domain knowledge in the environment description. In an OO-MDP, the state space is described through a set of objects, that follow a description of a class. $C = \{C_1, \ldots, C_c\}$ is the set of *classes*, where each class C_i is composed of a set of *attributes* denoted as $Att(C_i) = \{C_i.b_1, \ldots, C_i.b_b\}$, and each attribute b_j has a *domain* $Dom(C_i.b_j)$, specifying the set of values this attribute can assume. $O = \{o_1, \ldots, o_o\}$ is the set of objects that exist in a particular environment, where each object o_i is an instance of one class $C_i = C(o_i)$, so that o_i is described by the set of attributes from its class $o_i :: Att(C(o_i))$. Now, the environment state is described by the union of all object states: $s = \bigcup_{o \in O} o.state$, where an object state is the set of values assumed by each of its attributes at a given time $o_i.state = \left(\prod_{b \in Att(C(o_i))} o_i.b\right)$, hence $|s| = |O|$. Although relational representations such as OO-MDP can help to accelerate learning and to generalize task solutions [10], the convergence is still slow. Therefore, in order to be applied in complex domains, RL needs to be integrated with additional techniques to accelerate learning, such as TL.

2.2 Transfer Learning

The main idea of TL is to leverage knowledge gathered from previously solved source tasks in order to accelerate a target task learning. We here follow the formalism introduced by Lazaric [12], which for our purposes considers a task M as an *MDP*. The space of possible tasks to be presented to the learning agent is denoted as $\mathscr{M}$, and the environment $\mathscr{E}$, in which the agent is situated, presents tasks to be solved following an unknown distribution Ω, $\mathscr{E} = \langle \mathscr{M}, \Omega\rangle$. Notice that TL algorithms assume that the presented tasks follow a (unknown) distribution, since without a relation between tasks, no knowledge can be reused.

In order to solve an MDP, the agent maps a knowledge space $\mathscr{K}$ to a policy $\pi \in \mathscr{H}$, where $\mathscr{H}$ is the set of possible policies that can be chosen by the learning algorithm, $\mathscr{A} : \mathscr{K} \to \mathscr{H}$. $\mathscr{K}$ is all the available knowledge the agent has to infer a policy. When solving a single task, the set $\mathscr{K}$ usually contains only samples of interactions with the environment, which are used to learn a policy as discussed in Section 2.1. However, when a set of solved tasks $\mathscr{M}^{source} \subset \mathscr{M}$ is available, the agent can store a knowledge set $\mathscr{K}^{source} = \{\mathscr{K}^1, \ldots, \mathscr{K}^L\}$, $L = |\mathscr{M}^{source}|$ and use it as additional input to solve the target task: $\mathscr{A}_{TL} : \mathscr{K}^{source} \cup \mathscr{K}^{target} \to \mathscr{H}$.

However, deciding *when* and *what* to save in $\mathscr{K}^{source}$ during (or after) the training in source tasks, and *how* to relate previous tasks with the new one is a long studied question in the TL area, and this question has no single answer valid for all domains [18]. Previous works considered many ways to store and reuse useful information in $\mathscr{K}^{source}$, following varied representations and assumptions, as discussed in the next Section.

2.3 Related Works

Previous works have successfully transferred samples of low-level interactions with the environment [13, 27, 30], policies [6, 34], value or Q functions [28, 33], abstract or partial policies [10, 11, 36], action suggestions [1, 29, 38, 40], and heuristics or biases for a more effective exploration [3, 4], each of them presenting benefits over learning from scratch. The reuse of Q-functions has been of great avail to TD learning algorithms [27, 28, 33]. In addition to providing a significant jumpstart on the target task, Q-values are flexible enough to be reused by different TD learning algorithms. Furthermore, the transferred Q-function can still be refined in the target task to cope with possible differences between the tasks. However, as a consequence of being a relatively low-level information, the learning algorithm must be able to precisely translate state-action tuples from the source to the target task.

An *Inter-Task Mapping* [33] is an appropriate way to characterize *how* the tasks are related, and usually a pair of mappings $\mathcal{X}_X$ and $\mathcal{X}_A$ are defined. While $\mathcal{X}_X$ is used to map state variables across tasks: $\mathcal{X}_X(x_{i,target}) = x_{j,source}$, $\mathcal{X}_A$ is used to map actions: $\mathcal{X}_A(a_{i,target}) = a_{j,source}$. Then, a given Inter-Task Mapping can be used to TL through *Q-value Reuse*:

$$Q(s,a) = Q_{source}(\mathcal{X}_X(s), \mathcal{X}_A(a)) + Q_{target}(s,a). \quad (1)$$

Even when hand-coded Inter-Task Mappings are not available, they can be autonomously learned from the target task. *MASTER* [31] estimates a transition function using samples from the target task, and compares it with a similar model from source tasks to autonomously build Inter-Task Mappings. However, a relatively high number of samples in the target task is required in order to build the transition function model.

Mapping Learning via Classification [34] can learn an Inter-Task Mapping using less samples, at the expense of requiring more domain knowledge in task descriptions. The designer must describe the state space through "objects", composed of groupings of state variables, similarly to an OO-MDP description (Section 2.1). Then, a classifier is trained for each *object type* in the source task, in order to predict how each object is affected by state transitions: $C_{X,t}(s_{i,source}, r, s'_{i,source}) = i$, here $C_{X,t}$ is a classifier trained to identify objects belonging to type t, $s_{i,source}$ and $s'_{i,source}$ are state variables related to object i before and after the state transition, and r is the reward observed in the current step. After the learning process is finished in the source tasks, a classifier will be available for each object type. Once trained, those classifiers can be used to predict the correspondence of objects across tasks. Some interaction samples in the target task are stored, and a mapping is learned based on classifier outputs: $C_{X,t}(s_{i,target}, r, s'_{i,target}) = i$. Action mappings can also be learned by training a similar classifier for actions. However, although requiring less instances than MASTER, this method still needs samples in the target task.

