Neurocomputing 107 (2013) 23–32

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Transferring task models in Reinforcement Learning agents

Anestis Fachantidis [a],*, Ioannis Partalas [b], Grigorios Tsoumakas [a], Ioannis Vlahavas [a]

[a] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece
[b] Laboratoire LIG, Université Joseph Fourier, 38041 Grenoble Cedex 9, France

ARTICLE INFO

Available online 2 November 2012

Keywords:
Reinforcement Learning
Transfer learning
Model-based Reinforcement Learning

ABSTRACT

The main objective of transfer learning is to reuse knowledge acquired in a previous learned task, in order to enhance the learning procedure in a new and more complex task. Transfer learning comprises a suitable solution for speeding up the learning procedure in Reinforcement Learning tasks. This work proposes a novel method for transferring models to Reinforcement Learning agents. The models of the transition and reward functions of a source task, will be transferred to a relevant but different target task. The learning algorithm of the target task's agent takes a hybrid approach, implementing both model-free and model-based learning, in order to fully exploit the presence of a source task model. Moreover, a novel method is proposed for transferring models of potential-based reward shaping functions. The empirical evaluation, of the proposed approaches, demonstrated significant results and performance improvements in the 3D Mountain Car and Server Job Scheduling tasks, by successfully using the models generated from their corresponding source tasks.

1. Introduction

In Reinforcement Learning (RL), two of the most important approaches to improve data efficiency and speed up the learning process are *transfer learning* and *Model-based RL algorithms*. Whereas direct-RL algorithms compute for long periods before finding good policies, model-based algorithms learn an explicit model of the environment and at the same time, the value function which determines the agent's policy. The model generated from this learning process can either be used for generating simulated experiences or planning.

Transfer learning (TL) refers to the process of using knowledge that has been acquired in a previous learned task, the *source task*, in order to enhance the learning procedure in a new and more complex task, the *target task*. The more similar these two tasks are, the easier it is to transfer knowledge between them.

These data efficient approaches are especially important in real world applications of Reinforcement Learning such as robotics, where extensive interaction with the environment needs time and can be costly. Novel ways to combine transfer learning and model-based learning should be considered.

This work proposes a new RL method dubbed ***T**ransfer learning **i**n **M**odel-based **R**einforcement **L**earning **A**gents* (TiMRLA). In principle, TiMRLA is a transfer learning framerwork for an agent implementing hybrid learning. A TiMRLA agent learns both from its interaction with the environment and also from mental experiences generated by a model of the environment's dynamics (model-based learning). Both of its learning mechanisms are able to exploit knowledge gathered from a related source task. TiMRLA's model-based learning, can use a transferred model of the full source task's Markov Decision Process (MDP) to assist learning in the target task. TiMRLA is able to choose whether it will use a target or a source task model. Then, it generates mental experiences using the chosen model and learns directly from them.

TiMRLA's direct learning part (model-free) can also use transferred knowledge from a source task by encoding it in the agent's reward signal. Concretely, the transferred model of a source task is used to shape the reward the agent receives in the target task, efficiently assisting learning.

Specifically, the key insight of our proposed approach is to achieve maximum data efficiency and knowledge exploitation by combining transfer learning with a hybrid learning mechanism (model-based and model-free). This paper extends our previous work [4] with two new methods for transferring models of shaped reward functions and a more sophisticated method for assisting direct learning using model-based shaping. These methods demonstrate new possible ways to further exploit a transferred model. Moreover, new experiments are conducted in two new domains, the Low-Power Mountain Car 3D and the Server Job Scheduling. Finally, this work also offers new insights and discussion on the main-proposed method.

The main contributions of this paper are:

- Learning and transferring the model of a source task to a relative target task. A hybrid, Dyna-like [19] approach is proposed for the target task's learning algorithm implementing a revised version of the TiMRLA algorithm.

* Corresponding author. Tel.: +30 6973 218649.
E-mail addresses: afa@csd.auth.gr (A. Fachantidis), ioannis.partalas@imag.fr (I. Partalas), greg@csd.auth.gr (G. Tsoumakas), vlahavas@csd.auth.gr (I. Vlahavas).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.08.039

- A novel method for transferring shaping rewards, that is, the transfer of a, potential-based shaping function that was successfully used to shape the rewards in a source task (see Section 4.3).
- A novel method for model-based assisting of the direct-learning process is also proposed and discussed (see Section 4.4).
- The empirical evaluation of these approaches, demonstrates significant results and performance improvements in the 3D Mountain Car (3D MC) task, the Low-Power 3D Mountain Car and in Server Job Scheduling, by successfully using the models generated from their corresponding source tasks (see Section 7).

The remainder of this paper is organized as follows. In Section 2 the background knowledge needed for the proposed methods is presented. Section 3 discuses related work. Section 4 describes the proposed methods. Section 5 describes the RL domains used for the experiments and Section 6 the experimental setup. Section 7 present and discuss the results and finally, Section 8 concludes this work.

2. Background

2.1. Model-based Reinforcement Learning

Reinforcement Learning methods that rely on temporal-difference updates, such as Q-Learning or Sarsa, impose certain conditions for convergence to the optimal policy π^*. One such condition is that each state–action pair $\langle s,a \rangle$ should be visited an infinite amount of times [19]. These conditions cannot be easily met in complex or continuous environments. Moreover, these direct-learning algorithms are heavily based on the exploration of the state space, which is not always feasible and can be costly in real-world domains.

On the contrary, model-based approaches try to learn a model of the environment by decomposing it to its main parts, the transition and reward functions T and R, respectively. By learning a model of the underlying dynamics of an environment, model-based RL algorithms can produce simulated experiences for direct learning (e.g. Q-Learning), or use these models with dynamic programming and planning, in order to construct a policy [19]. Doing so they are able to learn a policy using significantly less interaction with the environment (experiences). With less the need for interaction with the environment, model-based learning is an approach towards better data efficiency. Such an approach also minimizes the amount of exploration needed.

To generate models of the transition and reward functions of an environment, model-based RL algorithms can use a variety of supervised learning techniques, such as Artificial Neural Networks (ANN) [9]. Well known model-based RL algorithms are Fitted R-max [8] and PEGASUS [13]. These algorithms have demonstrated the ability to learn near-optimal policies given a small amount of interaction with the environment.

2.2. Transfer learning

Transfer learning refers to the process of using knowledge that has been acquired in a previously learned task, the *source task*, in order to enhance the learning procedure in a new and more complex task, the *target task*. The more similar these two tasks are, the easier it is to transfer knowledge between them. By similarity, we mean the similarity of their underlying Markov Decision Process (MDP) that is, the transition and reward functions of the two tasks as also their state and action spaces.

The type of knowledge that can be transferred between tasks varies among different TL methods. It can be value functions, entire policies as also a set of samples from a source task used from a batch RL algorithm in a target task. This work proposes the transfer of full task models, that is, models of the transition and reward functions of a source task.

