6-2017

Scalable transfer learning in heterogeneous, dynamic environments

Trung Thanh Nguyen
Adobe Systems

Tomi Silander
Xerox Research Centre Europe

Zhuoru LI
National University of Singapore

Tze-Yun LEONG
Singapore Management University, leongty@smu.edu.sg

DOI: https://doi.org/10.1016/j.artint.2015.09.013

Citation

Nguyen, Trung Thanh; Silander, Tomi; LI, Zhuoru; and Tze-Yun LEONG. Scalable transfer learning in heterogeneous, dynamic environments. (2017). *Artificial Intelligence*. 247, 70-94. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/3039

Published in Artificial Intelligence, Volume 247, June 2017, Pages 70-94.
http://doi.org/10.1016/j.artint.2015.09.013

Scalable transfer learning in heterogeneous, dynamic environments

Trung Thanh Nguyen[a,*,1], Tomi Silander[b,1], Zhuoru Li[c], Tze-Yun Leong[d,c]

[a] *Adobe, 345 Park Avenue, San Jose, CA, USA*
[b] *Xerox Research Centre Europe, 6 chemin de Maupertuis, 38240 Meylan, France*
[c] *School of Computing, National University of Singapore, Singapore*
[d] *School of Information Systems, Singapore Management University, Singapore*

A R T I C L E I N F O

Article history:
Received in revised form 17 September 2015
Accepted 29 September 2015
Available online 3 October 2015

Keywords:
Model-based reinforcement learning
Transfer learning
Online feature selection

A B S T R A C T

Reinforcement learning is a plausible theoretical basis for developing self-learning, autonomous agents or robots that can effectively represent the world dynamics and efficiently learn the problem features to perform different tasks in different environments. The computational costs and complexities involved, however, are often prohibitive for real-world applications. This study introduces a scalable methodology to learn and transfer knowledge of the transition (and reward) models for model-based reinforcement learning in a complex world. We propose a variant formulation of Markov decision processes that supports efficient online-learning of the relevant problem features to approximate the world dynamics. We apply the new feature selection and dynamics approximation techniques in heterogeneous transfer learning, where the agent automatically maintains and adapts multiple representations of the world to cope with the different environments it encounters during its lifetime. We prove regret bounds for our approach, and empirically demonstrate its capability to quickly converge to a near optimal policy in both real and simulated environments.

1. Introduction

Next generation robotics aims at developing autonomous agents with integrative intelligence; these agents or groups of agents can automatically or semi-automatically collect sensor inputs, make decisions, learn new skills, and interact with humans or other agents to complete multiple, complex tasks in different real-world domains [1]. Emerging global research and development trends point to collaborative robots that can work and interact with people [2], dexterous robots with advanced manipulation skills and “soft touches” [3], and cognitive robots that are self-improving, robust, and flexible [4]. In Asia, for example, social and economic demands have led to intensified activities to develop assistive robots to complement a dwindling workforce and healthcare robots to help caring for an ageing population [5].

A major challenge in designing intelligent robots is to equip them with the capabilities to effectively use past experiences and at the same time efficiently learn new skills to perform different tasks in different environments. Real-world dynamics is usually uncertain, unstructured, and non-stationary; the tasks, objectives, resources, and conditions of the robots may also

* Corresponding author.
E-mail address: nttrung@adobe.com (T.T. Nguyen).
[1] The main part of the work was done while the author was at the National University of Singapore.

change over time. The skills required for successful problem solving and decision making are not easily described as simple rules, guidelines, or processes. A self-learning agent that can select useful environmental features and learn to behave, to a certain degree, based on trial and error is a promising approach to building robots that act in the real world.

1.1. *Reinforcement learning in robotics*

Reinforcement learning (RL) [6] trains an autonomous agent to behave intelligently based on the feedback it receives when interacting with the environment. An RL *task* is commonly represented as a Markov decision process (MDP) with a specific *domain* that includes the relevant actions and states with the defining features, and unknown *dynamics* (transition) and/or *goal* or *feedback* (reward) functions. The main challenges of applying RL in robotic domains include the multidimensional state and action spaces that incur high computational costs, the physical constraints that make acting in the real world much more difficult than in simulated worlds, the uncertainty due to partial observability of the physical environments and inherent noise in the sensor measurements, and the difficulty in tailoring the feedback or reward functions to guide intelligent behavior of the robots [7].

Despite the challenges, RL has recently been applied in complex tasks such as helicopter maneuvering [8], soccer robots [9], and robot navigation [10] with reasonable success. In these cases, careful selections of problem representations, prior experiences, and efficient approximations have rendered RL computationally feasible in the complex settings.

In this work, we design a representation of the world dynamics in model-based RL that allows efficient and scalable approximation of the agent's action effects. At the core of our method is the ability to automatically select the relevant features of the environment that allow the agent to predict how the environment reacts to its actions. This ability to automatically learn to *focus attention* on the critical factors of the problem is one of the crucial elements needed to make intelligent behavior in complex environments computationally feasible.

The value of online feature extraction is further accentuated in situations where the agent encounters many different environments in its lifetime, and where *transfer learning* is essential. While manually designing a small set of important features is non-trivial for many real-world tasks, doing so for different environments, some of which may be unknown *a priori*, is even harder. For example, the Mars exploration rover, Opportunity, is now expected to travel tens of kilometers through different terrains, with possibly varying characteristics and dynamics, to collect scientific objects [11]. Most of the environmental features that it will encounter are unlikely to be specified beforehand.

1.2. *Transfer learning in robotic reinforcement learning*

In the original RL framework, the agent's knowledge is task and domain specific. A small change in the task or its domain may render the agent's accumulated knowledge useless; costly relearning from scratch is often needed. In transfer learning, an agent applies the knowledge or experience gained in previous (source) tasks to influence and improve the performance of new, related (target) tasks. Transfer in RL assumes that knowledge from one or more source task(s) is used to learn one or more target task(s) faster than if transfer was not used. In contrast to *multi-task learning*, which assumes that all the tasks are drawn from an underlying, possibly unknown distribution, transfer learning is more general in allowing transfer amongst tasks with heterogeneous domains, dynamics, and goals [12]. Transfer learning is hence most relevant in building *lifelong learning* agents or robots – agents that can learn, retain, and transfer knowledge acquired from multiple tasks over multiple domains, in a sequential manner, to develop more accurate solutions or policies in learning new tasks over its lifetime [13].

Transfer in RL is different from another main category of transfer learning tasks that focus on classification, regression, and clustering in non-dynamic domains [14]. The main technical challenges involved, however, are similar in identifying the commonalities between the source tasks or domains and the target tasks or domains. Transfer in RL addresses the issues of what to transfer, how to transfer, and when to and when not to transfer to avoid or minimize negative effects on the learning performance in the new environment.

Many existing RL transfer techniques, however, assume the same state-action spaces i.e., *homogeneous* domains in different tasks. This assumption may not work well in real-life applications. For example, many environmental cues that help an agent navigate through a forest are simply missing when the agent tries to navigate at sea. While recent efforts have addressed inter-task transfer in heterogeneous domains or different state-action spaces [15–19], such mappings are hard to define when the agent operates in real-world environments with large state-action spaces and multiple goal states, with possibly different state feature distributions and world dynamics. A trade-off between computational complexity and sample efficiency is usually involved in automatically learning the mappings [20].

We propose an efficient, online system that tries to transfer old knowledge, but at the same time evaluates new options to see if they work better. The agent gathers experience during its lifetime and enters a new environment equipped with *expectations* on how different aspects of the world affect the outcomes of the agent's actions. The main idea is to allow an agent to collect a library of world models or representations, called *views*, that it can consult to focus attention in a new task. In this paper, we concentrate on approximating the dynamics or transition model. The feedback or reward model library can be learned in an analogous fashion. Effective utilization of the library of world models allows the agent to capture the transition dynamics of the new environment quickly; this should lead to a jumpstart in learning and faster convergence to a near optimal policy. A main challenge is in learning to select a proper view for a new task in a new environment, without any predefined mapping strategies. The ability to learn new views is also critical in the complex, and feature-rich real environments.

Through scoring the learned views and simultaneous evaluation without the views, our system mitigates potential negative effects by balancing the exploration-and-exploitation trade-off. This is in contrast to minimizing negative transfer effects through determining the underlying relationships between the experiences and the new tasks and domains [14]. Such relationships may not exist and are usually hard to determine in the heterogeneous, dynamic environments that we focus on.

1.3. A new transfer learning framework

We present a new transfer learning framework that supports all the essential steps of a fully autonomous RL transfer agent, as compared with most existing work that mainly focuses on one or two of these steps [12]:

1. Select an appropriate source task or set of source tasks to transfer to a target task;
2. Learn the relationships between the source and the target tasks; and
3. Transfer the relevant knowledge from the source task to the target task to improve learning performance.

Our system comprises the following components:

- Situation-calculus Markov decision process (CMDP), a new variant of factored MDP that compactly represents the relevant action effects in a dynamic environment;
- A collection of *views*, which summarize the experiences learned from constructing transition models in different environments based on multinomial logistics regression; the views are scored based on their relevance in the new environments;
- A view or transition model learning algorithm, multinomial Dual Averaging with Group Lasso (mDAGL), that supports online feature selection and relevance-focused updating;
- A new reinforcement learning algorithm, logistic regression RL (loreRL), that applies mDAGL on a CMDP to learn the transition models by automatically focusing on the relevant features in the environment; and
- A new transfer learning algorithm, Transfer ExpectationsS (TES), that may utilize the previously learned transition models to improve reinforcement learning in new environments.

The rest of the paper is organized as follows. We will next formalize the problem and describe the method of online feature selection and collecting views into a library. We will then present an efficient implementation of the proposed transfer learning technique. After discussing related work, we will demonstrate the effectiveness of our system through a set of simulated experiments and a real robotic application. We will conclude by discussing the lessons learned and the implications for future robotic research.

2. Background

A task or a problem M for an intelligent agent or robot to solve, e.g., navigate to the master bedroom or fetch a cup from the kitchen table, is typically modeled as a Markov decision process (MDP) defined by a tuple $M = (S, A, T, R)$, where S is a set of states; A is a set of actions; $T : S \times A \times S \to [0, 1]$ is a transition function or model, such that $T(s, a, s') = P(s'|s, a)$ indicates the probability of transiting to a state s' upon taking an action a in state s; $R : S \times A \to \mathcal{R}$ is a reward function or model indicating immediate expected reward after an action a is taken in state s. In RL, the state-action space $S \times A$ defines the *domain* of the task, the transition model T and the reward model R define the *objective* of the task [21]; the transition model T and (sometimes) the reward model R are unknown. The main challenge of transfer learning in RL is in transferring across heterogeneous domains, i.e., where the state-action spaces, and possibly the feature spaces, are different.

For each task M, the task or problem solution is to find a policy π that specifies an action to perform in each state so that the expected accumulated future reward (with possibly higher weights for more recent rewards) for each state is maximized [6]. The optimal policy π is usually derived by a *model-based* or a *model-free* approach in RL; the latter does not consider the transition model T in deriving the solution. The two RL approaches are based on different assumptions, problem characteristics, and constraints. We adopt the model-based approach as it is easier to incorporate domain or expert knowledge into the transition and reward models; knowledge transfer across different environments can also be facilitated through the models.

In model-based RL, the optimal policy is derived by first estimating the transition model T (and the reward model R, if necessary) through interacting with the environment. This study focuses on learning the models, and remains relatively agnostic about the actual planning techniques. However, since the purpose is to build a model about world dynamics, planning systems that rely on simulators appear most natural.