We are here interested in autonomously learning Inter-Task Mappings and transferring knowledge across tasks, but without collecting samples in the target task. Sinapov *et al.* [23] proposed to perform Zero-Shot TL through the use of task features. However, the focus of their proposal is on the selection of source policies and not much discussion is devoted on how to map the similarities between tasks. Isele *et al.* [9] propose an algorithm to autonomously blend multiple source policies in a lifelong learning setting, allowing also the Zero-Shot initialization of a new policy for new tasks. However, they rely on task features which require that the parametrization of the current task is known beforehand, which we do not assume.

Our proposal uses approximately the same domain knowledge given in [34], but with no samples in the target task. In the next sections we describe our proposal, together with some suggested Q-value based transfer methods.

3. OBJECT-BASED PROBABILISTIC INTER-TASK MAPPING

We are here interested in autonomously learning an Inter-Task Mapping without interactions with the environment in the target task. We argue that estimating such a mapping is possible because the domain knowledge contained in an OO-MDP task description is able to determine an initial estimate of how the source and target tasks are related. Since the state space is described by objects, an Inter-Task Mapping function needs to relate objects between tasks.

Taking into account the OO-MDP representation (Section 2.1), the object-oriented description of source and target tasks may be different in regard to[2]:

- $C^{source} \neq C^{target}$: The set of classes may be different in the source and target class. As the two tasks must have similarities in order to enable TL, we assume that a class mapping $\mathcal{X}_C$ is given, and that $\exists C_s \in C^{source} : \mathcal{X}_C(C_s) \in C^{target}$.
- $\exists C_s \in C^{source}, C_t \in C^{target}, C_t = \mathcal{X}_C(C_s)$: $Att(C_s) \neq Att(C_t)$: The attributes of one or more classes were changed (either attributes were removed or added).
- $\exists att_s \in Att(C^{source}), att_t \in Att(C^{target}), att_t = att_s$: $Dom(att_s) \neq Dom(att_t)$: The domain of one or more attributes has changed.
- $|O^{source}| \neq |O^{target}|$: The number of objects in the environment has changed.

Therefore, the TL algorithm must be able to cope with those kind of differences across tasks. For our TL proposal, as we don't have samples of the target task to precisely identify the relation between attributes of objects and the reward and transition functions, we define a Probabilistic Inter-TAsk Mapping (PITAM), $\mathscr{P}_o$, instead of a deterministic one:

$$\mathscr{P}_o = S^{target} \times S^{source} \rightarrow [0,1]. \quad (2)$$

Remember that here the states are defined from sets of objects O^{target} and O^{source}. As we need $|O^{source}|$ objects in order to reconstruct a state in the source task, and $|O^{target}|$ may have a different value, a mapping $\mathscr{P}_o$ defines a probability of a state, i.e., a set of objects in the target task, $s_t = \bigcup_{o \in O^{target}} o.state$, being related to each possible set of objects in the source task, $s_s = \bigcup_{o \in O^{source}} o.state$, and this probability function may be used by the TL algorithm. We learn a mapping $\mathscr{P}_o$ by following Algorithm 1, named *Zero-Shot Autonomous Mapping Learning*. Here we consider that there is one source task and one target task. This procedure is called each time a new state in the target task is mapped. Firstly, the agent counts the number of objects of each class in the source and target task (n_s and n_t). Then, for each class C_i in the source task, the set P_{C_i} is built with all possible n_s-combinations of objects that belong to $\mathcal{X}_C(C_i)$ in the state s_t to be translated. After this process is executed for all classes in C^{source}, P contains sets of objects of all classes, and the $assemble()$ operation assembles states described with $|O^{source}|$ objects. This procedure is executed by considering all possible Cartesian products that contains one element of each $P_{C_i}, C_i \in C^{source}$. After the assemble operation is processed, we have Ψ, $|\Psi| \geq 0$, which is the set of states in the source task for which a mapping from s_t is defined with a non-zero probability. The mapping probability is related to a similarity metric calculated for each state in Ψ. This metric must reflect an estimate of how close a state in the source task is in regard to the state being mapped from the target task, and should take into account possible changes of class description and/or attribute domains. In this paper, we consider all states that exist in the source task as equally similar to the desired target state:

[2]If the source and target tasks have no similar classes, then it is not possible to infer a mapping and regular learning must be applied.

$$similarity(s_t, s_s) = \begin{cases} 0 & \text{if } invalidState(s_s) \\ 1 & \text{otherwise} \end{cases}, \quad (3)$$

where $invalidState(O)$ is true if $\exists o \in s_s, a \in Att(C(o)) : o.a \notin Domain(a)$. The probability value is finally defined according to the relative similarity of each state. The intuition behind this similarity metric is that all valid mapped states, in which all attribute values for all objects belong to the interval defined by the attribute domain in the *source* task, will be mapped with equal probability in $\mathscr{P}_O$. In case any attribute value falls outside the domain, the probability for that state is 0. In case a class definition in the target task has more attributes than in the source, the extra attributes are ignored. If the target task has fewer attributes, the ones that do not exist in the target task receive a valid standard value ($Unknown$, U_k, in our case). Other similarity metrics can be used, but Equation (3) was enough to achieve good results in our experimental evaluation.