In order to enable TL across tasks with different state variables and action sets (i.e. different representations of the state and action spaces), one must define how these tasks are related to each other. One way to represent this relation is to use a pair $\langle X_S, X_A \rangle$ of *inter-task mappings* [25], where the function X_S maps a target state variable to a source state variable and function X_A maps an action in the target task to an action in the source task.

Inter-task mappings have been used for transferring knowledge in several settings like TD-learning [25], policy search and model-based algorithms [26,21]. In all these approaches the inter-task mappings, are provided by a human expert. Moreover, for RL domains where human intuition or knowledge cannot be applied, there are approaches for learning these inter-task mappings autonomously [21].

2.3. Reward shaping

In RL domains, the reward is the only evaluative feedback the agent receives. The main idea of reward shaping (RS) methodologies is to provide an additional reward to the agent, based on some prior knowledge about the task, aiming this way to improve learning performance [7]. Essentially, this *shaping reward* biases certain state–action values providing a more informative reward signal to the agent, assisting him on solving its task faster.

The concept of reward shaping can be represented by the following formula for the Q-learning algorithm:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + F(s_t,a_t,s_{t+1}) + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)),$$

where $F(s_t,a_t,s_{t+1})$ (or by simpler notation, $F(s,a,s')$) is the general form of a shaping rewards function.

The proper selection of this function depends on the task, the prior knowledge we have about it and a deeper understanding of how the agent's goal can be achieved. If such a selection can be done, reward shaping can significantly improve the learning performance [6]. Improperly implemented, reward shaping, which is directly incorporated in the value update process, can also harm the convergence of a learning algorithm, modifying the optimum policy.

Towards this direction, of a reward shaping that could achieve policy invariance, major improvements have been made with the introduction of the *potential-based reward shaping* [14].

Consider a Markov Decision Process (S,A,R,T,γ),[1] its optimal policy π^* and the same MDP but with reward shaping (S,A,R',T,γ) where $R' = R + F$. In [14] Ng et al. defined formal requirements for the form of a shaping rewards function F that guarantees the convergence of the augmented MDP to the same optimal policy π^*. The form of this shaping function in its general form is

$$F(s,a,s') = \gamma \Phi(s') - \Phi(s), \tag{1}$$

where $\Phi(s)$ is a potential function expressing a goal-oriented heuristic for the task. Defining and using such potential-based reward functions also guarantees a good learning behavior, meaning agents that are not mislead to learn sub-optimal policies. Finally, it is important to note that in the recent literature there exist methodologies for learning the shaping rewards such as autonomous shaping [10].

[1] Explaining the notation, S represents the state space, A the set of actions available, R is the reward function, T the transition function of the task and γ the discount factor of the return.

3. Related work

This section presents related work in transfer learning and contrasts it with the proposed approach. For a broader review of transfer learning in RL domains the reader can refer to a comprehensive survey of the field [21].

Fernández and Veloso [5] propose an approach that reuses past policies from solved source tasks to speed up learning in the target task. A restriction of this approach is that the tasks (source and target) must have the same action and state space. Our method allows the state and action spaces to be different between the target and the source task.

Advice based methods have been proposed in [24,27]. Taylor and Stone [24] proposed a method that uses a list of learned rules from the source task as advice, to guide the learning procedure in the target task. The authors introduce three different utilization schemes of the advice. Furthermore, Torrey et al. [27] export rules in first-logic order and translate them into advices.

Taylor et al. [25] proposed a method, named *transfer via inter-task mappings*, that initializes the weights of the target task, using the learned weights of the source task, by utilizing mapping functions. The method depends strongly on the function approximation scheme that is used in both tasks, as the function that transfers the weights is altered according to the approximation method.

The transfer of samples have also been proposed in [22,11]. Lazaric et al. [11] proposed a method that uses samples gathered in the source task, in order to train a batch RL algorithm in the target task. The state and action spaces are not allowed to differ between the two tasks. Taylor et al. [22] present an algorithm for transferring samples in model-based algorithms. The procedure of transfer is accomplished on-line, that is, when the agent interacts with the environment and the state–action space can change.

4. Transfer learning with TiMRLA

The TiMRLA method acquires task models from relative source tasks and implements task model transfer to a direct learning agent of a more complex target task. To make use of the transferred models it adds a model-based learning mechanism in the direct learning agent of the target task, enabling it to use such models. Thus, the transformed direct learning agent can be said to be hybrid as it learns both using e.g. TD-learning from its interactions with the environment and from a model (see Fig. 1). It is important to note here that with minor modifications, the proposed method could equally apply to plain model-based learners but the scope of this work is to demonstrate many effective usages of a model, including the assistance of direct learning. Concretely, this hybrid-learning approach is followed in order to maximize the utility of a source task model.

4.1. Learning the model of a source task

In order to capture the dynamics of a source task environment we have to learn models of the transition and reward functions of its underlying MDP. Our main concern here is not the agent's learning performance in the source task but a balanced exploration of the state–action space in order to achieve a good training set for the task's model.

In order to generate the source task model, the agent is gathering experiences and uses the model-based RL algorithm *model based policy improvement* (MBPI) [9]. MBPI is a model-based learning algorithm that can learn, on-line, the model of a task from real, gathered, experiences. Thereafter, MBPI uses that

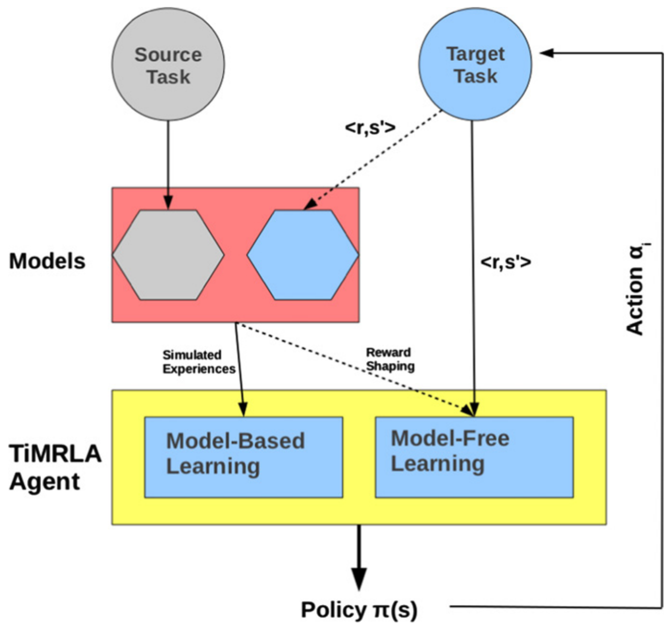


Fig. 1. Schematic representation of the TiMRLA algorithm. The dotted arrows represent optional functions of the TiMRLA algorithm.

model to generate simulated experiences and learns directly from them using Sarsa.

Our specific choice of the MBPI model-based learning algorithm in the source task should not be considered as a limitation of the method as every other model-based RL algorithm can be equally considered, as long as it can provide us with models of the reward and transition functions of the task. The gathered experiences, in the form of a tuple $\langle s,a,r,s' \rangle$, form a training set that will be used to learn the transition and reward functions of the task.