3. A multi-view transfer learning framework

A key idea of this work is that the agent can represent the world dynamics from its sensory state space in different ways. Such different *views* correspond to the agent's decisions to focus attention on only some features of the state in order

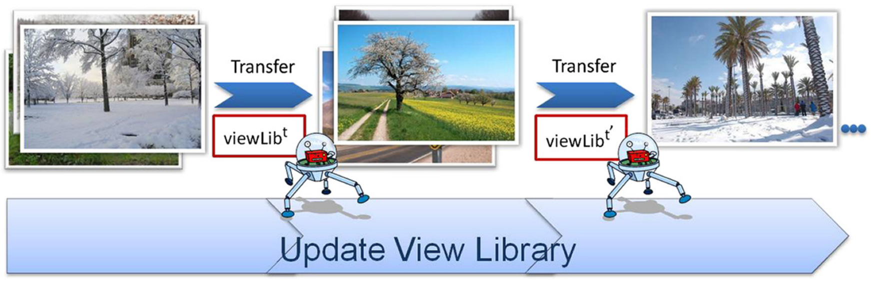


Fig. 1. Our life-long learning agent.

Algorithm 1 Overview of the proposed learning framework.

```
Input: View library $L$.
Output: Updated view library $L$.

Initialize the system for a task
$L^H$: historical record of how good each view was in previous tasks
$L' \leftarrow L$ /*A WORKING COPY OF $L$ TO FIND THE DYNAMICS OF THE CURRENT ENVIRONMENT*/
$B \leftarrow$ initialize belief of how well each view can approximate the current
      environment dynamics based on $L^H$.
/*———WHILE INTERACTING WITH THE ENVIRONMENT———*/
for $t = 0, 1, 2, \ldots$ do
   Select views to approximate the world transition dynamics
   $\{\mathbb{W}\} \leftarrow$ select the most promising views from $L'$ based on $B$
   Interact with environment based on that approximate model
   $\pi \leftarrow$ plan an action policy based on $\{\mathbb{W}\}$
   $s_t$ : current state in the environment
   $a_t \leftarrow$ choose an action to perform in $s_t$ according to action policy $\pi$
   Perform action $a_t$ and observe feedback: action outcomes from the environment
   Score all views with the new feedback
   $\mathbb{S} \leftarrow$ score all views in $L'$ with the new feedback
   $B \leftarrow$ update belief about the views in $L'$ based on the score $\mathbb{S}$
   Adjust all views with new feedback
   $L' \leftarrow$ Adapt all views toward this current environment based on the new feedback
   Break when the task ends
end for
Update view library L
$L \leftarrow$ Update the view library, e.g.
  If a view $\mathbb{W}$ is different from existing views in $L$,
    which means the transition dynamics of the corresponding action is new,
  then add that view $\mathbb{W}$ to $L$;
  else replace the old view in $L$ with the newly updated view $\mathbb{W}$.
```

to quickly approximate the state transition function. In other words, a view represents an *expectation* of the agent about the transition dynamics resulting from one (or several) of its actions in the task environment. In a new task, the agent will select appropriate views to solve the task, and to learn new views if the environment is novel. Fig. 1 illustrates the workflow of our life-long learning agent. Parts of this work have been published in [22–24].

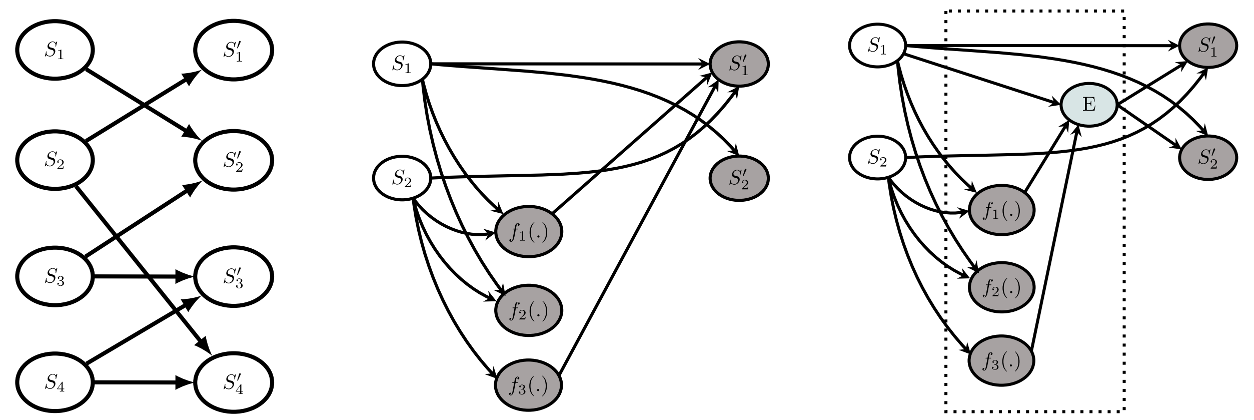
We sketch the overall behavior of our agent in Algorithm 1.

When the agent does not have any initial information about the transition dynamics of the environment in a new task, it selects "expectations" or views based on history that tells how well each view in the library has worked in previous tasks. We assume that the more frequently a view has worked in the past, the more likely it will work again in a new task.

The agent then operates according to the policy learned based on the transition model built from the selected views, assuming that the reward model is known. However, since the views have been selected without any reference to the actual characteristics of the new environment, it is highly likely that these views are inappropriate for the current task. In other words, the "expected" transition model may not correctly approximate the true transition dynamics of the current environment. The resulting policy may just perform poorly, leading to negative rewards.

To limit such negative transfer effects, our agent exploits the outcomes or feedback for each of its actions, in terms of state changes and rewards, to score all the views in the library. The score estimates the capability of each view in approximating the transition dynamics in the current environment; it is a primary criterion to re-select the views for subsequent decisions.

The new task and/or environment, however, may actually be very different from all the past experiences of the agent. In this case, none of the recorded views can capture or adapt to the new transition dynamics. Hence, environment feedback is also used to develop and incorporate new views into the library. These steps are repeated until a stopping criterion is met.



(a) Decomposing the transition function with a regular DBN.

(b) A DBN with separate state attributes (S_1 and S_2) state features (f_1, f_2, f_3).

(c) Customized DBN for CMDP with an effect variable E.

Fig. 2. Different ways to decompose a state transition function.

At the end of a task, the selected views will be recorded as new additions in the library if the transition dynamics captured is significantly different from the existing views. Otherwise, the agent would just update the existing views accordingly. Views that have not been used for a "long time" will be pruned to manage the library size.

We address three main technical challenges in this framework: First, the transition model $T(S, A, S)$ is task specific, which is probably a reason why there have not been many studies that transfer the transition model. Second, learning or updating a view or a transition model online in a complex and feature-rich environment is computationally expensive. Third, the view scoring method must be simple to be calculated online, based on feedback from the environment. The view library also needs to be efficiently updated.

3.1. *Situation calculus MDP: CMDP*

In a *factored* MDP, each state s is represented by a vector of n state-attributes $s_i, i = 1, \dots, n$. Each state attribute is a random variable that can take on multiple values; each state s is defined by the Cartesian product of the n state attributes. For example, the binary state attributes *Water* (with values *present* and *absent*) and *Martians* (with values *present* and *absent)* will define all the states that describe the "interestingness" of a particular location on Mars: {(water, Martians), (water, no-Martians), (no-water, Martians), (no-water, no-Martians)}.

The transition function or model for the factored states is commonly expressed in dynamic Bayesian networks (DBNs), a temporal variant of Bayesian networks. In the DBN representation of the transition model, $T(s, a, s') = \prod_{i=1}^{n} P(s_i' | Pa_i^a(s), a)$, where s_i is a state-attribute, Pa_i^a indicates a subset of state-attributes in s called the parents of s_i' (Fig. 2a), i.e., the attributes of s from which there are arcs to s_i' in the DBN. Learning T requires learning the subsets Pa_i^a and the parameters for conditional distributions, or the DBN local structures. A critical issue in this MDP formulation is the ambiguous definitions of the factored attributes or variables. The state-attributes or features serve to both define the state space and capture information about the transition model, even if these two purposes can be very different. For example, two state-attributes, the (x, y)-coordinates uniquely identify a state and compactly factorize the state space in a grid-world. A policy can be learned on this factored space. The state transition dynamics, however, may depend on other features of the state, such as the surface material at the location (state), the presence or absence of Martians, etc. Such features are often included in the state representations. While essential in formulating the transition or reward models, these features may complicate the planning or learning processes by increasing the size and complexity of the state space.

We present a variant of the factored MDP that defines a "compact but comprehensive" factorization of the transition function and supports efficient learning of the relevant features. We separate the state identifying *state-attributes* from the "merely" informative *state-features* in our representation (see Fig. 2b). This way, we can apply an efficient feature selection method on a large number of state features to capture the transition dynamics, while maintaining a compact state space.

Similar to the approach proposed by Konidaris and Barto [25], the state attributes could be defined in terms of the *agent-space* representation based on the capabilities afforded by the sensors or actuators of the robot, e.g., the (x, y)-coordinates of the robot location. The state features, on the other hand, could be defined in terms of the *problem-space* representation based on the domain or environmental characteristics of the task, e.g., the terrain or wetness of the floor surface, which may or may not be accurately detectable by the agent or robot.

Learning DBN structures of the transition function online, i.e., while the agent is interacting with the environment, is still computationally prohibitive in most domains. On the other hand, recent studies [26,27] have shown encouraging results in learning the structure of logistic regression models, which can effectively approximate the local structures in DBNs even in high dimensional spaces. While these regression models cannot fully capture the conditional distributions, their expressive power can be improved by augmenting low dimensional state representation with non-linear features of the state vectors. We will introduce an online sparse multinomial logistic regression method that supports efficient learning of the structured representation of the transition function in Section 4.

In addition, we predict the relative changes of states instead of directly specifying the next state-attributes in a transition (see Fig. 2c). In RL, an action will stochastically create an effect that determines how the current state changes to the next one [28,10,17]. Mediating state changes via action effects is a common strategy in the classic *situation calculus* [29]. Since the number of relative changes or action effects is usually much smaller than the size of the state space, or the size of state-attribute domains, the corresponding prediction task should be easier. The learning problem can then be expressed as a multi-class classification task of predicting the action effects.

Usually, the transition functions and reward functions are defined in terms of the state features – aspects of the state representation that help in predicting the action effects. In a specific task or environment, the same action in different states (e.g., different locations) with the same features tend to yield the same action effects (i.e., relative changes in the states).

Formally, a situation calculus MDP (CMDP) is defined by a tuple $M = (S, f, A, T, E, R, \gamma)$, where S, A, T, R, γ have the same meaning as in a regular MDP. $S = \langle S_1, S_2, .., S_n \rangle$ is the state space implicitly represented by vectors of n *state-attributes*. The function $f : S \to \mathbb{R}^m$ extracts m *state-features* from each state. E is an *action effect* variable such that the transition function can be factored as

$$T(s,a,s') = P(s' \mid s,a) = \prod_{i=1}^{n} P(s'_i \mid s, f(s), a)$$
$$= \prod_{i=1}^{n} \sum_{e \in E} P(s'_i \mid e, s) P(e \mid s, f(s), a). \tag{1}$$

Fig. 2c shows an example of this decomposition. The agent uses the feature function f to identify the relevant features, and then uses both state attributes and features to predict the action effects. We also assume that the effect e and current state s determine the next state s', thus $P(s'|e,s)$ is either 0 or 1. This defines the semantic meaning of the effect which is assumed to be known by the agent. The remaining task is to learn $P(e|s,a) = P(e|x(s),a)$, where $x(s) = (s, f(s))$ is a vector containing both the state attributes and the state features. Assuming the effect e is discrete, learning $P(e \mid x(s), a)$ is a classification problem. In Section 4 we will show how to solve this problem by using multinomial logistic regression methods.

3.2. *Transferring expectations: TES*

In our framework, the knowledge gathered and transferred by the agent is collected into a library $\mathcal{T}$ of online effect predictors or views. A *view* consists of a structure component $\bar{f}$ that picks the features which should be focused on, and a quantitative component Θ that defines how these features should be combined to approximate the distribution of action effects. Formally, a *view* is defined as $\tau = (\bar{f}, \Theta)$, such that $P(E|S,a;\tau) = P(E|\bar{f}(S),a;\Theta) = \tau(S,a,E)$, in which $\bar{f}$ is an orthogonal projection of $x(s)$ to some subspace of $\mathbb{R}^m$, where m is the dimension of the subspace. Each view τ is specialized in predicting the effects of one action $a(\tau) \in A$ and it yields a probability distribution for the effects of the action a in any state. This prediction is based on the features of the state and the parameters $\Theta(\tau)$ of the view that may be adjusted based on the actual effects observed in the task environment.