The execution of Algorithm 1 is illustrated in the following example.

EXAMPLE 1. *Suppose that the source and target tasks have the classes* $C^{source} = \{Class1, Class2\}, C^{target} = \{Class1, Class2, Class3\}$, *with* $\mathscr{X}_C(Class1) = Class1$, *and* $\mathscr{X}_C(Class2) = Class2$. *The attributes are* $Att^{source}(Class1) = \{a1, a2\}$, $Att^{target}(Class1) = \{a1\}$, $Att^{source}(Class2) = \{b1\}$, $Att^{target}(Class2) = \{b1, b2\}$, *and* $Att(Class3) = \{c1\}$, *with domains* $Dom^{source}(a1) = Dom(a2) = \{0, 1, U_k\}$, $Dom^{target}(a1) = \{0, 2\}$, $Dom^{source}(b1) = \{0, 1\}$, $Dom^{target}(b1) = Dom(b2) = \{0, 1\}$, *and* $Dom(c1) = \{0, 1\}$. $O^{source} = \{o1_{s1}, o1_{s2}, o2_{s2}\}$, $O^{target} = \{o1_{t1}, o2_{t1}, o1_{t2}, o2_{t2}, o3_{t2}, o1_{t3}\}$, *where the index* $s1$ *means that the object belongs to* $Class1$ *from the source task and* $t1$ *means that the object belongs to* $Class1$ *from the target task.*

In the target task we want to translate the state $s_t = \{\langle 0\rangle, \langle 2\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle, \langle 0\rangle\}$. *When building* P_{Class1}, $n_t = 2$ *and* $n_s = 1$, *and hence the combinations are* $P_{Class1} = \{(\langle 0\rangle), (\langle 2\rangle)\}$. *On its turn,* $n_t = 3$ *and* $n_s = 2$ *when building* $P_{Class2} = \{(\langle 0, 1\rangle, \langle 1, 0\rangle), (\langle 0, 1\rangle, \langle 0, 0\rangle), (\langle 1, 0\rangle, \langle 0, 0\rangle)\}$. $Class3$ *is ignored because it does not exist in the source task.*

The assemble operation then builds states by combining each possible elements of P_{Class1} *and* P_{Class2}, *resulting in* $\Psi = \{(\langle 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle), (\langle 0\rangle, \langle 0, 1\rangle, \langle 0, 0\rangle), \ldots, (\langle 2\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle)\}$. *However, when computing the similarity metric,* $o2_{t1}$ *has value* $o2_{t1}.a1 = 2 \notin Dom^{source}(a_1)$.

Therefore, all states containing that object have 0 *similarity value and are removed from the mapping. The similarity function ignores* $b2$ *(because it does not exist in the source task) and initiates* $a2$ *with a standard value (because it does not exists in the target task). The following states have similarity value equals to* 1*:* $(\langle 0, U_k\rangle, \langle 0\rangle, \langle 1\rangle)$, $(\langle 0, U_k\rangle, \langle 0\rangle, \langle 0\rangle)$, *and* $(\langle 0, U_k\rangle, \langle 1\rangle, \langle 0\rangle)$. *Thus, these states are mapped with probability* $\frac{1}{3}$.

In the next section we describe approaches to transfer Q-values using a PITAM.

4. TRANSFERRING KNOWLEDGE USING PITAM

After defining $\mathscr{P}_o$, the agent can make use of the mapping to transfer knowledge from one task to another. Here, we assume that the learning agent uses a TD algorithm (thus a Q-function is available) and that a mapping $\mathscr{X}_a$ for actions is given (even though learning $\mathscr{X}_a$ is possible [34], so far no algorithm can learn such a mapping without samples from the target task).

Algorithm 1 Zero-Shot Autonomous Mapping Learning

Require: Set of objects in both tasks O^{source} and O^{target}, set of classes C^{source}, class mapping $\mathscr{X}_C$, current object-oriented state s_t in the target task.

for $\forall C_i \in C^{source}$ **do**
  $C_t \leftarrow \mathscr{X}_C(C_i)$
  // number of objects belonging to C_i *in each task.*
  $n_t \leftarrow numObj(O^{target}, C_t)$
  $n_s \leftarrow numObj(O^{source}, C_i)$
  // n_s*-combinations of objects from the target task.*
  $P_{C_i} \leftarrow combinations(s_t, n_t, n_s, C_t)$
// Assembles all possible states using objects from P.
$\Psi \leftarrow assemble(P)$
// Sum of similarity values.
$totalSim \leftarrow 0$
for $\forall s_s \in \Psi$ **do**
  // Equation (3)
  $totalSim \leftarrow totalSim + similarity(s_t, s_s)$
for $\forall s_s \in \Psi$ **do**
  $\mathscr{P}_o(s_t, s_s) \leftarrow \frac{similarity(s_t, s_s)}{totalSim}$

Algorithm 2 Q-table initialization

Require: Set of possible states S^{target} and S^{source}, action mapping $\mathscr{X}_A$, PITAM $\mathscr{P}_O$.

for $\forall s_t \in S^{target}$ **do**
  $\kappa \leftarrow \emptyset$
  // For all mapped states with a non-zero probability.
  for $\forall s_s \in S^{source}, \mathscr{P}_O(s_t, s_s) > 0$ **do**
    // Stores mappings to states in the source task.
    $\kappa \leftarrow \kappa \cup \langle s_s, \mathscr{P}_O(s_t, s_s)\rangle$
  for $\forall a \in A^{target}$ **do**
    // Calculates the resulting Q-value.
    $Q(s_t, a) \leftarrow initQ(\kappa, \mathscr{X}_A(a))$ ▷ Eq (4) or (5).