In our specific implementation, two cascade neural networks are used to learn the models of these functions. Cascade neural networks were chosen as the characteristics of this type of networks were considered suitable for the RL scheme [17,16].

Providing a brief description of the cascade neural networks (CNN) we state their basic characteristics. CNN's are fully connected neural networks with short-cut connections. Their topology is determined while learning, with new-candidate neurons being trained separately from the network, adding to it only the best of them (in terms of error). These candidate neurons are created as short-cut connected neurons in a new hidden layer, which means that the final CNN will consist of a number of hidden layers with one – short-cut connected – neuron in each (short-cut connections are connections that skip layers and connect to neurons in the later layers) [16]. Then, after each training epoch, new candidate neurons are prepared, continuing in the same way.

Following up with the method's flow, the source task agent starts to learn and progressively creates the requested models. Every *e* episodes MPBI learns new models of the transition and reward functions, based on the gathered experiences so far. We should note here that, the newly generated models are discarded if their validation mean square error (MSE) is more than that of the previously generated model. This way the method keeps only the best models in terms of validation MSE. We chose validation MSE (not train MSE) to avoid over-fitting and also test the ability of the model to predict from unseen data, such as those in the validation data set.

Furthermore, in continuous domains (e.g. Mountain Car), a value function approximation technique is necessary in order to approximate the value of a state, using nearby state values. We choose to use the Cerebral Model Articulation Controller (CMAC) [1], a well-known, tile-coding, function approximator [19] shown to have good performance in our experimental domains.

4.2. Model-based learning using source task models

In a target task, TiMRLA implements model-based learning by generating mental (simulated) experiences from a source task model and directly learning from them using Q-Learning or Sarsa (policy improvement) (see line 25 of Algorithm 1).

This model can also be a new target task model which can be generated while learning in the target task. In this case, the method does not implement transfer learning. It is important to note here that this work emphasizes and empirically evaluates, the exclusive use of a source task model. Although it is out of scope for this work, we also tested and propose a simple method for choosing which model the agent should use at each episode (target or source).

In the beginning of each episode the two models will compete for being the current episode's "Model" (line 6 of Algorithm 2). The choice can be based on their error on predicting the after-states s' from the current agent's state s, this error is calculated as the euclidean distance between a predicted state and the one experienced (line 19 of algorithm 2).

Algorithm 1. Model based policy improvement (MBPI) [9].

```
1:  $Q \leftarrow Q_0$
2:  $D \leftarrow 0$
3:  repeat
4:    for $e$ episodes do
5:      $s \leftarrow StartStateFromEnvironment()$
6:      repeat
7:        $\alpha \leftarrow SelectAction(Q)$
8:        $r \leftarrow rewardFromEnvironment()$
9:        $s' \leftarrow nextStateFromEnvironment$
10:       $D \leftarrow D \cup (s,\alpha,r,s')$
11:     until $s'$ is terminal
12:   end for
13:   $M \leftarrow LearnModel(D)$
14:   for n iterations do
15:     $s \leftarrow randomStartStateFromSimulator()$
16:     $d \leftarrow 0$
17:     repeat
18:       $\alpha \leftarrow SelectActionSimulate(Q,s)$
19:       $r \leftarrow M.PredictReward(s,\alpha)$
20:       $t \leftarrow M.PredictTermination(s,\alpha)$
21:       if $t$ then
22:         $Q(s,\alpha) \leftarrow Q(s,\alpha)+a(r-Q(s,\alpha))$
23:       else
24:         $s' \leftarrow M.PredictNextState(s,\alpha)$
25:         $Q(s,\alpha) \leftarrow Q(s,\alpha)+a(r+\gamma\ max_{\alpha'}\ Q(s',\alpha')-Q(s,\alpha))$
26:         $s \leftarrow s'$
27:       end if
28:       $d \leftarrow d+1$
29:     until $t = true$ or $d = depth$
30:   end for
31: until Q has converged
```

4.3. Transferring shaping rewards

As discussed earlier in this text, reward shaping can be a feasible technique to assist a goal-driven RL agent and improve its learning performance. This part of the proposed method concerns a novel approach for transferring the source task model of a successfully applied, potential-based, reward shaping function. Starting with the source task, a potential-based reward shaping was used of the form

$$F(s,s') = \Phi(s') - \Phi(s),$$

where F is based on the general form of Eq. (1) which in this work is defined as $F: S \times S \rightarrow \mathbb{R}, \gamma = 1$.

Based on prior work [6] we choose to set $\Phi(s) = V(s)$, where $V(s)$ represents the value function. In [6] this has shown to be an effective potential function. TiMRLA which normally learns the original models of the transition and reward functions of the source task, now learns the augmented reward function R', forming a training set of tuples $\langle s,a \rangle$ with their corresponding shaped rewards $r = r + F(s,s')$.

Same as in the previous section, a cascade neural network is used to learn this potential-based reward function but with one major difference than before. Since $\Phi(s) = V(s)$ the $F(s,s')$ function is based on the values of state variables which are constantly changing and are time-dependent (i.e. while learning, R' is changing in contrast with the original R which is stationary).

In order to learn this augmented reward function efficiently, its learning starts only after the agent converges to a policy and no more learning takes place. Then the specific values $V(s)$ are stable and moreover, represent a better policy than that of the early learning phase.

In the target task, the method transfers the model of this augmented reward function and use that to assist both model-based learning and model-free (direct) (see Section 4.4). For model-based learning, the model of R' is used in the same way as in the previous section (see Section 4.2) replacing the model of the original reward function. Then, mental experiences are generated from the MBPI algorithm directly learning from them with Sarsa (or Q-Learning).

4.4. Assisting direct learning using source task models

In the target task, direct-learning is implemented using Sarsa (λ) accompanied with eligibility traces for temporal-credit assignment. Although this is a model-free approach, in our proposed method it is also assisted by a novel model-based mechanism, *model-based shaping*, which assists the model-free learning process following a reward shaping approach. TiMRLA implements reward shaping for the target task agent, providing him additional rewards generated by the transferred models of the source task.

The TiMRLA algorithm can either use a model of the original reward function or a model of an augmented, potential-based reward function. Because the original reward function of a source task can be less informative on the long-term utility of a specific state–action pair (e.g. -1 reward for every state–action pair), we propose and evaluate the use of the augmented reward function model, when implementing model-based assistance of the direct learning process.

Specifically, we set the augmented reward function R'_{source} of the source task, as the target task's potential function $\Phi(s) = R'_{source}(s)$ forming a new augmented and potential-based reward function for the target task, where

$$\begin{aligned} R'_{trg}(s_t,a_t) &= R_{trg}(s_t,a_t) + F_{trg}(s_{t-1},s_t) \\ &= R_{trg}(s_t,a_t) + R'_{src}(s_t,a_t) - R'_{src}(s_{t-1},a_{t-1}) \end{aligned}$$

The equation above could be seen as an "update rule" for the target task's augmented reward function, executing two queries to the reward shaping model of the source task, one for the current state s_t and one for the preceding state s_{t-1}. Concluding with this part of the proposed approach, we could summarize our key insight to the following: "*When a certain, potential-based, reward shaping scheme is successfully used in a source task, a model of its resulting-augmented reward function, could also be successfully used in a relevant target task*".