We denote the subset of views that specify the effects for action a by $\mathcal{T}^a \subset \mathcal{T}$. The main challenge is to build and maintain a comprehensive set of views that can be used in new environments likely resembling the old ones, but at the same time allow adaptation to new tasks with completely new transition dynamics and feature distributions.

At the beginning of every new task, the existing library is copied into a working library which is also augmented with fresh, uninformed views, one for each action, that are ready to be adapted to new tasks. We then select, for each action, a view with a good “track record”, i.e., it has been “used” or applied many times or with a high recency-weighted *score*. This view is used to estimate the optimal policy based on the transition model specified in Equation (1), and the policy is used to pick the first action a. The action effect is then used to score all the views in the working library and to adjust their parameters. In each round the selection of views is repeated based on their scores, and the new optimal policy is calculated based on the new selections. At the end of the task, the actual library is updated by possibly recruiting the views that have “performed well” and retiring those that have not. A more rigorous version of the procedure is described in Algorithm 2.

3.2.1. *Scoring the views*

To assess the quality of a view τ, we measure its predictive performance by a cumulative log-score. This is a *proper* score [30] that can be effectively calculated online.

Given a sequence $D^a = (d_1, d_2, \ldots, d_N)$ of observations $d_i = (s_i, a, e_i)$ in which action a has resulted in effect e_i in state s_i, the score for an a-specialized τ is

$$S(\tau, D^a) = \sum_{i=1}^{N} \log \tau(s_i, a, e_i; \theta^i(\tau)),$$

Algorithm 2 TES: **T**ransferring **E**xpectation**s** using a library of views.

```
Input: $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$: view library; CMDP$_j$: a new $j$th task; $\Phi$: view goodness evaluator
Let $\mathcal{T}_0$ be a set of fresh views – one for each action
$\mathcal{T}_{tmp} \leftarrow \mathcal{T} \cup \mathcal{T}_0$     /* THE WORKING LIBRARY FOR THE TASK */
for all $a \in A$ do   $\hat{T}[a] \leftarrow \arg\max_{\tau \in \mathcal{T}^a} \Phi(\tau, j)$   end for     /* SELECTING VIEWS */
for $t = 0, 1, 2, \ldots$ do
    $a_t \leftarrow \hat{\pi}(s_t)$, where $\hat{\pi}$ is obtained by solving MDP using transition model $\hat{T}$
    Perform action $a_t$ and observe effect $e_t$
    for all $\tau \in \mathcal{T}^{a_t}_{tmp} \cup \mathcal{T}^{a_t}$ do   $Score[\tau] \leftarrow Score[\tau] + \log \tau(s_t, a_t, e_t)$   end for
    for all $\tau \in \mathcal{T}^{a_t}_{tmp}$   do   Update view $\tau$ based on $(f(s_t), a_t, e_t)$   end for
    $\hat{T}[a_t] \leftarrow \arg\max_{\tau \in \mathcal{T}^{a_t}_{tmp}} Score[\tau]$     /* SELECTING VIEWS */
end for
for all $a \in A$ do   $\tau^* \leftarrow \arg\max_{\tau \in \mathcal{T}^a_{tmp}} Score[\tau]$;
                       $\mathcal{T}^a \leftarrow growLibrary(\mathcal{T}^a, \tau^*, Score, j)$     /* UPDATING LIBRARY */
end for
if $|\mathcal{T}| > M$ then   $\mathcal{T} \leftarrow \mathcal{T} - \{\arg\min_{\tau \in \mathcal{T}} \Phi(\tau, j)\}$   end if   /* PRUNING LIBRARY */
```

Algorithm 3 Grow sub-library $\mathcal{T}^a$.

```
Input: $\mathcal{T}^a, \tau^*, Score, j$: task index; $c$: constant; $H_{\tau^*} = \{\}$: empty history record
Output: updated library subset $\mathcal{T}^a$ and winning histories $H_{\tau^*}$
case $\tau^* \in \mathcal{T}^a_0$   do   $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\}$     /* ADD NEWBIE TO LIBRARY */
otherwise   do   Let $\bar{\tau} \in \mathcal{T}$ be the original, not adapted version of $\tau^*$
                 case $Score[\tau^*] - Score[\bar{\tau}] > c$   do   $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\}$
                 otherwise   do   $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\} - \{\bar{\tau}\}$
                                  $H_{\tau^*} \leftarrow H_{\bar{\tau}}$     /* INHERIT HISTORY */
$H_{\tau^*} \leftarrow H_{\tau^*} \cup \{j\}$
```

where $\tau(s_i, a, e_i; \theta^i(\tau))$ is the probability of effect e_i given by the view or effect predictor τ based on the features of state s_i and the parameters $\theta^i(\tau)$ that may have been adjusted using previous data $(d_1, d_2, \ldots, d_{i-1})$.

3.2.2. *Growing the library*

After completing a task, the highest scoring views for each action are considered for recruiting into the actual library. The winning "newbies" or new entries are automatically accepted. In this case, the data has most probably come from the distribution that is far from the any current models, otherwise one of the current models would have had an advantage to adapt and win. In other words, this means that the existing views have generated low or negative rewards in the process of deriving the optimal policy in RL; they have not been used further and a new set of world dynamics has been learned. Instead of trying to directly identify the underlying commonalities between the old and new tasks and/or environments, our framework mitigates potential negative transfer effects by learning the new views or transition dynamics in parallel to exploiting and examining if the existing views can improve learning. This ability is important in real domains where the underlying system dynamics may not be easily determined.

The winners τ^* that are adjusted versions of old views $\bar{\tau}$ are accepted as new members if they score significantly higher than their original versions, based on the logarithm of the prequential likelihood ratio [31] $\Lambda(\tau^*, \bar{\tau}) = S(\tau^*, D^a) - S(\bar{\tau}, D^a)$. Otherwise, the original versions $\bar{\tau}$ get their parameters updated to the new values. This procedure is just a heuristic and other inclusion and updating criteria may well be considered. The policy is detailed in Algorithm 3.

3.2.3. *Pruning the library*

To keep the library relatively compact, a plausible policy is to remove views that have not performed well for a long time, possibly because there are better predictors or they have become obsolete in the new tasks or environments. To implement such a retiring scheme, each view τ maintains a list H_τ of task indices that indicates the tasks for which the view has been the best scoring predictor for its specialty action $a(\tau)$. We can then calculate the recency weighted track record for each view. In practice, we have adopted the procedure by Zhu et al. [32] that introduces the recency weighted score at time T as

$$\Phi(\tau, T) = \sum_{t \in H_\tau} e^{-\mu(T-t)},$$

where μ controls the speed of decay of past success. Other decay functions could naturally also be used. The pruning can then be done by introducing a threshold for recency weighted score or always maintaining the top m views.

4. A view learning algorithm

In *TES*, a view can be implemented by any probabilistic classification model that can be quickly learned *online*. This requirement excludes most of the batch or offline classification methods. In this study, we introduce a scalable online sparse multinomial logistic regression algorithm to incrementally learn a view. The proposed algorithm optimizes an objective

Algorithm 4 The *mDAGL* algorithm.

```
Input: $\lambda, \alpha$

Let $h(W)$ be a strongly convex function with modulus 1
Let $W^1 = W^0 = \arg\min_W h(W)$
Let $\bar{G}^0 = \bar{0}$
for $t = 1, 2, 3, \ldots$ do
    $(y^t, x^t) \leftarrow$ observe data
    $(W^{t+1}, \bar{G}^t) \leftarrow$ mDAGL-update$(t, y^t, x^t, W^t, \bar{G}^{t-1}, \lambda, \alpha)$
end for
```

function similar to the *group lasso* [33] which has been recently suggested for efficient feature selection among a very large set of features [27].

Multinomial logistic regression is a simple yet effective classification method. Assuming K classes of d-dimensional vectors $x \in \mathbb{R}^d$, we represent each class k with a d-dimensional prototype vector W_k. Classification of an input vector x is based on how "similar" it is to the prototype vectors. Similarity is measured with inner product $\langle W_k, x\rangle = \sum_{i=1}^{d} W_{ki}x_i$, where x_i denotes feature i. The probability of a class is defined by $P(y = k \mid x; W_k) \propto e^{\langle W_k, x\rangle}$. The parameter vectors of the model form the rows of a matrix $W = (W_1, \ldots, W_K)^T$.

Let $l_t(W^t) = -\log P(y^t|x^t; W^t)$ denote the item-wise log-loss of a model with coefficient matrix W^t predicting a data point (y^t, x^t) observed at time t. A typical objective of an online learning system is to minimize the total loss by updating its W^t over time. However, the resulting model will often be very complicated and over-fitting. To achieve a parsimonious model, we express our *a priori* belief that most features are irrelevant or superfluous by introducing a regularization term $\Psi(W) = \lambda \sum_{i=1}^{d} \sqrt{K}\|W_{\cdot i}\|_2$, where $\|W_{\cdot i}\|_2$ denotes the 2-norm of the ith column of W, and λ is a positive constant. This regularization is similar to that of group lasso. It communicates the idea that it is likely that a whole column of W has zero values (especially, for large λ). A column of all zeros suggests that the corresponding feature is not necessary for classification.

The objective function can now be written as

$$L(T) = \sum_{t=1}^{T} l_t(W^t) + \Psi(W^t)$$
$$= \sum_{t=1}^{T} -\log \frac{e^{\langle W^t_{y^t}, x^t\rangle}}{\sum_k e^{\langle W^t_k, x^t\rangle}} + \lambda \sum_{i}^{d} \sqrt{K}\|W^t_{\cdot i}\|_2,$$

where W^t is the coefficient matrix learned using $t-1$ previously observed data items. The quality of a sequence of parameter matrices $W^t, t \in (1, \ldots, T)$ with respect to a fixed parameter matrix W can be measured by the amount of extra loss, or regret

$$R_T(W) = L(T) - L_W(T)$$
$$= \sum_{t=1}^{T} (l_t(W^t) + \Psi(W^t)) - \sum_{t=1}^{T} (l_t(W) + \Psi(W)).$$

We want to learn a series of parameters W^t to achieve small regret with respect to a good model W that has a small loss $L_W(T)$.

4.1. Online learning of multinomial regularized logistic regression with group lasso: mDAGL

Recently, Xiao et al. [26] introduced a dual averaging method for solving lasso online, and Yang et al. [27] extended the work for solving group lasso. The methods are simple, efficient, and scalable for learning the regularized regression models. Following the same approach, we introduce mDAGL (Algorithm 4), a new algorithm to incrementally learn W to optimize the specified objective function in the sparse multinomial logistic regression. Typically, group lasso is used to regularize groups of coefficients where each coefficient corresponds to a particular feature. In our case of multinomial logistic regression, a group comprises the coefficients of a feature; in other words, a group is a column in coefficient matrix W.

In the standard online stochastic gradient descent method, after observing the data vector (y^t, x^t), we adjust the parameters of the model toward the directions that maximize the likelihood of the observation (or equivalently, minimize the item-wise log-loss l_t). The dual averaging method is a version of the stochastic gradient descent, thus we again have to compute the gradient G^t of the log-loss function for each observation (y^t, x^t). But instead of moving the parameters based on these directions G^t, we use G^t to update the average of gradients, $\bar{G}^t$, for observations encountered this far, and move the parameters away from those average directions (away, since we are minimizing). We will next describe the method more formally.