We here propose two strategies to initiate the Q-function in the target task: $QAverage$ and $QBias$. In both cases, the initialization is intended to better guide the exploration during learning. Algorithm 2 describes how this initialization works for both strategies. For each possible state in the target task, the agent defines the object-oriented representation of this state, and then uses it to get a mapping κ to object sets in the source task. κ is a set of tuples $\langle s_s, p\rangle$, where s_s is the object-oriented state in the source task and p is the probability defined by PITAM $\mathscr{P}_O$. κ is then used to define the initial value of the Q-function to s_t, which here is calculated for each strategy as follows:

1. **QAverage**: Here, all states with a non-zero probability defined by $\mathscr{P}_O$ contribute to a resulting Q-value, proportionally to their PITAM probability:

$$initQ(\kappa, a) = \sum_{\langle s_s, p\rangle \in \kappa} Q^{source}(s_s, \mathscr{X}_{\mathscr{A}}(a))p. \quad (4)$$

2. **QBias**: Alternatively, we can initiate only the Q-value of the best action with a small bias value b [4]. The main idea here is to initiate in the new domain reusing

the optimal policy, but at the same time introducing only a small value in the Q-table to enable faster policy refinements:

$$initQ(\kappa, a) = \begin{cases} b & \text{if } \mathcal{X}_A(a) = \\ \underset{a_s \in A^{source}}{\arg\max} \sum_{\langle s_s, p \rangle \in \kappa} Q^{source}(s_s, a_s), \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

The standard value for uninitialized Q-values is 0. Through the combination of PITAM with one of the aforementioned initiation strategies, we can autonomously transfer knowledge across tasks with no need of gathering samples of interactions in the target task. In the next sections we describe the Experimental evaluation to show the benefits of our proposal.

5. EXPERIMENTAL EVALUATION

In order to evaluate the effectiveness of our proposal under several conditions, we test the speed-up provided by the autonomously learned PITAM in varied modifications of two benchmark domains: *Predator-Prey* and *Goldmine*. The following algorithms were compared in our experiments: (i) **Regular Learning**: The standard Q-Learning algorithm; (ii) **QAverage**: An autonomously learned PITAM with the *QAverage* knowledge reuse strategy following Equation (4); (iii) **QBias**: Another PITAM-based algorithm, following the Q-table initialization described by Equation (5); and (iv) **QManualMapping**: A manually defined Inter-Task Mapping using Q-Value Reuse, as described by Equation (1).

For all experiments we set $\alpha = 0.1$, $\gamma = 0.9$, and all algorithms use an ϵ-greedy exploration with $\epsilon = 0.5$ decayed by 0.999 after each episode. The parameters were chosen in preliminary experiments. We describe the domains and the evaluated experiments in the following sections.[3]

5.1 Domains

In the *Predator-Prey* domain [4, 27], predators aim at collaboratively capturing all preys spread in the environment. While each predator is one RL agent in our experiments, preys move randomly at each step (preys could also be considered as reasoning agents, but in our experiments they simply apply random movements regardless of the situation). Our experiments are executed in a 10x10 grid, as depicted in Figure 1. At each step, Predators can apply one action to move towards a desired direction $A = \{North, South, West, East\}$. Agents can occupy the same position without punishment, and a prey is captured when it occupies the same position as one predator. In this domain agents cannot observe the whole grid, and their visual field is limited by a *depth* parameter, as shown in Figure 1 by the agent with $depth = 3$. The object-oriented representation is a *Multiagent OO-MDP* [22] defined as: $C = \{Prey, Predator\}$, and $Att(Prey) = Att(Predator) = \{x, y\}$. The agent can observe the relative position of objects inside its visual field. For example, the green predator state depicted in Figure 1 is observed as: {*Prey: (-1,2), Prey: (3,-1), Predator: (1,1), Predator: (2,2), Predator: (U_k,U_k)*}. Therefore, the *domain* of attributes depends on the visual depth: $Dom(x) = Dom(y) = \{-depth, \ldots, 0, \ldots, depth, U_k\}$. Predator-Prey tasks are hard to solve because the environment is non-stationary and partially observable. With an increase in the visual depth the task becomes solvable in less steps, because preys can be observed from a farther position, however, the state space is increased and it becomes harder to achieve convergence.

[3]Codes at `https://github.com/f-leno/TiRL_Leno_et_al`.

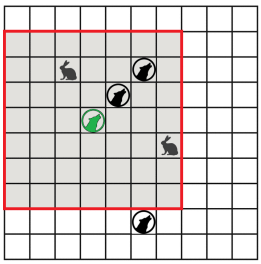

Figure 1: Illustration of the Predator-Prey domain. The red square depicts the visual field (depth=3) of the green predator.