Algorithm 2. Transferring Models in Reinforcement Learning Agents—TiMRLA.

```
1:  $Q \leftarrow Q_0$ $D \leftarrow 0$
2:  $Model \leftarrow SourceModel$
3:  repeat for each episode
4:    $s \leftarrow StartStateFromEnvironment()$
5:    $\alpha \leftarrow Select\_e\_GreedyAction(Q)$
6:    if $TargetModel.error < SourceModel.Error$ then //optional
7:      $Model \leftarrow TargetModel$
8:    end if
9:  until end
10: repeat for each step
11:   $s' \leftarrow nextStateFromEnvironment$
12:   $r \leftarrow reward$
13:   $r \leftarrow r + SourceModel.PredictReward(s')$ //Model-Based Reward Shaping
14:   $r \leftarrow r - SourceModel.PredictReward(s)$
15:   $\alpha' \leftarrow Select\_e\_GreedyAction()$
16:   $CMAC.Q(s,\alpha) \leftarrow CMAC.Q(s,\alpha) + a(r + \gamma CMAC.Q(s',\alpha') - CMAC.Q(s,\alpha))$
17:   $s'_{trg} \leftarrow TargetModel.PredictState(s,a)$
18:   $s'_{src} \leftarrow SourceModel.PredictState(s,a)$
19:   $SourceModel.Error \leftarrow SourceModel.Error + d(s'_{src},s')^2$
20:   $TargetModel.Error \leftarrow TargetModel.Error + d(s'_{trg},s')^2$ //optional
21:   $s \leftarrow s'; \alpha \leftarrow a'$
22:   if $TrainingPeriodElpased = true$ then
23:     $ImprovePolicy(Q)$ //using MBPI in our case, see Alg. 1
24:     $LearnTargetModel(D)$ //optional
25:   else
26:     $D \leftarrow D \cup (s,\alpha,r,s')$
27:   end if
28: until $s'$ terminal
```

4.5. The TiMRLA algorithm

Learning in the source task provides us with the best, in terms of validation MSE, neural networks representing models of the transition and reward functions (see Section 4.1). These two networks are used by TiMRLA to predict next states and their rewards. The reward function model can either represent the real reward function of the source task or the augmented potential-based one.

The flow of the proposed algorithm (see Algorithm 2), as it executes in the target task, is the following: in lines 1–2 some basic initializations are made. The weights in the CMAC, representing our value function $Q(s,a)$, are all set to zero and the initial model to be used is the source model. In the beginning of each episode based on the the start state an action is selected (ϵ-greedy) with respect to the policy. Next, takes place the (optional) process of choosing the model that will be used (lines 6 and 7) in the specific episode. In lines 11 and 12, for each step,the agent obtains the current state and its reward. Next, model-free learning with Sarsa (λ) takes place, with its update rule using an added "CMAC" notation to emphasize the fact that these values come from a CMAC function approximator.

Next, in lines 17–20 and for each even numbered step (not in every step, for robustness) the two models (target and source) are queried for the new state s' (already known) based on the preceding state–action pair (s,a). Then, the algorithm computes and compares two euclidean distances $d(s',s'_{target})$ and $d(s',s'_{source})$ where the second argument in the distances, represents the state predictions of the two models. The distances are stored in a tabular form with a window of 1000 distances and the two mean errors are calculated for that window. As mentioned previously, at the start of each episode, based on this averaged error (distance), TiMRLA chooses to use the target or the source task model (the one with the less error). Again, these steps are optional and we can choose not to train an original target task model (only transfer source task models).

In the next lines, the flow continues with the MBPI algorithm for policy improvement using generated experiences from the models.

5. Empirical domains

5.1. Generalized Mountain Car

The Mountain Car (MC) domain was originally proposed by [12] and extended by [18]. In the standard 2D task an underpowered car must be driven up to a hill. The state of the environment is described by two continuous variables, the horizontal position $x \in [-1.2,0.6]$, and velocity $v_x \in [-0.007,0.007]$. The actions are {neutral, left and right} which modify the velocity by -0.001,0 and 0.001, respectively. At each time step, an amount of $-0.0025 * \cos 3x$ is added to the velocity, in order to take gravity into account.

Each episode starts with the car at the bottom of the hill and ends when x becomes greater than 0.5. At each time step, the agent selects among the three available actions and receives a reward of -1. The objective is to move the car to the goal state, as fast as possible. The main difficulty in this environment is that the force of gravity is greater than the force that the engine is capable to provide. Therefore, the agent controlling the underpowered car must move back and forth, in order to exploit the effect of gravity to its favor and manage to reach the goal state.

The 3D MC extends the 2D task by adding an extra spatial dimension. The 3D task was originally proposed in [23]. The state is composed by four continuous variables, the coordinates in space x, and $y \in [-1.2,0.6]$, as well as the velocities v_x and $v_y \in [-0.07.0.07]$. The available actions are {neutral, west, east, south, north}. The Neutral action has no impact on the velocity of the car. The west and east actions add -0.001 and 0.001 to v_x, respectively, while south and north add -0.001 and 0.001 to v_y, respectively. Additionally, in order to simulate the effect of gravity, at each time step, $-0.0025 * \cos 3x$ and $-0.0025 * \cos 3y$ is added to v_x and v_y, respectively. Each episode starts with the car at the bottom of the basin. The goal state is reached when $x \geq 0.5$ and $y \geq 0.5$. At each time step, the agent receives a reward of -1.

In this work we use the MC software[2] that is based on version 3.0 of the RL-Glue library[3] [20].

5.2. Low-Power Mountain Car 3D

Low-Power Mountain Car 3D is a variant of the MC 3D domain where the car is even more underpowered. This significantly

[2] Available at http://library.rl-community.org/.
[3] Available at http://glue.rl-community.org/.

increases the difficulty of an agent to find the goal as also, the time needed (steps) to reach it. Specifically, the west and east actions now add -0.0008 and 0.0008 to v_x, respectively, while south and north add -0.0008 and 0.0008 to v_y, respectively. It is important to note that these changes in the dynamics of the environment, essentially alter the transition function of the original task.

5.3. Server job-shop scheduling

In the domain of Server Job Scheduling (SJS) [28] a specified number of jobs wait in a queue of a server to be processed. There are certain types of jobs and each of them has a utility function representing the user's anticipation over time. The goal of a scheduler for this domain is to pick a job (action) based on the status of the queue (state). Its immediate reward in each step is minus the summation of the utilities of the remaining jobs in the queue. Each episode lasts 200 steps and begins with a 100 jobs pre-loaded in the queue. Transfer learning experiments on this domain [28,2] have used two job types for the source tasks (eight state variables) and the standard four job types for the target task (16 state variables). Finally, we should note that SJS is considered a difficult RL domain having large state and action spaces. Also it is a stochastic domain as for the first 100 time-steps, new jobs of random types arrive in the queue.