Algorithm 5 mDAGL-update.

```
Input: $t, y^t, x^t, W^t, \bar{G}^{t-1}, \lambda, \alpha$

$G^t \leftarrow$ use equation (2) with $(y^t, x^t), W^t$
$\bar{G}^t \leftarrow \frac{t-1}{t}\bar{G}^{t-1} + \frac{1}{t}G^t$
$W^{t+1} \leftarrow$ use equation (A.1) with $\bar{G}^t, \beta_t = \alpha\sqrt{t}, \lambda$
return $(W^{t+1}, \bar{G}^t)$
```

Algorithm 6 The loreRL algorithm.

```
Input: mDAGL regularization parameters $\lambda$, $\alpha$, CMDP variables $S, f, A, E, R, \gamma$, exploration $\epsilon$.
Let $W = (W_1, W_2, \dots, W_{|A|}) = (W^0, W^0, \dots, W^0)$
Let $\bar{G} = (\bar{G}_1, \bar{G}_2, \dots, \bar{G}_{|A|}) = (\vec{0}, \vec{0}, \dots, \vec{0})$
$s_0 \leftarrow$ random initial state
for $t = 1, 2, 3, \dots$ do
    $\pi \leftarrow$ Solve MDP using transition model $T(\bar{W})$
    $a \leftarrow \pi(s^t, \epsilon)$          #$\epsilon$-greedy action selection
    Take action $a$ yielding effect $e$, next state $s^{t+1}$
    $(W_a, \bar{G}_a) \leftarrow$ mDAGL($t, e, x(s^t), W_a, \bar{G}_a, \lambda, \alpha$)
end for
```

We initialize the parameters W to a $K \times d$ matrix of all zeros. Let G^t_{ki} be the partial derivatives of function $l_t(W)$ with respect to W_{ki} at W^t ($G^t_{ki} = \frac{\partial l_t}{\partial W_{ki}}(W^t)$). We define $\bar{G}^t$ to be a matrix of average partial derivatives, i.e., $\bar{G}^t_{ki} = \frac{1}{t}\sum_{\tau=1}^{t} G^\tau_{ki}$, where taking the gradient of the loss function l_τ with respect to the parameter W_{ki} gives us the formula

$$G^\tau_{ki} = -x^\tau_i (I(y^\tau = k) - P(k|x^\tau; W^\tau)). \tag{2}$$

We notice that this partial derivative points either away or towards the observed feature x^τ_i depending on whether the observation belongs to the class k or not.

For any data observed at time t, we use the average gradient $\bar{G}^t$ to update the coefficient matrix W via

$$W^{t+1} = \arg\min_W \left(\langle \bar{G}^t, W \rangle + \Psi(W) + \frac{\beta_t}{t} h(W) \right), \tag{3}$$

where β_t is a non-negative, non-decreasing sequence of real numbers, and $\langle\cdot,\cdot\rangle$ denotes an inner product between two matrices; $\langle \bar{G}^t, W\rangle = \sum_{k,i} \bar{G}^t_{ki} W_{ki}$. The first term is minimized by W that points to the direction $-\bar{G}^t$. While the first term prefers long vectors, the regularization term $\Psi(W)$ balances this out. The third term introduces an extra regularization in terms of a strongly convex function $h(W)$ that is needed for convergence and sparsity. In practice we use the Froebenius norm $h(W) = \sum_{k,i} W^2_{ki}$ and $\beta_t = \sqrt{t}$.

Solving the minimization problem above leads to an update rule (Algorithm 5) in which each column of the W^{t+1} is a scaled version of the corresponding column of the $\bar{G}^t$. Furthermore, if the length of the average gradient matrix column is small enough, the corresponding parameter column should be truncated to zero. This amounts to feature selection. The full definition and proof of the rule are detailed in Appendix A.

A regret analysis confirms that the solution will converge and that the average maximal regret asymptotically approaches zero with rate $O(\frac{1}{\sqrt{t}})$. The full analysis is detailed in Appendix B.

4.2. Model-based RL with multinomial logistic regression: loreRL

Our main task is to turn transition model learning into the learning of conditional distributions $P(E \mid s, f(s), a)$ using multinomial logistic regression for which attention to relevant features can be efficiently implemented online via mDAGL.

The key steps of our method, called loreRL (RL with regularized logistic regression), are presented in Algorithm 6. Inputs to loreRL are the CMDP components (except the transition function), regularization parameters λ and α of mDAGL algorithm, and the ϵ that determines the probability of taking a random action. We first initialize logistic regression parameters W_a and the average gradient matrices $\bar{G}_a$ for each action $a \in A$. We also randomly select a starting state s^0.

At each time step, a random action a is chosen with a small probability ϵ, but otherwise we calculate the optimal policy π for an MDP with the transition model $T(W)$ is based on the current effect predictors. While we have used value iteration (like in *Rmax*) for finding the optimal policy, any other model-based RL technique can be used as well. We do not focus on the planning part of RL here, but search heuristics such as those used in Dyna-Q [34] or Prioritized Sweeping [35] can be deployed for a more scalable algorithm. After performing an action a in state s^t and observing its effect e, the experience $(e, s^t, f(s^t))$ will be presented to the mDAGL algorithm that updates the parameter matrix W_a and the gradient matrix $\bar{G}_a$.

As we just do ϵ-greedy random sampling, it is impossible to guarantee PAC convergence to an optimal policy. Assuming that observed data is i.i.d., we can prove that difference in optimal value functions of two CMDPs with different logistic regression based transition functions is bounded by the difference in their parameters. This leads to a corollary for convergence to near optimal policy.

Theorem 1 *(Difference in value function). Let $M_1 = (S, f, A, T(W^{M_1}), E, R, \gamma)$ and $M_2 = (S, f, A, T(W^{M_2}), E, R, \gamma)$ be two CMDPs with optimal policies π_1 and π_2 respectively. Let us denote by V_π^M the value function for policy π in CMDP M. Let*

$$\epsilon_1 = 2\sqrt{\max_{a\in A, e\in E} \|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1},$$

then

$$\max_{s\in S}\left(V_{\pi_1}^{M_2}(s) - V_{\pi_2}^{M_2}(s)\right) \le \frac{2\gamma V_{\max}\epsilon_1}{1-\gamma},$$

where $W_e^{(a),M_1}$ and $W_e^{(a),M_2}$ refer to the vector of coefficients corresponding to class $E = e$ under action a in model M_1 and M_2 respectively, $\|\cdot\|_1$ is the 1-norm of vector, and V_{max} is the maximum value of any state for any policy in either of the CMDPs.

By taking M_2 to be a CMDP based on the optimal W^* and M_1 an estimated CMDP based on mDAGL, we can derive a vanishing bound for value difference of policies. In case the true transition model is representable by a sparse W^*, we would most probably converge to a near optimal policy. The full proof for the theorem is in Appendix C.

When we cannot express the true transition dynamics as logistic regression based on available state features, it is hard to give guarantees of performance. However, we can still have some confidence in doing well. The logistic regression model P_l^* closest (in Kullback–Leibler distance) to the true model P_{true} (possibly not a logistic regression model) is the one[2] that has the smallest expected log-loss. While our optimality criterion is the expected regularized log-loss, we expect the regularized log-loss optimal model P_Ψ^* to be close to P_l^* thus almost as close to P_{true} as we can get. This relatively small KL-distance can be converted to relatively small distances in actual transition probabilities, which can then further be converted to a relatively small bound on value differences by the same arguments used in proving Theorem 1 (in Appendix C). Therefore, since our model would very likely converge close to P_Ψ^*, we can expect to do almost as well as P_Ψ^*.

5. Experiments

We examine the performance of our expectation transfer algorithm *TES* that transfers views to speed-up the learning process across different complex, feature-rich, heterogeneous, and dynamic environments. We show that *TES* can efficiently:

1. learn the appropriate views online;
2. select views using the proposed scoring metric;
3. achieve a good jump start; and
4. perform well in the long run.

We first evaluate our approach in a simulated navigation domain where the assumptions hold. We then conduct a case-study in a real robotic domain to see if the theoretical results are useful in practice.

5.1. Simulated robot navigation

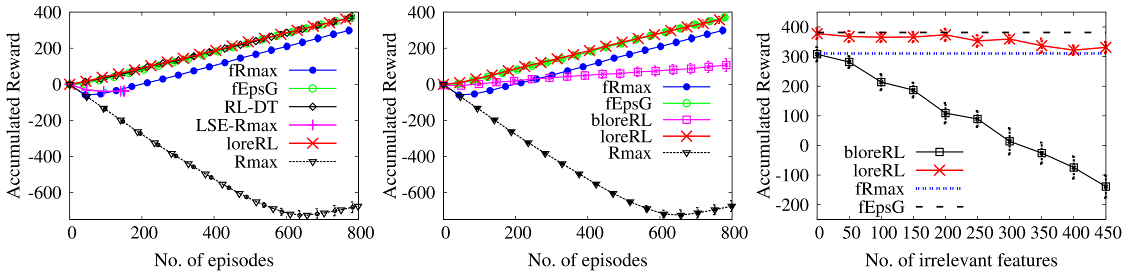
Environment. We consider a grid-based robot navigation problem in which each grid-cell has the surface of either sand, soil, water, brick, or fire. In addition, there may be walls between cells. The surfaces and walls determine the stochastic dynamics of the world. However, the agent also observes numerous other features in the environment. The agent has to learn to focus on the important features to learn the environment dynamics model, and consequently to achieve its goal.

Action, states, and rewards. The agent can perform four actions (move up, down, left, right), which will lead it to one of the four states around it or leave it in its current state. Effects of the actions are captured in five outcomes (moved up, left, down, right, did not move). The states are defined by the (x,y)-coordinates of the agent. The agent spends 0.01 units of energy to perform an action. It loses 1 unit if falling into a state of fire, but gains 1 unit when successfully reaching an exit door.

Goal. The goal is to reach any exit door in the world consuming as little energy as possible. A task ends when agent reaches a terminal state, i.e., any exit door or state with fire.

Tasks. We design fifteen tasks with grid sizes ranging from 20×20 to 30×30. Each task has a different state space and different terminal states. Each state (cell) is characterized by its surface materials and the walls around it, and 200 additional irrelevant, random binary features. The tasks may involve different dynamics as well as different distributions of the surface materials. In our experiments, the environment transition dynamics is generated using three different sets of multinomial logistic regression models; each combination of cell surfaces and walls around the cell will lead to different

[2] Such model may not always exist since the parameter set is open. However, for our argument, any model with almost infimum distance to the true model will do.



(a) CMDP with 10 features. (b) Extra 200 irrelevant features. (c) Rewards after 800 episodes.

Fig. 3. Accumulated rewards in a 900 state CMDP for various model-based RL-methods.

Table 1
Average running time per episode in 800 episodes when acting in an environment with 210 features. (Slow *RL-DT*, *LSE-Rmax* could only be run with 10 features.) Run on Intel Xeon CPU 2.13 GHz, 32 GB RAM.

Algorithm	*fRmax*	*fEpsG*	*RL-DT*	*LSE-R.*	*bloreRL*	***loreRL***
Time (sec.)	0.26	0.25	9.09	67.53	4.3	0.55

transition dynamics at the cell. The probability of going through a wall is rounded to zero and the freed probability mass is evenly distributed to other effects. The agent's starting position is randomly picked in each episode.

Transfer learning. The maximum size M of the view library, initially empty, is set to be 20; threshold $c = \log 300$. In a new environment, the *TES*-agent mainly relies on its transferred knowledge. However, we allow some ϵ-greedy exploration with $\epsilon = 0.05$. The parameters for view learning algorithm are that $\lambda = 0.05$, $\alpha = 1.5$. We conduct leave-one-out cross-validation experiment with fifteen different tasks. In each scenario the agent is first allowed to experience fourteen tasks, over 100 episodes in each, and it is then tested on the remaining task. No recency weighting is used to calculate the goodness of the views in the library. We next discuss experimental results averaged over 20 runs showing 95% confidence intervals for some representative tasks.