In order to test the effectiveness of our proposal when transferring knowledge across tasks, we also consider the *Goldmine* domain in our experimental evaluation. A certain number of miners aim at collecting gold pieces that are spread in the environment. Miners are autonomous agents that can move towards any of the compass directions, and a gold piece is collected when a miner enter its position. The class definitions are $C = \{Miner, Gold\}$, and $Att(Miner) = Att(Gold) = \{x, y\}$. The miner who captures a gold piece receives +100 of reward, and the default reward is 0. A visual depth is again considered in this domain. Notice that the *Goldmine* and *Predator-Prey* tasks are similar, but *Goldmine* is much easier to solve because the environment is stationary and gold pieces do not move. Thus, the optimal policy is to move towards gold pieces as soon as they are seen. For all experiments, 100 evaluations episodes were carried out without exploration for each 10 learning episodes. For all source tasks, 2500 learning episodes were executed and then the learned Q-table is stored for posterior reuse. For all target tasks, 4500 learning episodes are executed in total. In all cases, when the agent is in a blind state (no observation is received), a random action is executed. All statistical significance tests were carried out by a 95% confidence Wilcoxon signed-rank test. The average performance of a random agent in the target task is roughly 76 steps in average. First we train a learning agent in a *Predator-Prey* task to be used as source for the first experiments. Figure 2a depicts the average number of steps to solve the task during learning. Even though the agent has not learned the optimal policy after episode 2500, the observed performance shows that it is possible to learn an effective policy in this task, as the performance achieved by the agent after the learning process is much better than of a random agent. In the next Sections we describe the evaluated scenarios and discuss the results. For the sake of simplicity we refer to the performance after 4500 learning episodes as asymptotic performance. Although convergence is not achieved yet after this amount of learning episodes, only small changes in performance are observed after it for all scenarios.

5.2 Exp1 - Scaling Tasks

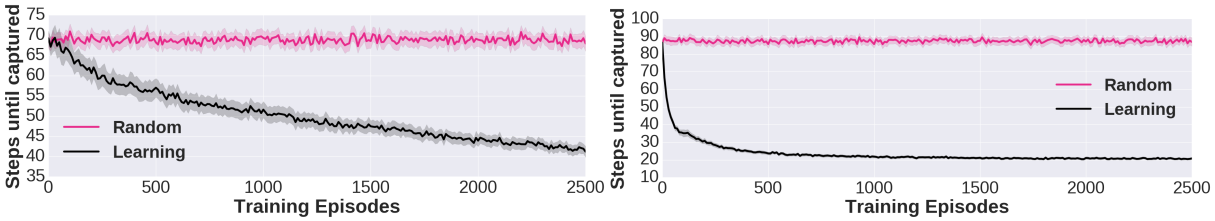

Figure 2: The average of steps to solve the task during evaluation episodes in 100 repetitions for: (a) *Predator-Prey* and (b) *Goldmine* domains. The shaded area corresponds to the 99% confidence interval.

In this first experiment we evaluate the effectiveness of each TL approach when the source and target tasks are in the *Predator-Prey* domain and differ only in attribute domains and number of objects, that is, the visual depth and number of predators/preys are increased without any change in transition and reward functions. For both tasks, all predators receive +1 of reward when a prey is captured, and 0 otherwise. While in the source task we train 3 predators trying to catch 1 prey with $depth = 3$, the target task has 4 predators trying to catch 2 preys with $depth = 4$. As we have no changes in the class and action sets, the mapping $\mathcal{X}_C$ for PITAM is a direct translation, and the mapping for *QManualMapping* is defined according to Table 1. Notice that this mapping relies on a domain-specific knowledge that the closest prey is more important to the value function, and such domain knowledge is not needed for PITAM.

Figure 3 shows the performance of each TL algorithm compared with regular learning. All TL algorithms present advantages in terms of jumpstart. While *Regular learning* (at first equivalent to a random actuation) solves the task after 76.21 steps on average, *QManualMapping*, *QAverage*, and *QBias* solve the task, respectively, in 62.43, 40.71, and 45.48 steps on average. Even though all TL techniques have a significantly better performance right after the beginning in the new task, $QManualMapping$ takes longer to improve its policy in the new task because the original Q-values in the source task are still used and harder to be replaced. The difference between *Regular learning* and *QManualMapping* is not statistically significant between 240 and 450 learning episodes, and *Regular learning* achieves a significantly better performance for the rest of learning process. On their turn, $QBias$ and $QAverage$ have a better asymptotic performance, roughly stabilizing on 33.50 and 30.03 steps, against the *Regular learning* performance of 45.67 steps. The difference between *QBias* and *QAverage* is statistically significant through the entire experiment. Note that, after 1500 learning episodes, the performance for *Regular learning* roughly plateaus before achieving the performance of PITAM-based algorithms.

5.3 Exp2 - Changes in Reward Functions

Here we evaluate the algorithms when the reward function of the target task is different from the source task (without many differences in the optimal policy). While the source task is the same as in Section 5.2, in the target task agents receive +100 of reward when a prey is captured, and $\frac{\sqrt{100}}{dist}$ otherwise, where *dist* is the distance to the closest prey.

Figure 4 shows that the relative results are very similar to the first experiment, which shows that all the evaluated TL approaches are robust in terms of simple changes of scale in reward functions across tasks. The main differences between experiments 1 and 2 are that here the standard deviation is higher, and the reward function in this task is less effective to guide the agent towards the task solution (which is evidenced when comparing the *Regular learning* performance in both experiments). However, TL algorithms remain better than *Regular learning*. *QAverage* and *QBias* are again better in both jumpstart and asymptotic performance. The difference between *QAverage* and *QBias* is only significant until episode 200. After that, they have a equivalent performance. Here, *QManualMapping* is significantly better than *Regular learning* during the whole experiment.