6. Experiments

6.1. Source task setup

First, after thorough experimentation we concluded to a set of parameters for the MBPI agent in the source task. We set ϵ to 0.5 decaying by a factor of 0.99, the step-size parameter α to 0.2 and γ, the discounting factor, to 1 as this task is an episodic one. Also, depth d of the MBPI algorithm was set to 1 as [9] show that a greater depth decreases the overall learning performance.

As the standard MBPI algorithms specifies, after every e episodes, the ANN's train based on the set of experiences D gathered so far. We set e to 40 episodes. So, every 40 episodes the ANNs produced are compared with the previously trained based on their validation MSE, finally keeping the best of them. We let the agent learn for a total of 1000 episodes.

TiMRLA in the above standard setup, transfers models of the original transition and reward functions of a source task. Instead, when we want to transfer an augmented reward function, that is, the reward function plus a potential-based shaping, a different approach is followed.

Specifically, in the MC 2D task we use reward shaping for a standard Sarsa (λ) agent by defining a potential function $\Phi(s)=V(s)$ and the resulting reward shaping function $F(s,s')=\Phi(s')-\Phi(s)$. Using this reward shaping method, we achieved significantly improved performance in the standard MC 2D task, which made it eligible for transferring through our proposed method.

A Cascade ANN (CNN) was then used to learn the augmented–shaped reward function R'_{source}. In this case the CNN for the reward function is trained only after learning stops, so that a potential function defined as above, becomes stationary. Then, every 40 episodes in a total of 500 episodes, a new model of the reward function is trained according to the method described above.

In our context the CNN's for the reward and transition functions of the source task were trained with a learning rate of 0.07 resulting to two networks both consisting of 10 single-neuron hidden layers, fully connected through short-cut connections (typical topology for CNN, see Section 4.1 and [16,15]). The transition function CNN had three input nodes representing the two state variables and the action, and two output nodes representing the new state (predicted after-state). The reward function CNN had the same input layer but with a single-node output layer representing the predicted reward. For implementation, we used the neural network library *Fast Artificial Neural Network* (FANN) [15].

Concerning the evaluation of these experiments, standard average reward plots were used as also two other metrics presented in [23]: time to threshold and area under the learning curve.

Time to threshold is an evaluation metric expressing the time needed for an agent to reach a specific performance threshold. In our setting time is expressed in agent's steps and threshold in episodes (i.e. episodes needed for an agent to reach the goal in some given steps–threshold, see Table 2). *Area under the learning curve* is another metric expressing the ratio r of the areas defined by two learning curves. This metric is based on the the fact that a learning algorithm accruing more reward than another (e.g. transfer vs. no-transfer) has also a greater area under its curve (see Table 3).

In the SJS domain, an agent learned the model of an SJS task whith two types of jobs. The two-job source task agent learned for 3000 episodes while recording its experiences to form the CNN's training set. Two CNN's were trained, one for the transition function and one for the reward function of the task. The transition function CNN had nine input nodes representing the eight state variables and the action, and eight output nodes representing the new state (predicted after-state). The reward function CNN had the same input layer but with a single-node output layer representing the predicted reward.

6.2. Transferring in Mountain Car 3D

First, as our methodology allows for different state and action variables between the source and the target task, inter-task mappings are set. For our experimental domain this is successfully done in [22]. Manual mapping of the different actions and states should not be seen as a limitation of the method, as there exist methods that are able to automatically learn the appropriate mappings [23,3]. Table 1 shows the inter-task mapping used in this experiments.

In the target task, for the model-based learning part of the algorithm, the best results were obtained by setting the model-based policy improvement period to 1, which means that policy improvement would take place in every episode. The iterations n were set to 2000, and ϵ and the step-size parameter α, were set to 0 (greedy) and 0.07, respectively, (see Algorithm 1). This parameter set is the one that gave us the best results. A discussion on the importance and relation of these parameters to the final agent's performance, can be found in Section 7.

In MC 3D the versions of the TiMRLA agent compared were:

- A standard TiMRLA agent which transfers and uses models of the original MC 2D transition and reward functions.
- A TiMRLA-augmented reward function agent, using a model of the original MC 2D transition function and a model of its augmented, potential-based, reward function.

Table 1
The inter-task mapping used by TiMRLA in Mountain Car to generate target task instances from the source task model.

Actions		State variables	
3D action	2D action	3D variable	2D variable
Neutral	Neutral	$\mathbf{x}$	$\mathbf{x}$
North	Right	$\mathbf{v_x}$	$\mathbf{v_x}$
East	Right	$\mathbf{y}$	$\mathbf{x}$
South	Left	$\mathbf{v_y}$	$\mathbf{v_x}$
West	Left		

- A TiMRLA-augmented reward function + direct learning assist agent, which also assists its direct learning process by implementing model-based reward shaping, based on the model of the augmented reward function.
- A no-transfer case, for which a standard Sarsa (λ) agent is used with CMAC function approximation and eligibility traces.

6.3. Transferring in Low-Power Mountain Car 3D

For Low-Power MC 3D the same settings apply as in the previous section. Here, a standard TiMRLA agent is compared to no-transfer but for standard TiMRLA we also experiment with another case, a standard TiMRLA agent with a decaying learning rate for its simulated experiences.

As it is shown in the next section (see Section 7) the model-based learning rate α of the MBPI algorithm (see line 25, Algorithm 1) strongly correlates with the agent's performance when we transfer models to less relevant tasks, such as Low-Power MC 3D. So, we initially set α to 0.07 (as before) but set its decaying to 0.99 per episode. Summarizing, in Low-Power MC 3D the versions of the TiMRLA agent compared were:

- A standard TiMRLA agent set as in Section 6.2 with its model-based learning rate set to 0.07.
- A standard TiMRLA agent but with a decaying model-based learning rate.
- A no-transfer case, where a standard Sarsa (λ) agent is used with CMAC function approximation and eligibility traces.

6.4. Transferring in server job scheduling

In this experiment a standard TiMRLA agent transferred a model of the two job SJS task to a four job SJS task. The TiMRLA agent is compared to a standard Sarsa (λ) agent with a CMAC function approximator without transfer learning. For TiMRLA, to enable transfer between the two SJS task we first set the necessary intertask mapping and used the one described in [2] which maps the state variables from job type 1 to job types 1 and 2 and job type 2 to job types 3 and 4. The same applies for the action mappings.

For the parameters of the target task, after an experimentation, the following values were selected. The model-based policy improvement period was set to 1, the iterations n were set to 100, and ϵ and the step-size parameter α, were set to 0 (greedy) and 0.01 (decaying by 0.999 per episode), respectively. To allow a clear comparison between them and account for the effect of transfer learning, the previous parameters were also chosen for the Sarsa agent. Moreover, TiMRLA's direct learning part used the Sarsa learning update and the same CMAC function approximator (Fig. 2).

7. Results and discussion

7.1. Mountain car 3D

For MC 3D, Fig. 3 presents the average cumulative reward that is obtained by each algorithm over the learning episodes. The graphs were produced from 20 independent trials per method and with a smoothing–averaging window of 20 episodes, for clarity purposes. They show a clear performance gain of the proposed algorithm TiMRLA over the standard model-free agent, which is also statistically significant at $p < 0.05$ (one-tail t-test). Its performance demonstrates the positive effect of transferring a model from a source task and moreover that this model can still assist learning even after many episodes.