5.1.1. Online view learning in feature-rich environments

We show the empirical evaluation results of *loreRL* in a 900 cell/state world. We aim to demonstrate that the "single expectation" model-based RL, *loreRL*, can a) learn views that generalize and approximate the transition model to achieve fast convergence to near optimal policy, and b) with feature selection, perform well in complex, feature rich environments. We also want to see if the theoretical promises derived under assumption of i.i.d. sampling can be realized in practice. We compare accumulated rewards of *loreRL* with factored Rmax (*fRmax*), in which the network structures of transition models are known [36], and with factored ϵ-greedy (*fEpsG*), in which the optimistic Rmax exploration of *fRmax* is replaced by an ϵ-greedy strategy. We also compare our method with RL-DT [37] and LSE-Rmax [38], which are the state of the art model-based RL algorithms for learning transition models. For these tests, we run *loreRL* with $\alpha = 1.5$, $\lambda = 0.05$, $\gamma = 0.95$, exploration $\epsilon = 0.05$, parameter $m = 10$ for *fRmax*, $m = 5$ for *Rmax* ($m = 5$ is small for *Rmax*, but increasing it did not yielded better result), fixed $m = 10, \sigma = 0.99$ for *LSE-Rmax* (values originally used in [38]).

Generalization and convergence. We first show that when the feature space is small, *loreRL* performs as efficiently as the state of the art methods. *RL-DT* employs a decision tree to generalize transition dynamics knowledge over states, but it is implemented with an ϵ-greedy exploration strategy. *LSE-Rmax* appears to be the best structure learning method for ergodic factored MDPs [38]. *fRmax* and *fEpsG* have correct DBN structures provided by an oracle. All the methods are implemented with our customized DBN to incorporate domain knowledge. *Rmax* is included as a reference to show the effect of knowledge generalization.

As seen in Fig. 3a, *loreRL* can approximate the world dynamics using samples in all the states, thus it converges as fast as *fEpsG*, and *RL-DT* to near optimal policy. Although *fRmax* is provided with the correct DBN structure, its accumulated rewards are lower due to aggressive exploration to find the optimal model. After exploration the policy is guaranteed to be near optimal, but it may still take a long time (or forever) to catch up with *loreRL*. While *LSE-Rmax* follows the *Rmax* scheme, it starts with a simple model and explores a bit less aggressively than *fRmax*, gaining some advantage in early episodes. However, *LSE-Rmax* appears to require much more data to choose a more complex model. Its accumulated reward drops below *fRmax* after 150 episodes, and the angle of the curve suggests that its DBN structure is still not correct. We do not run *LSE-Rmax* for more episodes, as the algorithm is computationally very demanding (Table 1).

When the feature set includes many irrelevant features (Fig. 3b), *loreRL* is able to learn the relevant ones and still gain nearly as high accumulated rewards as *fEpsG* which has relevant features provided by an oracle. *loreRL*'s running time is also not much longer than *fRmax*'s or *fEpsG*'s (Table 1). Other methods are too slow to run in this high-dimensional environment.

These results also suggest that with ϵ-greedy exploration and random restarts, near optimal policy can be found even without i.i.d. data sampling.

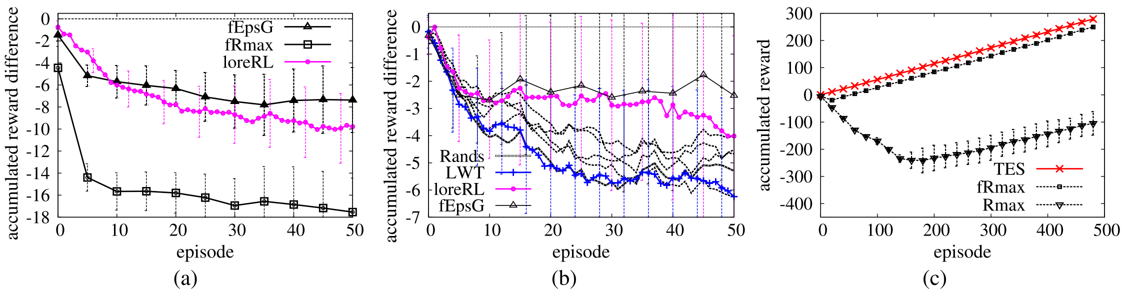


Fig. 4. Performance difference to *TES* in early trials in a) same dynamics, b) heterogeneous environments. c) Convergence.

Table 2
Cumulative reward after first episodes. For example, in Task 1 *TES* could save $(0.616 - 0.113)/0.01 = 50.3$ actions compared to *LWT*.

Methods	Tasks														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loreRL	−0.681	−0.826	−0.814	−1.068	−0.575	−0.810	−0.529	−0.398	−0.653	−0.518	−0.528	−0.244	−0.173	−1.176	−0.692
LWT	0.113	−0.966	−0.300	**0.024**	−1.205	**−0.345**	−1.104	−1.98	−0.057	−0.664	**−0.230**	−1.228	0.034	0.244	−0.564
TES	**0.616**	**−0.369**	**0.230**	−0.044	**−0.541**	−0.784	**−0.265**	**0.255**	**0.001**	**−0.298**	−1.184	**−0.077**	**0.209**	**0.389**	**−0.407**

Feature selection. To understand the role of feature selection, we compare *loreRL* with a *bloreRL* that is based on multinomial logistic regression without feature selection (without the regularization term). *fEpsG* and *fRmax* are base lines.

Fig. 3b shows the accumulated rewards when the environment has 200 irrelevant binary features. As seen, *loreRL* is still able to quickly converge to the optimal policy, and outperforms *fRmax* and *bloreRL*. Fig. 3c shows performances of *loreRL* and *bloreRL* after 800 episodes as a function of the number of irrelevant features. Only minimally affected by the actual number of irrelevant features, *loreRL* can quickly select the relevant features and outperform *bloreRL*. *loreRL* does not lose much to *fEpsG* either. While *fRmax* may find an optimal policy before *loreRL* due to aggressive exploration, its accumulated rewards are still lower than *loreRL*'s. We also observe that *loreRL*, through selecting a small set of features, runs much faster than *bloreRL* (Table 1).

5.1.2. *View selection and multi-view transfer in complex environments*

Transferring expectations between same dynamics tasks. To ensure that *TES* is capable of basic model transfer, we first evaluate it on a simple task. We train and test *TES* on two environments which have the same dynamics and 200 irrelevant binary features that challenge the agent's ability to learn a compact model for transfer. Fig. 4a shows how much the other methods lose to *TES* in terms of accumulated rewards in the test task. *loreRL* is an implementation of *TES* equipped with the view learning algorithm that does not transfer knowledge. *fRmax* is the factored *Rmax* in which the network structures of transition models are provided by an oracle [36]; its parameter m is set to be 10 in all the experiments. *fEpsG* is a heuristic in which the optimistic *Rmax* exploration of *fRmax* is replaced by an ϵ-greedy strategy ($\epsilon = 0.1$). The results show that these oracle methods still have to spend time to learn the model parameters, so they gain lower accumulated rewards than *TES*. This also suggests that the transferred view of *TES* is likely not only compact but also accurate. Fig. 4a further shows that *loreRL* and *fEpsG* are more effective than *fRmax* in early episodes.

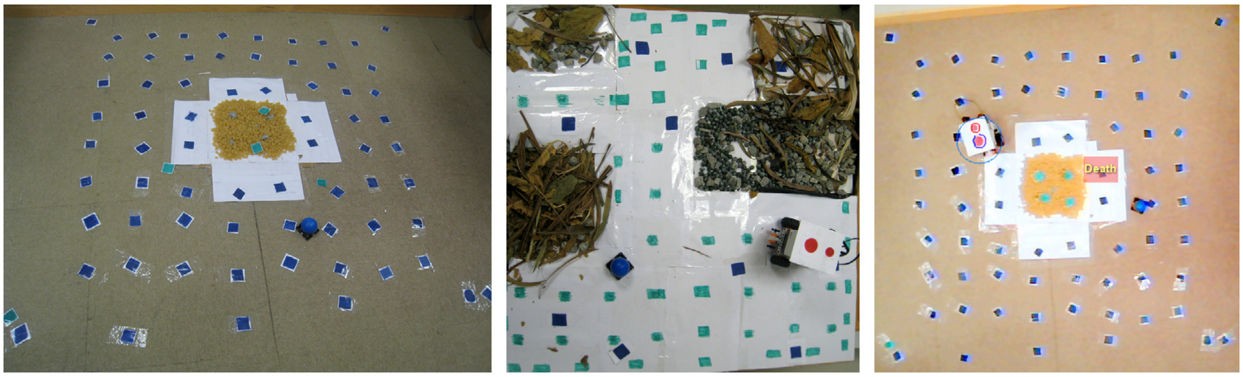
View selection vs. random views. Fig. 4b shows how different views lead to different policies and accumulated rewards over the first 50 episodes in a given task. The *Rands* curves show the accumulated reward differences with respect to *TES* when the agent follows some random combinations of views from the library. For clarity we show only 5 such random combinations. For all these curves, the differences quickly turn negative in the beginning indicating less reward in early episodes. We conclude that our view selection criterion outperforms random selection.

Multiple views vs. single view, and non-transfer. We compare the multi-view learning *TES* agent with a non-transfer agent *loreRL*, and an *LWT* [39] agent that tries to learn only one good model for transfer. We also compare with the oracle method *fEpsG*. As seen in Fig. 4b, *TES* outperforms *LWT* which, due to differences in the tasks, also performs worse than *loreRL*. When the earlier training tasks are similar to the test task, the *LWT* agent performs well. However, the *TES* agent also quickly picks the correct views, thus we never lose much but often gain a lot. We also notice that *TES* achieves higher accumulated rewards than *loreRL* and *fEpsG* that are bound to make uninformed decisions in the beginning.

We also notice that due to its fast capability of capturing the world dynamics, *TES* running time is just slightly longer than *LWT*'s and *loreRL*'s, which do not perform extra work for view switching but need more time and data to learn the dynamics models.

5.1.3. *Jumpstart*

Table 2 shows the average cumulative reward after the first episode (the jumpstart effect) for each test task in the leave-one-out cross-validation. We observe that *TES* usually outperforms both the non-transfer and the *LWT* approach.



(a) Environment 1: 6×6 feet. (b) Environment 2: 5×5 feet. (c) Environment 3: 6×6 feet.

Fig. 5. Three different environments. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

5.1.4. Convergence

To study the asymptotic performance of *TES*, we compare with the oracle method *fRmax* which is known to converge to a (near) optimal policy. Notice that in this feature-rich domain, *fRmax* without the pre-defined DBN structure is just similar to *Rmax*. Therefore, we also compare with *Rmax*. For *Rmax*, the number of visits to any state before it is considered "known" is set to 5, and the exploration probability ϵ for known states starts to decrease from value 0.1.

Fig. 4c shows the accumulated rewards and their statistical dispersions over episodes. Average performance is reflected by the angles of the curves. As seen, *TES* can achieve a (near) optimal policy very fast and sustain its good performance over the long run. It is only gradually caught up by *fRmax* and *Rmax*. This suggests that *TES* can successfully learn a good library of views in heterogeneous environments and efficiently utilize those views in novel tasks.

5.2. Real robot navigation

The theoretical analyses presented above have shown the advantages of *loreRL* and *TES* over the state of the art model-based RL algorithms. We have also demonstrated the efficiency of our methods through a set of empirical evaluations in simulated domains. These results, though valuable, are obtained under assumptions that are favorable for our approach. In this section, we aim to further evaluate the proposed methods in a real robotic domain where we cannot expect the effects of actions to follow a logistic regression model.

5.2.1. Experiment set-up

Environments. Fig. 5 shows the three environments used in our case studies. They are designed so that the robot's actions would have different effects at different locations, and the environment surfaces are the main factors affecting the action effects. The surfaces are made of various materials such as beans, soil, hay, leaves, shells, paper board, and nylon Berber carpet. These materials have different physical effects on the objects moving on them. The slopes and obstacles on the surfaces also contribute to the different effects of the actions. In some areas, the surfaces may change because of the robot's actions. We restore the surfaces to the original conditions after every episode.

For a robot to efficiently plan its path in these environments, only a small set of features based on the slopes, the obstacles, and the materials of the surfaces in different areas of the environments are relevant. These features, however, are very hard to define. It is preferable to leave the robot to automatically select the relevant features from a large set of simple features. To test our approach, therefore, we simply draw green and blue marks on the surfaces. The robot is marked with two red marks. There are also a blue ball, and several death-marked spots in the environment. Numerous features can be derived from these artifacts. The robot will need to select a few features that may serve as proxies to the true factors that affect its action effects.