Table 1: Manual Inter-Task Mapping translating the target task with 2 preys and 4 predators to a source task with 1 prey and 3 predators. Here, z is the local agent, $Prey$ is the set of prey objects, $Predator$ is the set of predator objects, and *dist* is an Euclidean distance function.

source task	**target task**
prey1	$arg\min_{p \in Prey} dist(z,p)$
predator1	$p1 = arg\min_{p \in Predator} dist(z,p)$
predator2	$arg\min_{p \in (Predator - p1)} dist(z,p)$

5.4 Exp3 - Changes in Transition Functions

Here we change the state transition function across tasks. While the source task and reward functions are the same as in Section 5.2, the action effects are depicted in Figure 5.

Although different, the source and target tasks are still related because the agent moves towards the intended direction, with an additional collateral effect of moving in some direction in another axis. Results shown in Figure 6 depict that; in this case, *QManualMapping* presented no advantages over *Regular learning*. Until learning step 60, the performance of *QManualMapping* was not significantly different from *Regular learning*, and after that $QManualMapping$ was significantly worse than all other algorithms. *QBias* is significantly better than *Regular learning* in jumpstart, but *QAverage* is not, and they are consistently better after 1000 learning episodes, which shows that the TL initialization provided a better exploration than learning from scratch. *QBias* and *QAverage* have similar performances during the learning process and are not significantly different in asymptotic performance.

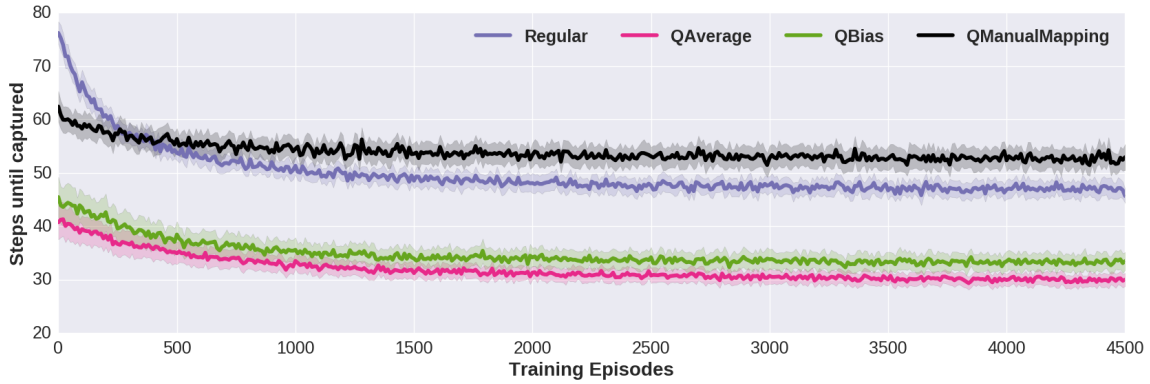

Figure 3: Average performance in 50 repetitions of Experiment 1. The shaded area corresponds to the 99% confidence interval.

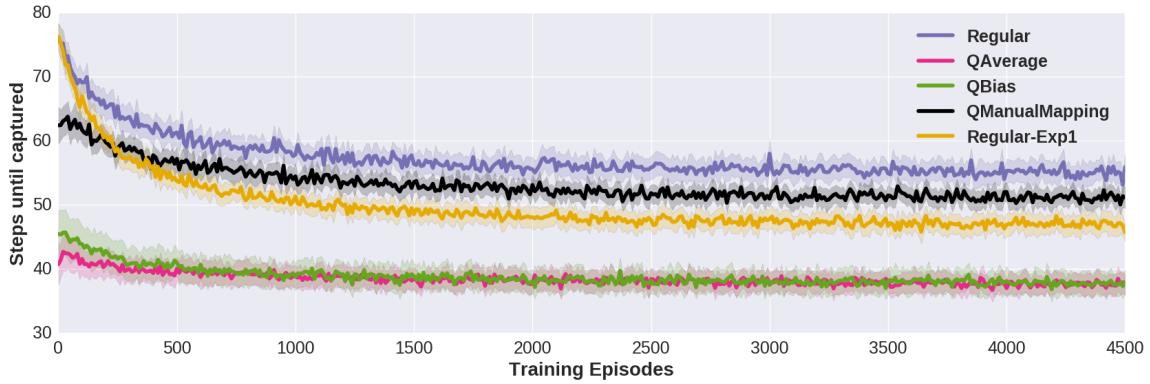

Figure 4: The average performance in 50 repetitions of Experiment 2. The shaded area corresponds to the 99% confidence interval. Regular learning in Experiment 1 was included to allow comparisons in regard to reward function effectiveness.

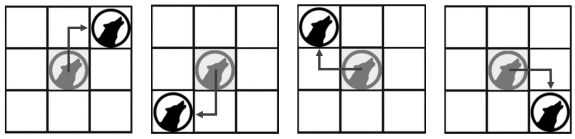

Figure 5: Predator movement after using actions *North* (a), *South* (b), *West* (c), and *East* (d) in Experiment 3.

5.5 Exp4 - TL Across Tasks

The last experiment accomplishes TL through different yet similar domains. Here we take as source task the *Goldmine* task described in Section 5.1. The target task is defined as in Section 5.2. The Class Mapping is defined as $\mathcal{X}_O(Prey) = Gold$, $\mathcal{X}_O(Predator) = Miner$. *QManualMapping* maps the closest predators to the closest miners, and the closest preys to the closest gold pieces, similarly as in Table 1. Comparing Figures 2a and 2b, we can see that the *Goldmine* task is simpler to solve and thus being able to reuse its solution in another (more complex) task is very interesting.