Next, TiMRLA using a potential-based reward model demonstrates even better performance than transferring a model of the original reward function of the source task. This confirms our initial hypotheses that a potential-based reward shaping scheme which shows good results in a source task, can also improve the performance in a relevant target task.

Concerning the simulated experiences, it is important to note that the best results were obtained with policy improvement taking place in every episode. This shows us that the models succeeded in producing real-like experiences that were positively influencing the learning algorithm just like complementary steps.

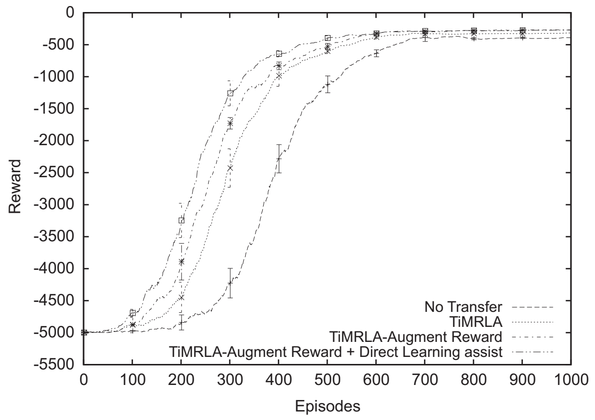


Fig. 3. Average reward in a period of 1000 episodes in MC 3D. The curves are smoothed with a window of 20 episodes with standard error bars every 100 episodes.

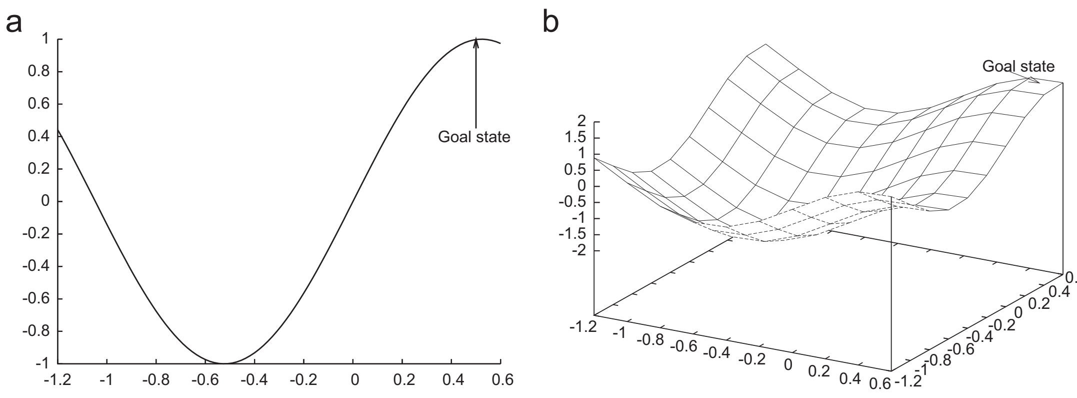


Fig. 2. (a) The Mountain Car 2D task. (b) The Mountain Car 3D task.

Although that, the tuned value for the step-size parameter was 0.07 being less than that used for the real experiences which was set to 0.2. This can be seen as a reasonable consequence of using approximate models since they are error prone and can produce at times, errors that could negatively alter the learning procedure.

Another interesting finding is that the best results were obtained with the simulated actions ϵ parameter set to zero (see Algorithm 1, line 18). We hypothesize that the model-based learning part, favors greedy action selection because the agent explores anyway by just being placed at random start states. This sub-optimal placing is the exploration mechanism for the agent's mental experiences, and he balances by implementing greedy action selection.

Table 2 shows the results of the time-to-threshold evaluation metric in MC 3D, comparing standard TiMRLA with the no-transfer case. Four different performance thresholds were set concerning the steps needed from the agent to reach the goal. The greatest improvement for the proposed approach can be seen in the first threshold (4000 steps) which a TiMRLA agent is able to achieve roughly after 227 episodes in contrast to the no-transfer case where the agent needed about 323 learning episodes.

For the other thresholds the difference is still significant but degrading, showing that the positive effect of transferring source task models is especially important in the first learning episodes where the agent had less interaction with the environment.

Table 3 shows the results of the area under the curve evaluation metric in MC 3D. Here, all the proposed TiMRLA methods were compared with the no-transfer case by calculating and comparing the ratio of the areas under their learning curves. TiMRLA using both augmented reward function model (pot. function) and direct learning assist, demonstrated the best performance. Its learning curve covers 28.5% more area than that of the no-transfer case showing that the parallel use of such a reward model for simulated experience and for reward shaping can show significant improvements (Table 4).

It is important to note that TiMRLA in this fully deployed version, has better performance than TiMRLA using just an augmented reward model but without direct learning assist. A first conclusion is that direct learning assist significantly improves the learning performance. One more insight for this can be that both the simulated experiences rewards and the shaped rewards come from the same rewards distribution where in the other case of TiMRLA, the agent is directly experiencing the original task rewards but in its model-based part the augmented ones, probably creating a form of incompatibility.

Table 2
Time-to-threshold evaluation metric for four different step thresholds in Mountain Car 3D, comparison of standard TiMRLA to No-Transfer. The results average 20 independent trials of each method.

Threshold (steps)	Time-to-threshold (episodes)				Avg. improvement (%)
	No-Transfer	St. dev.	TiMRLA	St. dev.	
4000	323	63.16	227	50.12	29.7
2000	425	55.37	320	41.83	24.7
1000	528	61.66	398	54.16	24.6
400	684	65.73	592	59.34	13.4

Table 3
Average area under the learning curve ratios in MC 3D from 20 independent trials. All versions of TiMRLA compared to No-Transfer.

	Area under the curve	Improvement (ratio) (%)
No-Transfer	2788.68	–
TiMRLA	3257.72	16.8
TiMRLA-Augm. reward	3428.76	22.9
TiMRLA-Augm. reward + direct assist	3584.97	28.5

7.2. Low-Power Mountain Car 3D

For Low-Power MC 3D, Fig. 4 shows the average cumulative reward obtained by each algorithm over the learning period. episodes. The proposed method demonstrated a stable stat. significant (at $p<0.05$) when using a decaying learning rate for the simulated experiences. This result shows the positive effect of transferring a task model, even to less relevant target tasks.