Environment 1 and Environment 3 are deliberately designed so that the robot should learn its *views* (transition model) based on the blue marks. The transition dynamics in these two environments are very similar. However, the two environments are also different (in terms of irrelevant features): the blue balls, the death places, and the green marks are at different locations. We will explain the features in more detail shortly. Environment 2 is very different from Environment 1 and Environment 3. It is designed so that the robot should learn its views based on the green marks instead of the blue ones.

We treat the environments as discrete MDPs. We discretize the Environment 2 into a state space consisting of 8×8 (x,y)-locations and 8 different orientations of a robot, which yields a state space of 512 states. Environment 1 and 3 are larger, so we discretize them into a state space consisting of 10×10 (x,y) locations and 8 different orientations of a robot. The environments are in two different sizes, 5×5 feet and 6×6 feet (Fig. 5).

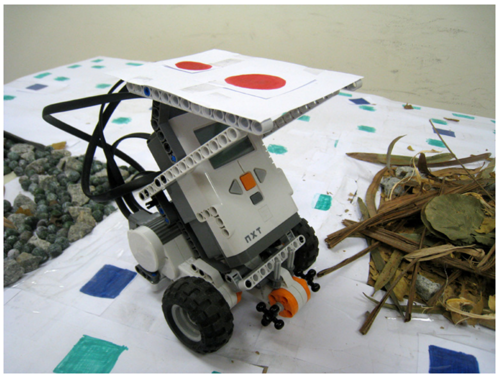


Fig. 6. The robot. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

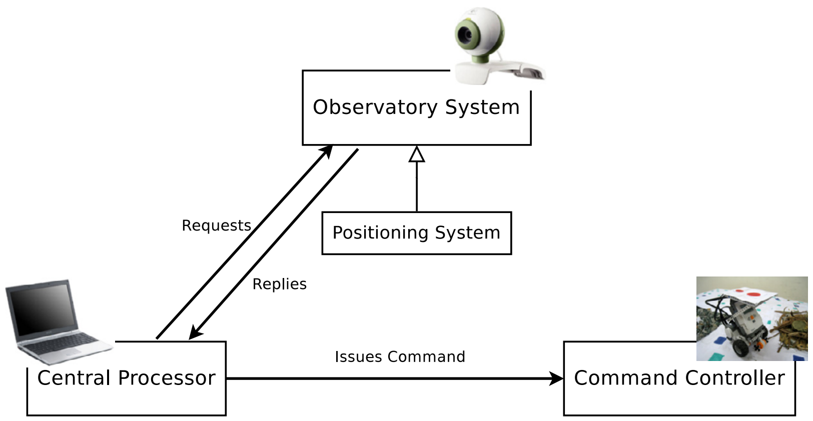


Fig. 7. The system architecture.

Robot. We use the LEGO Mindstorms NXT v1.1 kit to build a three-wheel robot as depicted in Fig. 6. Two front wheels of the robot are attached to two separate motors; the back wheel is free rolling. The track width is 11.2 cm. The robot carries a white panel on top with a big and a small red marks for positioning and orientation detection.

The robot system comprises three main components: a central processor, an observatory system, and a command controller (see Fig. 7). The positioning system is a sub-component of the observatory system. Information of the environment and the robot's position is captured by a webcam and sent from the *observatory system* to the *central processor* to update robot's knowledge-base as well as to plan the next action. The action command is then transmitted via Bluetooth to the *command controller* embedded in the robot for execution. We implement the controller in leJOS.[3]

Actions. The robot is programmed to rotate its left and right wheels in three different ways, corresponding to three actions. For the first action, the robot rotates both its left and right wheels 246 degrees. For the second action, the robot rotates its left wheel 90 and right wheel −90 degrees at the same time. For the third action, the robot rotates its left wheel −90 and right wheel 90 degree. As the robot may be still moving after each action, we let the system idle for 200 milliseconds after an action, waiting for the robot to stop completely. These actions, under ideal situations, correspond to the actions of move-forward, turn-left, and turn-right, respectively.

Action effects. Due to inaccurate robot motors, sensors, and various real world factors such as the surface materials, slopes of the surface, and obstacles, the actions may change the robot's relative location in four different ways, including moved forward one cell, moved diagonally forward to the cell on the robot's left, moved diagonally forward to the cell on the robot's right, and did not move. The robot's orientation can also be changed in five different ways, including: turned to the next orientation on left, the second next orientation on left, the next orientation on right, the second next orientation on right, and did not turn. That would result in a total of 20 different effects.

Sensors. We mainly process information from the web-cam in the observatory system. The web-cam is attached to the ceiling above the area. The robot, therefore, can fully observe the environment. However, the robot can only capture the big and small red marks on the top of the robot itself, and the information of the locations of the green, blue, red marks, and

[3] leJOS, Java for LEGO Mindstorms. http://lejos.sourceforge.net/.

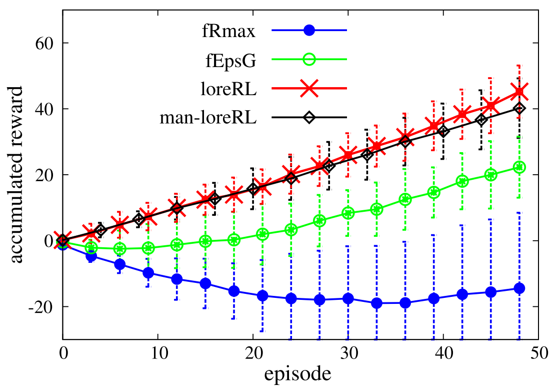


Fig. 8. Accumulated rewards by various methods.

Table 3
Average running time per episode in 50 episodes. Run on Intel Centrino Duo T2400 (1.83 GHz), 2 × 512 MB RAM.

Algorithm	*fRmax*	*fEpsG*	*man-loreRL*	***loreRL***
Time (sec.)	13.44	12.77	9.35	10.81

the ball in the environment. As the features are simple, we use the basic algorithms in OpenCV library[4] to detect them. The result is nearly perfect.

State attributes and features. As mentioned, an environment is discretized to n rows × m columns × o orientations, so the full environment state space can be identified or factorized by those three *state attributes*. However, these attributes alone do not contain enough information for predicting action effects or transition dynamics. Therefore, it is critical that each state is also defined with a long vector of binary *state features*. The "green" binary indicator $f_i^G(s)$ of a state s is set to 1 iff there is a green mark that is further than i units but closer than $i+1$ units from the (x, y)-center of the state s ($i \in \{0,\dots,99\}$). A unit equals the width of the environment divided by 100. Similar features are defined for blue marks and to the blue target ball yielding 300 binary features. Eight indicators for different robot orientations are also included in the feature-base together with four intentionally redundant "there is/is-not a green/blue mark in a state"-bits. All together these yield 312 binary features per state. The intuition behind these features is that they serve as proxies to surface materials, slopes on the surfaces, obstacles, etc. which appear to be important factors in determining the dynamics in the environments, but the robot's sensors cannot capture. Although only few among these 312 features are important for modeling robot's actions, the robot does not know which ones actually matter. *The robot has to learn to select the relevant features based on feedback while interacting with the environment*.

Task. The robot is assumed to know the reward model before any start. The robot's task is to travel in the environments from a random starting point to reach the *blue ball*, which will earn it a reward of 2 points. The robot will receive -1 point if it falls out of the area or into "death spots" marked with orange rectangles, and -0.05 points for reaching any other states. An episode ends if the robot reaches a terminal state or gets stuck, i.e., could not move, for four consecutive actions.

In other words, the robot aims for the highest cumulative reward in each episode. It tries to reach the blue ball as fast as possible, but it will avoid visiting the death spots or moving out of the map. In case it is very costly or impossible to reach the ball, the robot could give up by running into a death spot or moving out of the map.

5.2.2. *Online view learning for model-based RL*

The robot battery does not allow us to compare our algorithm with the slow *RL-DT* and *LSE-Rmax* algorithms, thus we will only compare *loreRL* with the fine-tuned algorithms, including *fRmax*, *fEpsG*, and *man-loreRL*, in which we manually select important features and specify the DBN-structures for the transition models. *man-loreRL* is based on multinomial logistic regression models with the 12 manually selected features, including eight indicators for different robot orientations, and four indicators telling if there is/is-not a green/blue mark in a state. We run the experiments with *loreRL* and *man-loreRL* setting $\alpha = 0.5$, $\lambda = 0.05$, $\gamma = 0.95$, exploration $\epsilon = 0.05$, and parameter $m = 10$ for *fRmax*. All results are averaged over 20 runs, and we report the 95% confidence intervals.

As shown in Fig. 8, *loreRL* appears to quickly capture the environment dynamics and outperform the other methods. Even with manually selected features, *fRmax* and *fEpsG* require more exploration to learn the dynamics. *man-loreRL* gains the rewards a bit faster, but in the end it loses to *loreRL* slightly, possibly due to the (unforeseen) insufficiency of the manually selected features. Table 3 further shows that *loreRL* is fast. Its average running time per episode with 312-features is only slightly slower than *man-loreRL* with 12 manually selected features.

[4] http://opencv.org/.

Table 4
Four robot testing scenarios.

Scenarios	Transferring knowledge	Experiences	Developing new views	Test on
1. *loreRL*	No	No	Yes	Env. 3
2. *LWT*	Yes	Env. 2	No	Env. 3
3. *TES-woRi*	Yes	Env. 2	Yes	Env. 3
4. *TES*	Yes	Env. 1 and Env. 2	Yes	Env. 3

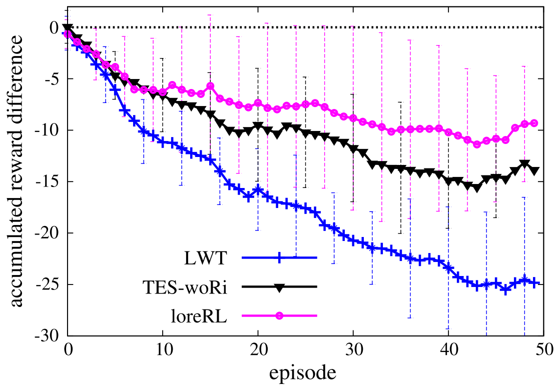


Fig. 9. Performance difference to *TES* in early trials in robotic domain.

The evaluations above have demonstrated that our new model-based RL, *loreRL* may work effectively in the real world. It may converge fast to a near optimal policy, achieving a high accumulated reward.

5.2.3. Transferring expectations in complex, heterogeneous environments

This case-study is designed to test if *TES* could effectively manage a good library of views to reduce the negative transfer effects and achieve better performance than the other methods. We compare the robot's performance in four scenarios as detailed below, and report the results for accumulated reward, jumpstart, and running time.

1. *loreRL*: the robot has no experience in Environment 1 and Environment 2. It runs and learns the views (transition model) directly on Environment 3.
2. *LWT*: the robot has first experienced Environment 2, and then uses that knowledge (transition model) to learn the action policy and run on Environment 3. The robot does not update its knowledge of the transition model in the new environment. We discuss *LWT* in more details in the related work section.
3. *TES-woRi*: the robot has first experienced Environment 2, and then runs on Environment 3. Different from the scenario 2, however, the robot here has the capability to adapt and develop new transition model for this novel Environment 3 if necessary. This *TES-woRi* robot is actually the *TES* robot, but we do not let it to experience the environment that is similar to the testing environment. This set-up evaluates *TES*'s capability to work with novel environments and to reduce negative transfer effects.
4. *TES*: the robot has first experienced Environment 1, and then Environment 2, before it runs on Environment 3. In this setting, we will see how effectively *TES* builds the library, and selects views from the library to solve a new task.

Table 4 summarizes and highlights the differences in these four scenarios.

We do not test the setting of letting the robot to experience with Environment 1 and running on Environment 3. This setting would allow us to see if *TES* can learn good views to transfer between similar Environment 1 and Environment 3. However, this effect can also be observed clearly by comparing *TES* and *TES-woRi*. If *TES* outperforms *TES-woRi*, it means that *TES* learned compact views of Environment 1 into the library.