Figure 7 shows that using a simpler version of the target task presents very good results for TL. Again, all TL algorithms presented benefits in regard to jumpstart, showing that simply following a prey is a very effective strategy. *QBias* and *QAverage* present better results in asymptotic performance, showing that the initialization provided by TL greatly improved the learned policy. *QAverage* losses some performance when trying to adapt in the new task, however the final performance is still far better than *Regular learning* and *QManualMapping*. *QManualMapping* and *Regular learning* have equivalent performances after 2000 learning steps. *QBias* has the best performance in this experiment, as the difference between *QBias* and *QAverage* is statistically significant from 1300 learning episodes until the end of the training.

Table 2 summarizes the results of each algorithm in all experiments. Notice that PITAM approaches (QBias and QAverage) were never worse than *Regular learning* and Q-value reuse with a hand-coded Inter-Task Mapping. The results discussed in this paper show that PITAM is a promising method, and can be a determinant step towards building autonomous agents capable of reusing knowledge across tasks.

6. CONCLUSION AND FURTHER WORKS

Being able to find correspondences between tasks is a key issue to successfully transfer knowledge across tasks. We here rely on object-oriented task descriptions to autonomously define Probabilistic Inter-TAsk Mappings (PITAM), which estimates how similar two given states in the target and source tasks are. We also proposed two methods to reuse Q-table estimates across tasks using the learned

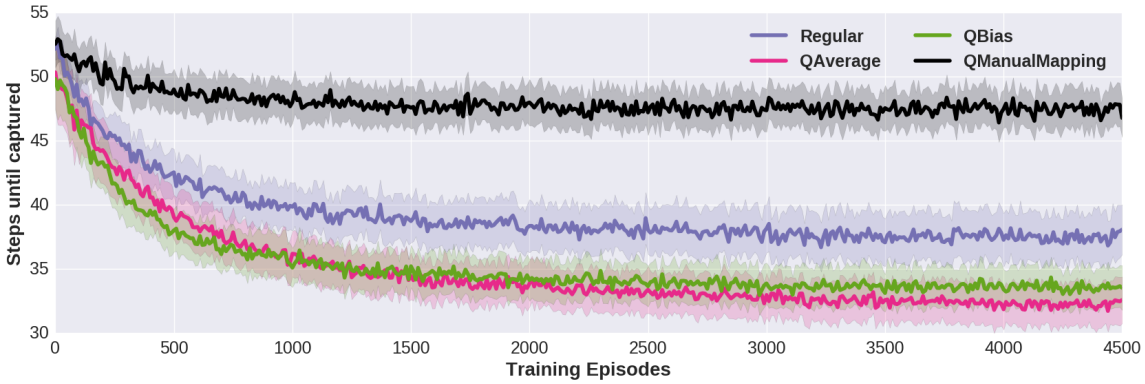

Figure 6: Average performance in 75 repetitions of Experiment 3. The shaded area corresponds to the 99% confidence interval.

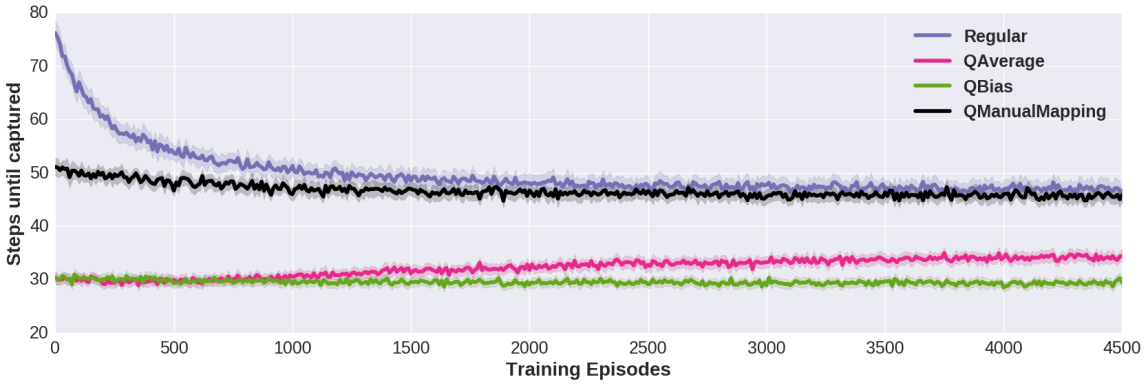

Figure 7: Average performance in 50 repetitions of Experiment 4. The shaded area corresponds to the 99% confidence interval.

Table 2: Summary of all experimental results. Values here represented are the average number of steps to solve the task in each experiment, where *jumpstart* is the performance after 0 learning episodes and *asymptotic* is the performance after 4500 episodes.Best results (according to statistical significance tasks) are denoted in bold.

		Regular	QManualMapping	QAverage	QBias
Experiment 1	jumpstart	76.21	62.43	**40.75**	45.48
	asymptotic	45.67	53.09	**30.03**	33.50
Experiment 2	jumpstart	76.21	62.43	**40.75**	45.48
	asymptotic	55.97	52.34	**37.48**	**38.04**
Experiment 3	jumpstart	52.18	52.70	50.35	**49.83**
	asymptotic	38.08	47.68	**32.59**	**33.54**
Experiment 4	jumpstart	76.21	51.22	**30.34**	**30.48**
	asymptotic	45.67	45.92	34.48	**29.48**

PITAM. Our preliminary experiments included evaluations when transferring knowledge between tasks with different state spaces, reward functions, transition functions, and different domains. When compared to regular learning and *Q-value Reuse* with a hand-coded Inter-Task Mapping, our proposal presented benefits in all evaluated scenarios, even though less domain knowledge is needed.