An interesting finding here is that when a constant learning rate is used in low-power MC 3D (standard TiMRLA), although we have an initial performance boost, the agent finally converges to a sub-optimal policy (see Fig. 3). One possible explanation is based on the task differences. Low-power MC 3D has different dynamics and transition function. Thus, it can be seen as a less relevant task for

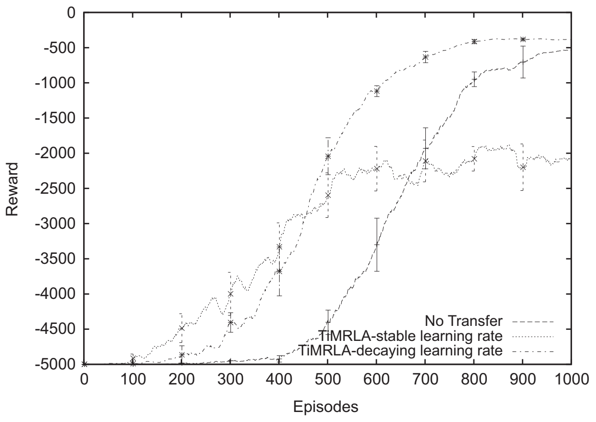


Fig. 4. Average reward in a period of 1000 episodes in low-power MC 3D. The curves are smoothed with a window of 20 episodes with standard error bars every 100 episodes.

Table 4
Time-to-threshold evaluation metric for four different step thresholds in Low-Power Mountain Car 3D, comparison of No-Transfer to TiMRLA with decaying learning rate and standard TiMRLA. The hyphen (-) denotes thresholds that were not achieved in a duration of 1000 episodes. The results average 20 independent trials of each method.

Threshold (steps)	Time-to-threshold (episodes)							
	No Transfer	St. dev.	TiMRLA	St. dev.	Improv. (%)	TiMRLA decay	St. dev.	Improv. (%)
4000	541	62.39	300	61.54	44.5	368	53.67	31.9
2000	696	60.48	804	62.96	− 15.5	506	49.76	27.3
1000	796	57.36	–	–	–	615	50.27	22.7
400	–	–	–	–	–	820	52.83	–

the standard 2D MC models. Using these models initially improves performance but in the long term, their constant learning rate (contribution) in the agents value function, is harmful.

One solution given is using a decaying learning rate for the simulated experiences. This approach demonstrated the significant improvement over the no-transfer case (see Fig. 4). A limitation of this approach is that requires some parameter tuning as to achieve a similar overall performance improvement. One has to set both the initial learning rate for the simulated experiences, as also its decaying factor.

The experiment results are also confirmed in Table 2 using the time-to-threshold evaluation metric. TiMRLA and no transfer cannot achieve the smaller thresholds whereas TiMRLA whith a decaying learning rate, is able to achieve all threshold values in 1000 episodes.

Based on the above, we can conclude that when transferring models from tasks with different reward and/or transition functions, the learning rate for the simulated experiences plays a crucial role for the learning performance and should be better set to decay.

Finally, the mean predictive accuracy of the MC 2D Transition model was also calculated. The mean euclidean distance of a real after-state s' to the predicted one was 0.018 for the state variables x and y.

7.3. Server job-shop scheduling

Fig. 5 depicts the average cumulative reward obtained by TiMRLA and Sarsa (λ) over a learning period of 5000 episodes. The proposed approach significantly assisted the performance of the agent especially between episodes 1300 and 2250 where a statistically significant difference in performance was found. Note that the SJS is a harder domain since the connection between the source and the target tasks is not as straightforward as in mountain car. Moreover, the SJS target task agent must handle two new-unseen job types. For this reason, a decaying learning rate was chosen when learning from the simulated-mental experiences generated from the CNN's.

As both compared algorithms have the same direct learning part (Sarsa (λ)) we can observe that they converge to the same average reward levels in the last 2000 episodes. Transferring a task model with TiMRLA in this domain, helped on significantly speeding-up learning but not on finding a marginally better policy since both algorithms exploration, function approximation and learning rule mechanisms are the same. Interestingly, the CNN's demonstrated an ability to learn good models of the SJS tasks and while the mental experiences generated by them were more noisy compared to MC3D noise was counter balanced by the CMAC function approximator.

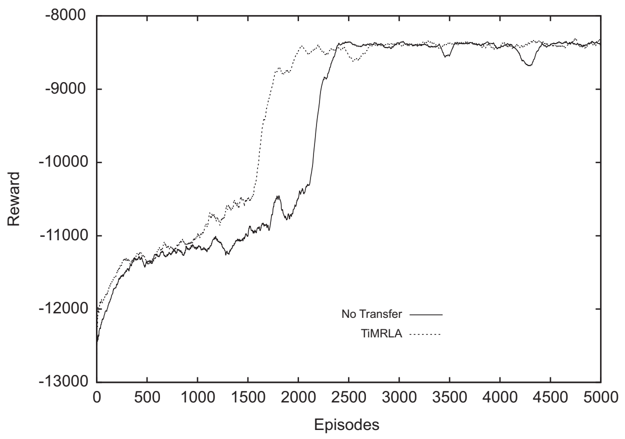


Fig. 5. Average reward in a period of 5000 episodes in SJS. The curves average 12 trials of each algorithm and are smoothed with a window of 100 episodes.

8. Conclusions and future work

This work examined the viability and benefits of transferring the full model of a task to a different but relevant task. The tasks are allowed to have different actions and state variables by constructing and using the right inter-task mappings. The use of cascade neural networks in an RL scheme showed promising results by providing us with accurate enough models of the transition and reward functions of the the source task.

The source's task model was successfully used from a novel algorithm capable of exploiting it. The proposed algorithm is equipped with two learning mechanisms (hybrid) one for model-free learning and one for model-based. Empirical results demonstrate a significant performance improvement when compared to the "no-transfer" case. The model of a source task, generated using supervised learning, can be efficiently used by a standard model-free learning algorithm either through a model-based add-on module (TiMRLA model-based transfer), or directly, with approaches such as that of model-based reward shaping. Moreover, concurrent learning of TD-Learning with TD-learning from simulated experiences, can be compatible and efficient.

Proposals for future work include the evaluation of the method with the use of different model-based algorithms and the further investigation of the importance of the simulated experiences learning rate (contribution) when transferring across tasks with different transition and/or reward function. One possible approach other than a decaying learning rate, could be that of an agent who autonomously adjusts its learning rate according to its performance, convergence progress, etc. Finally, the development of new hybrid-learning algorithms may further demonstrate the advantages of these approaches.