Accumulated reward. Fig. 9 shows the differences in accumulated rewards of the robot with *LWT*, *TES-woRi*, and *loreRL* respectively to the robot with *TES*. As seen in the figure, the differences are all below 0, suggesting that *TES* could effectively transfer the view library in heterogeneous environments. *TES* could select the best models to approximate the world dynamics quickly, and outperform the other methods. It also suggests that *TES* with *mDAGL* has successfully learned compact action models, likely without redundant features, to the view library for transfer. For example, the robot has to work with 100 initial features related to the ball position, and the balls in Environment 1 and Environment 3 are at different positions, but the robot can still take advantage of the transferred knowledge. Checking the transferred model also confirms our hypothesis.

In addition, the data shows that *LWT* achieves the lowest accumulated reward, suggesting that *LWT* suffers from serious negative transfer effects, which usually appear when knowledge is transferred across heterogeneous environments. *TES-woRi*

Table 5
The robot cumulative rewards after the first episodes in 10 repeats.

Method	Repeat										
	Avg	1	2	3	4	5	6	7	8	9	10
loreRL	−0.35	0.4	−2.45	−0.9	0.65	−1	−0.8	0.7	0.25	0.5	−0.85
LWT	−0.26	1.45	−1.1	−0.55	−0.5	−0.45	1.4	−0.5	−2.4	1	−0.95
TES-woRi	**0.36**	−0.85	−0.5	1.3	1.1	−0.6	0.3	1.05	−0.75	1.45	1.1
TES	**0.31**	0.15	0.8	−0.4	0.95	1.3	0.8	0.75	0.85	−1.15	−0.95

Table 6
The robot's average running time per episode in 50 episode runs. The systems are run on an Intel Core Duo Processor T2400 (1.83 GHz) laptop with 2×512 MB RAM.

Method	*loreRL*	*LWT*	*TES-woRi*	*TES*
Time (sec.)	18.32	13.13	14.34	16.23

also suffers from the negative transfer effects, but due to its ability to quickly recognize the unsuitability of the transferred knowledge, it could adopt new action models to alleviate the negative effects. We see that *TES-woRi* loses to *loreRL* but not to a large extent.

Jumpstart. To examine the jumpstart, we organize the cumulative rewards after the first episodes in 10 repeats for all four algorithms into Table 5. In 10 repeating runs *TES* wins over *loreRL* 7 times, *LWT* 6 times (draw in 1 repeat), *TES-woRi* 5 times; *TES-woRi* wins over *loreRL* 8 times, *LWT* 7 times. We see that the *TES* methods have lower rewards than the others in several runs, and do not show statistically significant superior performance over the others. The results, however, are expected, because *TES* requires explorative interactions with the environment to select the appropriate views, but the starting locations in this test are quite close to the terminal "death" states. Nevertheless, it should be noted that *TES* methods still perform better than the others in the majority of the runs. In addition, on average they gain higher jumpstarts.

Running time. Table 6 shows the robot's running times per episode averaged over 50 episodes. A reason that *LWT* has the shortest average running time is that *LWT* transfers and uses only one model for each action. In addition, since *mDAGL* is used in the implementation of *LWT*, the transferred model is quite compact with only a few features and so can be learned quickly. Besides, the policy learned based on that "wrong" model appears to guide the robot to the terminal "death" states or to go out of the map, so the episodes run with *LWT* is often shorter than the others.

TES-woRi and *TES* have to manage a larger view library of 3×2 and 3×3 action models respectively, so their running times are longer. However it is interesting to note that they are not much slower than *LWT*, and even faster than *loreRL* which does not spend time to process transferred knowledge. That is probably because *TES* can save quite a lot of time from not having to plan an optimal policy with complex action models. The *TES* methods start the planning with compact and simple transferred action models, and switch to use fresh models later, if necessary, after the robot has accumulated some data to eliminate a large number of noncritical features in the model representations. Of course, another possible reason may be that the libraries in this experiment are relatively small.

6. Related work

Our work addresses two main challenges in transfer learning in RL: learning the transition (or reward) models, and transferring the relevant knowledge to perform new tasks in possibly different, dynamic environments. In this section, we compare our work with existing efforts in learning transition models and those that focus on transfer learning in RL.

6.1. Learning transition models

DBN has been a popular choice for factoring and approximating transition models. In DBN learning, feature selection is equivalent to selecting the parents of the state variables from the previous time slice. Recent studies have led to improvements in sample complexity for learning the optimal policy. Those studies assume maximum number of possible parents for a node [36,40], or knowledge of a planning horizon that satisfies certain conditions [38]. However, the improvements in sample complexity are achieved at the expense of computational complexity since these methods have to search through a large number of parent sets. Hence, these methods appear feasible only in manually designed, low-dimensional state-spaces.

Instead of searching for an optimal model with a minimal number of samples at almost any cost, our approach attempts to save costs from early on, and gradually improve the model acknowledging that the true model may actually be unattainable. In this spirit the structure learning study by Degris et al. [41] resembles our work, but they do not address online learning with large feature sets.

Ross et al. [42] use Markov chain Monte Carlo (MCMC) to sample from all possible DBN structures. However, the Markov Chain used has a very long burn-in period and slow mixing time, making sampling computationally prohibitive in large problems.

Robinson and Hartemink [43] apply a sampling method with conjugate priors to learn the optimal structure of non-stationary dynamic Bayesian networks by identifying relevant *move sets*, i.e., the edges that change over time. This approach, while scalable to hundreds or even thousands of variables, suffers from the same sample and computational complexity trade-off. Moreover, the work aims at pattern recognition instead of prediction, and does not address online learning.

Kroon and Whiteson [44] propose a feature selection method that works in conjunction with KWIK-Factored-Rmax to learn the structure and parameters of a factored MDP. This framework can extract a minimal set of features for representing the transition (and reward) model. Experiments show that planning on the reduced model yields improved performance. However, the computational cost of this method is still relatively high, since the KWIK-Factored-Rmax algorithm needs to search through a large combinatoric space of possible DBN structures to find the candidate structures for the proposed feature extraction method.

Leffler et al. [10] also suggest to predict relative changes in states, which corresponds to the action effects in our formulation. However, the important features are manually selected to aggregate information from similar states for action effect prediction, as compared with our focus on learning those features automatically.

Hester and Stone employ Quinlan's C4.5 [45] to learn a decision tree in *RL-DT* [37] to predict relative changes of every state variable. Despite adapting C4.5 for online learning, the method is still very slow as a costly tree induction procedure has to be repeated many times in a large feature space. In addition, all the data needs to be stored for the purpose, which is undesirable in some applications. TEXPLORE [9] is an extension of *RL-DT* by the same authors to integrate a special heuristic for exploration to the RL algorithm; the model learning part, however, remains the same as *RL-DT*. The exploration heuristic in TEXPLORE is quite general, and could be also incorporated into our algorithm.

Strehl and Littman [46], and Walsh et al. [47] propose an online linear regression method with L2-regularization to approximate the transition model in continuous MDP. L2, however, does not implement feature selection.

6.2. *Transfer learning in reinforcement learning*

The surveys by Taylor and Stone [12] and Lazaric [21] offer a comprehensive exposition of recent methods to transfer various forms of knowledge in RL. Most of these methods attempt to transfer structure and experiential knowledge in the forms of low level knowledge such as task instances, action-value pairs, full policies, full task models, prior distributions, or high level knowledge such as relevant action sets, partial policies, rules, relevant feature sets, or proto-value functions. In heterogeneous domains with different state-action spaces, inter-task transfers are often implemented through 1) task invariant representations that distinguish between the fixed sensor agent-space and the varying environmental problem-space [15] or 2) specific mapping criteria established through policy reuse [16], action correlation [17], state abstraction [18], inter-space relation [19], or other methods. Most of the task invariant representations and inter-task mappings have to be pre-defined, which renders these approaches difficult to be implemented in real-world tasks.

Not much research, however, has focused on compactly transferring represented transition or reward models in terms of action effects for online learning, without pre-defined invariant representations or mapping strategies in heterogeneous domains.

6.2.1. *Transferring a library of options*

While superficially similar to our framework, the case-based reasoning approaches [48,49] focus on collecting good decisions instead of building models of world dynamics. Skill learning methods [50,51] focus on building and using abstract actions in different tasks. In their recent work [52], Brunskill and Li analyze the bound on the sample complexity when abstract actions, or options, are introduced. Options are discovered using a greedy approach, but they are only adopted if they reduce or match the sample complexity for future RL tasks. This way negative transfer in the form of worse sample complexity can be avoided. While the empirical results are impressive, the source tasks and target tasks all share the same transition model and have a small state space. The empirical effectiveness in larger and heterogeneous domains is still to be investigated.

6.2.2. *Transferring multiple models*

Like us, Wilson et al. [53] aim at transferring knowledge in heterogeneous domains. They formalize the problem as learning a generative Dirichlet process for MDPs and suggest an approximate solution using Gibbs sampling. Our method can be seen as a structure learning enhanced alternative implementation of this generative model. Our online-method is computationally more efficient, but the MCMC estimation should eventually yield more accurate estimates. Both models can also be adjusted to deal with non-stationary task sources. Wilson et al., however, demonstrate the method for transferring reward models, and it is unclear how to extend the approach for transferring transition models.

Multiple models have previously been used to guide behavior in non-stationary environments [54,55]. Unlike our work, these studies usually assume a common concrete state space. In representation selection, Konidaris and Barto [51] focus on selecting the best abstraction to assist the agent's skill learning, and Van et al. [56] study using multiple representations together to solve a RL problem. Talvitie and Singh [57] and Fernandez et al. [16] both transfer a library of policies learned in previous tasks to bias exploration in new tasks. This method assumes a constant inter-task state space, otherwise a state mapping strategy is needed.

Lazaric and Restelli [58] suggest three different algorithms to transfer samples from multiple source tasks to solve a target task. While they are able to show many advantages of their approach with the offline learning fitted Q-iteration algorithm [59] on several experiments, a potential drawback is the need of a large number of samples in all the source tasks, and the strong assumptions on the similarity amongst the different tasks.

6.2.3. *Learning inter-task mappings for transfer*

Instead of keeping multiple models for transfer, Liu and Stone [60] learn a mapping function between actions and features in the source and the target tasks, but need to assume the availability of the structures of the QDBN in the pair of tasks. Taylor proposes TIMBREL [20] to transfer observations in a source to a target task via manually tailored inter-task mapping. This work is extended into MASTER [61] that attempts to automate the feature mapping. Similar to our view transferring approach, the MASTER algorithm automatically learns an efficient mapping through the task instances for transfer by leveraging a classification technique. However, the offline search strategy in MASTER to do one-to-one mapping from a source to a target task is costly in complex environments with large irrelevant feature sets; the trade-off between computational complexity and sample efficiency renders this approach difficult to apply in real-world tasks. Ammar et al. [62] most recently propose to use a special Restricted Boltzman Machine for supporting inter-task mapping. While this approach improves learning performance, the computational cost is still high, and many random samples in the source and target tasks are needed to effectively learn the mapping.

6.2.4. *Transferring knowledge about dynamics with feature selection*

Jong and Stone [63] propose hypothesis testing and Monte-Carlo simulation methods for detecting state variables that can be ignored without sacrificing the optimality of the policy. These methods require the task to be solved in order to learn such a state abstraction from Q-values. This differs substantially from loreRL which selects features online through mDAGL within one episode.

Calandriello et al. [64] recently propose interesting extensions to the fitted Q-iteration algorithm [59] based on sparse linear regression models. These extensions could enable an agent to efficiently share knowledge between tasks in heterogeneous domains to do feature selection, and to learn Q-value functions. Different from our setting, however, this multi-task RL approach aims to solve all the tasks together fast, and usually assumes to have access to sample data or models in all the tasks.