Further works will focus in evaluating the benefits of our proposal in more complex tasks, such as the *Keepaway* and *Half Field Offense* Robot Soccer Tasks [8, 24]. Furthermore, defining an algorithm to transfer Value Function Approximators [7] across tasks is the next step towards making PITAM more scalable. The excellent results observed when transferring knowledge from a much simpler task suggest that the learning process can be greatly benefited from our proposal if the learner is able to organize the tasks to be solved in order of progressive difficulty. Therefore, PITAM may be greatly improved by combining it with *Curriculum Learning* approaches [2, 26]. Finally, PITAM can be combined with other paradigms of TL, such as in the integrated TL Framework proposed in [20, 21].

Acknowledgments

We gratefully acknowledge financial support from São Paulo Research Foundation (FAPESP), grants 2015/16310-4 and 2016/21047-3, and CNPq (grant 311608/2014-0).

REFERENCES

[1] O. Amir, E. Kamar, A. Kolobov, and B. Grosz. Interactive Teaching Strategies for Agent Training. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 804–811, 2016.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48, 2009.

[3] R. A. Bianchi, L. A. C. Jr., P. E. Santos, J. P. Matsuura, and R. L. de Mantaras. Transferring Knowledge as Heuristics in Reinforcement Learning: A Case-Based Approach. *Artificial Intelligence*, 226:102 – 121, 2015.

[4] G. Boutsioukis, I. Partalas, and I. Vlahavas. Transfer Learning in Multi-agent Reinforcement Learning Domains. In *Proceedings of the 9th European Workshop on Reinforcement Learning*, 2011.

[5] C. Diuk, A. Cohen, and M. L. Littman. An Object-oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 240–247, 2008.

[6] F. Fernández and M. Veloso. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 720–727, 2006.

[7] M. Geist and O. Pietquin. Algorithmic Survey of Parametric Value Function Approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):845–867, 2013.

[8] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone. Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.

[9] D. Isele, M. Rostami, and E. Eaton. Using Task Features for Zero-Shot Knowledge Transfer in Lifelong Learning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1620–1626, 2016.

[10] M. L. Koga, V. F. da Silva, and A. H. R. Costa. Stochastic Abstract Policies: Generalizing Knowledge to Improve Reinforcement Learning. *IEEE Transactions on Cybernetics*, 45(1):77–88, 2015.

[11] M. L. Koga, V. F. d. Silva, F. G. Cozman, and A. H. R. Costa. Speeding-up Reinforcement Learning Through Abstraction and Transfer Learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pages 119–126, 2013.

[12] A. Lazaric. *Transfer in Reinforcement Learning: A Framework and a Survey*, pages 143–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[13] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of Samples in Batch Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 544–551, 2008.

[14] M. L. Littman. Reinforcement Learning Improves Behaviour from Evaluative Feedback. *Nature*, 521(7553):445–451, 2015.

[15] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[17] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.

[18] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[19] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. J. Wiley & Sons, Hoboken (N. J.), 2005.

[20] F. L. d. Silva and A. H. R. Costa. Transfer Learning for Multiagent Reinforcement Learning Systems. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3982–3983, 2016.

[21] F. L. d. Silva and A. H. R. Costa. Accelerating Multiagent Reinforcement Learning through Transfer Learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 5034–5035, 2017.

[22] F. L. d. Silva, R. Glatt, and A. H. R. Costa. Object-Oriented Reinforcement Learning in Cooperative Multiagent Domains. In *Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 19–24, 2016.

[23] J. Sinapov, S. Narvekar, M. Leonetti, and P. Stone. Learning Inter-Task Transferability in the Absence of Target Task Samples. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 725–733, 2015.

[24] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[25] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[26] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone. Automatic Curriculum Graph Generation for Reinforcement Learning Agents. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, pages 2590–2596, San Francisco, CA, February 2017.

[27] M. Tan. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the 10th International Conference on Machine Learning (ICML)*, pages 330–337, 1993.

[28] A. Taylor, I. Dusparic, E. Galvan-Lopez, S. Clarke, and V. Cahill. Accelerating Learning in Multi-Objective Systems through Transfer Learning. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2298–2305, July 2014.

[29] M. E. Taylor, N. Carboni, A. Fachantidis, I. P. Vlahavas, and L. Torrey. Reinforcement Learning

Agents Providing Advice in Complex Video Games. *Connection Science*, 26(1):45–63, 2014.

[30] M. E. Taylor, N. K. Jong, and P. Stone. Transferring Instances for Model-Based Reinforcement Learning. In *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Artificial Intelligence*, pages 488–505, September 2008.

[31] M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous Transfer for Reinforcement Learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2008.

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

[33] M. E. Taylor, P. Stone, and Y. Liu. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[34] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2007.

[35] G. Tesauro. Temporal Difference Learning and TD-Gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995.

[36] N. Topin, N. Haltmeyer, S. Squire, J. Winder, M. desJardins, and J. MacGlashan. Portable Option Discovery for Automated Learning Transfer in Object-Oriented Markov Decision Processes. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3856–3864, 2015.

[37] L. Torrey and M. E. Taylor. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In *Proceedings of 12th the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1053–1060, 2013.

[38] L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using Advice to Transfer Knowledge Acquired in one Reinforcement Learning Task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECAI)*, pages 412–424, 2005.

[39] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[40] Y. Zhan, H. Bou-Ammar, and M. E. Taylor. Theoretically-Grounded Policy Advice from Multiple Teachers in Reinforcement Learning Settings with Applications to Negative Transfer. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2315–2321, 2016.