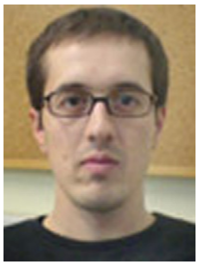
References

[1] J.S. Albus, A new approach to manipulator control: the cerebellar model articulation controller (cmac), J. Dyn. Syst. Meas. Control 97 (1975) 220.
[2] Kyriakos C. Chatzidimitriou, Ioannis Partalas, Pericles A. Mitkas, Ioannis Vlahavas, Transferring evolved reservoir features in reinforcement learning tasks, in: European Workshop on Reinforcement Learning, 2011.
[3] A. Fachantidis, I. Partalas, M.E. Taylor, I. Vlahavas, Transfer learning via multiple inter-task mappings, in: European Workshop on Reinforcement Learning, 2011.
[4] A. Fachantidis, I. Partalas, G. Tsoumakas, I. Vlahavas, Transferring models in hybrid reinforcement learning agents, Eng. Appl. Neural Networks (2011) 162–171.
[5] F. Fernández, M. Veloso, Probabilistic policy reuse in a reinforcement learning agent, in: 5th International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, pp. 720–727.
[6] M. Grzes, D. Kudenko, Online learning of shaping rewards in reinforcement learning, Neural Networks 23 (4) (2010) 541–550.
[7] Vijaykumar Gullapalli, Andrew G. Barto, Shaping as a method for accelerating reinforcement learning, in: Proceedings of the 1992 IEEE International Symposium on Intelligent Control, IEEE, 1992, pp. 554–559.
[8] N. Jong, P. Stone, Model-based exploration in continuous state spaces, in: The 7th Symposium on Abstraction, Reformulation, and Approximation (SARA '07), 2007, pp. 258–272.
[9] Shivaram Kalyanakrishnan, Peter Stone, Yaxin Liu, Model-based reinforcement learning in a complex domain, in: Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, Frank Dellaert (Eds.), RoboCup 2007: Robot Soccer World Cup XI, Lecture Notes in Computer Science, vol. 5001, Springer, Berlin/Heidelberg, 2008, pp. 171–183.
[10] G. Konidaris, A. Barto, Autonomous shaping: knowledge transfer in reinforcement learning, in: Proceedings of the 23rd International Conference on Machine Learning, ACM, 2006, pp. 489–496.
[11] Alessandro Lazaric, Marcello Restelli, Andrea Bonarini, Transfer of samples in batch reinforcement learning, in: ICML '08: Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 544–551.
[12] Andrew Moore, Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces, in: Machine Learning: Proceedings of the 8th International Conference, 1991.
[13] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang, Autonomous inverted helicopter flight via reinforcement learning, Exp. Robotics IX (2006) 363–372.
[14] Andrew Y. Ng, Daishi Harada, Stuart Russell, Policy invariance under reward transformations: theory and application to reward shaping, in: Proceedings of the 16th International Conference on Machine Learning, Morgan Kaufmann, 1999, pp. 278–287.

[15] Steffen Nissen, Implementation of a Fast Artificial Neural Network Library (FANN), Department of Computer Science University of Copenhagen (DIKU), Software available at retrieval, in: ACM 14th Conference on Information 〈http://leenissen.dk/fann/〉, 2003.
[16] Steffen Nissen, Large Scale Reinforcement Learning Using Q-Sarsa (λ) and Cascading Neural Networks, Master's Thesis, Department of Computer Science, University of Copenhagen, October 2007.
[17] François Rivest, Doina Precup, Combining TD-learning with cascade-correlation networks, in: Proceedings of the 20th International Conference on Machine Learning, AAAI Press, 2003, pp. 632–639.
[18] Satinder P. Singh, Richard S. Sutton, Reinforcement learning with replacing eligibility traces, Mach. Learn. 22 (1–3) (1996) 123–158.
[19] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[20] Brian Tanner, Adam White, Rl-glue: language-independent software for reinforcement-learning experiments, J. Mach. Learn. Res. 10 (2010) 2133–2136.
[21] Matthew Taylor, Peter Stone, Transfer learning for reinforcement learning domains: a survey, J. Mach. Learn. Res. 10 (2009) 1633–1685.
[22] Matthew E. Taylor, Nicholas K. Jong, Peter Stone, Transferring instances for model-based reinforcement learning, in: Machine Learning and Knowledge Discovery in Databases, vol. 5212, September 2008, pp. 488–505.
[23] Matthew E. Taylor, Gregory Kuhlmann, Peter Stone, Autonomous transfer for reinforcement learning, in: AAMAS '08: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, 2008, pp. 283–290.
[24] Matthew E. Taylor, Peter Stone, Cross-domain transfer for reinforcement learning, in: ICML '07: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 879–886.
[25] Matthew E. Taylor, Peter Stone, Yaxin Liu, Transfer learning via inter-task mappings for temporal difference learning, J. Mach. Learn. Res. 8 (2007) 2125–2167.
[26] Matthew E. Taylor, Shimon Whiteson, Peter Stone, Transfer via inter-task mappings in policy search reinforcement learning, in: 6th International Joint Conference on Autonomous Agents and Multiagent Systems, 2007, pp. 1–8.
[27] L. Torrey, J. Shavlik, T. Walker, R. Maclin, Skill acquisition via transfer learning and advice taking, in: 17th European Conference on Machine Learning, 2005, pp. 425–436.
[28] Shimon Whiteson, Peter Stone, Evolutionary function approximation for reinforcement learning, J. Mach. Learn. Res. 7 (2006) 877–917.

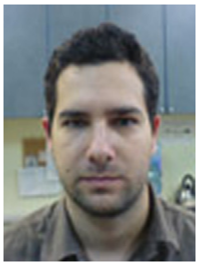
Anestis Fachantidis received a BSc in mathematics from the Aristotle University of Thessaloniki (AUTH), Greece in 2007 and an MSc in information systems from the University of Macedonia (UoM), Greece (2009). He worked as a research associate in the Information Systems Lab in UoM (2009) and in the Centre for Robotics and Neural Systems (CRNS) in the University of Plymouth, UK (2012).

Currently he is a PhD student in the Department of Informatics, AUTH. His research interests include various aspects of machine learning, reinforcement learning and transfer learning as also in their applications to robotics, web services and logistics.



Ioannis Partalas received a BSc in informatics from the University of Ioannina, Greece (April 2004), an MSc in information systems from Aristotle University of Thessaloniki, Greece (September 2006) and a PhD from the same department in 2009. He worked as an adjunct lecturer in University of Western Macedonia and also as a research associate in the Department of Informatics in Aristotle University of Thessaloniki.

Currently he is post-doctoral researcher in the Laboratoire d'Informatique de Grenoble. His main interests include Reinforcement Learning, ensemble methods, large-scale hierarchical classification and web crawling.



Grigorios Tsoumakas is a lecturer at the Department of Informatics of the Aristotle University of Thessaloniki (AUTH). He received a BSc in informatics from AUTH in 1999, an MSc in artificial intelligence from the University of Edinburgh in 2000 and a PhD in informatics from AUTH in 2005.

His research interests include various aspects of machine learning, knowledge discovery and data mining, including ensemble methods, distributed data mining, text classification and multi-label learning.

Ioannis Vlahavas is a professor in the Department of Informatics at the Aristotle University of Thessaloniki, Greece. He received a PhD degree in logic programming systems from the same university in 1988. During the first half of 1997 he was a visiting scholar at the Department of Computer Science at Purdue University. He specializes in logic programming, machine learning, automated planning, knowledge-based and artificial intelligence systems and he has published over 160 papers in scientific journals, conference proceedings and book chapters in international edited volumes and coauthored six books in these areas. He has been involved in more than 25 research and development projects, leading most of them. He was the chairman of the Second Hellenic Conference on AI and the host of the Second International Summer School on AI Planning. He is leading the Programming Languages and Software Engineering Laboratory (PLASE Lab, http://plase.csd.auth.gr/) and the Logic Programming and Intelligent Systems Group (LPIS Group, http://lpis.csd.auth.gr) (more information at http://www.csd.auth.gr/∼vlahavas).