Among the very few efforts that actually consider transferring the transition model to a new task is the work [39] by Atkeson et al., which suggests a locally weighted transfer learning technique called *LWT* to adapt previously learned transition models into a new situation [39]. While their work is conducted in continuous state space using a fixed state similarity measure, it can be adapted to a discrete case. Doing so corresponds to adopting a fixed single view. This approach could also be extended to be compatible with our work by learning a library of state similarity measures and developing a method to choose among those similarities for each task.

7. Discussion

Transfer learning research in artificial intelligence is commonly divided into two subfields [14]: one focuses on the classification, regression, and clustering tasks involving different domains, the other one on reinforcement learning with different tasks and/or different domains. While focusing on RL problems, our work addresses a set of the most challenging open research issues in both subfields: transferring relevant representation of the world dynamics, captured in the transition models, with no or little available source data or target data. Our approach adopts a new, efficient online learning strategy that minimizes the risk of negative transfers in exploiting the learned knowledge and by exploring the environment at the same time. This strategy aims directly at addressing the difficulty of, and hence the relative little existing work in handling negative effects in transfer learning in both subfields [12,14].

Our online transfer learning framework is also in line with the few findings on mitigating negative effects through identifying common patterns or relationships through domain adaptation, and globally and locally weighted structure mapping between the source and target tasks and/or domains/environments [58,65,66]. Commonalities between the old and the new tasks and/or domains, which are very difficult to succinctly define in real environments, are captured through the scoring and updating of the views in the view library. While more work is needed to show how different types of negative transfer effects are mitigated, the experimental results show that effective and efficient transfers are possible in our system.

Our theoretical framework relies on some assumptions that are often questionable in real environments. For example, the proposed representation is formulated for discrete state and action spaces, while in real domains the robot usually perceives a multivariate continuous domain and its actuators take many continuous parameters, such as the speed, the rotation directions, and rotation duration of the motors. Some of these assumptions can be easily relaxed. The features of states can naturally be continuous for the logistic regression. The multivariate actions can also be lifted to the same level with features. This will allow different (dimensions of) actions to be used differently in different environments. While the logistic regression requires the effect of the action to be discrete, the states themselves can be continuous as long as the effects map the states to the next states with a well defined (deterministic) function. Alternatively, the effect should be modeled as continuous. In this case sparsity promoting priors can also be used to implement feature selection as in the discrete case. Naturally, changing the state space to be continuous will also require planning to be implemented in continuous space.

While mDAGL is fast, and lightweight, the transition model can be costly to build when there are many actions and with possibly numerous effects, as many regression models are required. In some problems, it would be possible to group actions and effects together, e.g., moving up, left, down, right can be considered as a group of moving actions. This will reduce the number of logistic regression models to build. Furthermore, since the action effect models are assumed to be independent, regression models can be learned in parallel. These extension would scale up mDAGL and TES for applications with a large action space.

One significant simplification we have used in our robotic studies is total observability. A camera has been attached to the ceiling so that the robot could have a full observation of the environment, and its own location in the environment. Also, the planning is based on the assumption that the world is static, so that features of the state will never change. In general there may be other agencies changing the world, and the next state cannot easily be expressed as a combination of a small set of effects and a deterministic next state function. We would need a method to learn the stochasticity as well as the uncertainty of the feature values given a state attribute set to address this issue in future.

In theory, RL requires the robot to try each action many times in many states before any guarantees about its good performance can be given. While this self guided learning liberates programmers from manually specifying behavior of the robot, engaging in such an exploration can be dangerous and even seriously damage the robot. In reality some estimate about the danger of explorative actions should be included into a model.

We have applied the more costly but simple value iteration for policy learning in our experiments with *loreRL* and *TES*, as we have focused on transfer learning in the RL framework. A more clever interleaving of model-building and policy formulation can be designed. For example, value iteration can be simply replaced by the Dyna-Q [34] or Prioritized Sweeping [35] algorithms. When and how to apply the planning steps can also be determined by RL. In case of discounted rewards, planning in large state spaces can be implemented with forward sampling methods.

In essence, the main practical limitations for this work are as follows: For a single rather static task, there will certainly be an unnecessary overhead, since the system always tries to build a new model instead of just trying to improve the old one. If the action effects are not well predicted by any logistic regression model, some other online classifier can be used, but building efficient online classifiers is not a trivial task. Generalizing to continuous actions and effects would also need extra work.

8. Conclusions

An intelligent, autonomous, and interactive agent or robot should learn to solve different tasks in different environments. An intelligent agent should focus attention on the most relevant task and environmental features for problem solving. An intelligent agent should also retain or store the knowledge learned in previous tasks, and “understands” if, how, and when the learned knowledge can be effectively used in new situations. Toward these objectives, we have presented a framework for learning and transferring multiple expectations or views about world dynamics in heterogeneous environments. Unlike most existing efforts, the proposed framework is scalable as it does not assume fixed actions to be taken or constant environmental features, nor does it require accurate mapping functions to be defined across different tasks or environments. The framework can work with very little or no data available in the tasks or environments, which is a major challenge in transfer learning in both static and dynamic domains.

When the environments are different, the combination of learning multiple views and dynamically selecting the most promising ones yields a system that can learn a good policy faster and gain higher accumulated reward as compared to the common strategy of learning just a single good model and using it on all occasions.

Applying and maintaining multiple models require additional computation and memory. We have shown that by a clever decomposition of the transition function, model selection and model updating can be accomplished efficiently using online algorithms. Our experiments have shown that performance improvements in multi-dimensional, heterogeneous environments can be achieved with a small computational cost.

In addition, we have demonstrated how online multinomial logistic regression with group lasso can be used to quickly obtain a parsimonious transition model in model based RL. The method leads to fast learning since a single transition model can be learned using samples from all the states with a small set of features. The efficiency is gained, however, at the expense of losing generality. Not all transition functions can be accurately represented as predicting action effects using state features via logistic regression. Nevertheless, we believe that this compromise between scalability and generality is often a useful one. The generality problem may also be alleviated by introducing non-linear features that are combinations of the original ones. Other generalizations such as stochastic features and vector valued effects are also possible but are left for future work.

The current work addresses the question of learning good models, but the problem of learning good policies in large state spaces still remains. Our model learning method is independent of the policy learning task, thus it can well be coupled with any scalable approximate policy learning algorithms.

There are still many open problems and assumptions to address, yet this study has provided some insights for future research and development. The proposed transfer learning framework would serve as a base intelligent agent platform for monitoring, problem solving, and general decision support in heterogeneous, dynamic environments. We envision that this platform can be integrated with the latest agent-based or environmental sensors, navigation and localization functions, and higher level cognitive capabilities such as activity recognition, speech understanding, and decision making to develop a new generation of human-interactive robots.

Acknowledgements

We thank Dr. Haiqin Yang for helpful comments on the sparse multinomial logistic regression approach in an earlier draft of this work. This research is supported by Academic Research Grants: MOE2010-T2-2-071, T1 251RES1005, and T1 251RES1211 from the Ministry of Education in Singapore.

Appendix A. Update rule for mDAGL

Theorem 2 *(Update rule). Given $h(W) = \frac{1}{2}\|W\|_2^2$, a $K \times d$ average gradient matrix $\bar{G}^t$, and a regularization parameter $\lambda > 0$, the optimal solution of (3) is achieved column-wise as follows*

$$W_{\cdot i}^{t+1} = \begin{cases} \vec{0} & \textit{if } \|\bar{G}_{\cdot i}^t\|_2 \le \lambda\sqrt{K}, \\ \frac{t}{\beta_t}\left(\frac{\lambda\sqrt{K}}{\|\bar{G}_{\cdot i}^t\|_2} - 1\right)\bar{G}_{\cdot i}^t & \textit{otherwise}. \end{cases} \tag{A.1}$$

Proof sketch. For reading clarity, we delay the full proof to the next appendix, and only provide a sketch here. Since the minimization Problem 3 is component-wise on one column of W, we can focus on each of the column of W separately to find its solution. Because inner product of two vectors of same length will have smallest value when the two vectors are in opposite direction, the solution to each of the component-wise minimization problem should be a factor of φ ($\varphi \le 0$) to the corresponding column in the average gradient matrix. Subsequently, we can turn the problem into a basic quadratic function minimization problem. □

Proof. Let us rewrite the minimization problem,

$$W^{t+1} = \arg\min_W(\langle \bar{G}^t, W\rangle + \lambda\sqrt{K}\sum_i^d \|W_{\cdot i}\|_2 + \frac{\beta_t}{2t}\sum_i^d \|W_{\cdot i}\|_2^2)$$

Since the minimization problem is component-wise on one column of W, we can focus on each of the column of W separately to find its solution.

$$W_{\cdot i}^{t+1} = \arg\min_{W_{\cdot i}}\left(\langle \bar{G}_{\cdot i}^t, W_{\cdot i}\rangle + \lambda\sqrt{K}\|W_{\cdot i}\|_2 + \frac{\beta}{2t}\|W_{\cdot i}\|_2^2\right)$$

Since inner product of 2 vectors of same length will have smallest value when the 2 vectors are in opposite direction, solution to the above minimization problem should be $W_{\cdot i}^{t+1} = \varphi\bar{G}_{\cdot i}^t$ where $\varphi \le 0$.

We now need to solve the following minimization problem,

$$\varphi = \arg\min_{\varphi\le 0}\left(\varphi\|\bar{G}_{\cdot i}^t\|_2^2 - \varphi\lambda\sqrt{K}\|\bar{G}_{\cdot i}^t\|_2 + \varphi^2\frac{\beta}{2t}\|\bar{G}_{\cdot i}^t\|_2^2\right)$$

Solving for the minimum point of that familiar quadratic function, we have

$$\varphi = \begin{cases} \frac{t}{\beta_t}\left(\frac{\lambda\sqrt{K}}{\|\bar{G}_{\cdot i}^t\|_2} - 1\right) & \text{if } \|\bar{G}_{\cdot i}^t\|_2 > \lambda\sqrt{K}, \\ 0 & \text{otherwise}. \end{cases}$$

Therefore, the update rule is as in Theorem 1. □

Appendix B. Regret analysis of mDAGL

Theorem 3 *(Regret bound). Let the sequence of $\{W^t\}_{t\ge 1}$ be generated by the update rule (A.1), and assume that there exists a constant G such that $\|G^t\|_2^2 \le G^2, \forall t \ge 1$. If we choose $\beta_t = \alpha\sqrt{t}$ where $\alpha > 0$, then for any $t \ge 1$ and for any W that satisfies $h(W) \le D^2$ where D is a constant, the average regret is bounded as*

$$\frac{R_t(W)}{t} \le \frac{\Delta}{\sqrt{t}}, \quad t = 1, 2, 3.., \tag{B.1}$$

where $\Delta = \left(\alpha D^2 + \frac{G^2}{\alpha}\right)$.

Proof sketch. The item-wise loss function $l_t(W)$ of multinomial logistic regression is convex, thus the techniques used for binary case [26] can be applied for multinomial case as well. □

Since the average regret goes asymptotically to zero, it may look very feasible that the sequence (W^t) also converges to some optimal W^*. However, the regret analysis is valid for any sequence of data, and without additional assumptions about the data generating process there may not be any asymptotically optimal classifier W^*, thus convergence is not meaningful. To study convergence, we assume the data is to be sampled independently from some joint distribution p for data vector (y,x). In this case we try to find a W that minimizes the expected loss $E_p[l(W)] + \Psi(W)$. Now assuming that the optimal solution W^* is sparse, and some other technical assumptions, it is indeed possible to show that

$$P(\|W^t - W^*\|_2 > \epsilon) < \left[\epsilon^{-1}(\epsilon^{-1} + r^{-1}) + \frac{2}{c}\Delta\right] t^{-\frac{1}{4}}, \tag{B.2}$$

where r and c are constants (see Lemma 13 in [67] for the result and its assumptions).

Appendix C. Proof of Theorem 1 (difference in value function)

Proof. For any action a, consider the following expression, where x is a vector of all state attributes and features extracted from state s and e is the action effect leading to s' from s.

$$\begin{aligned}
&\sum_{s'\in S} P^{M_1}(s'|s)\log\left(\frac{P^{M_1}(s'|s)}{P^{M_2}(s'|s)}\right)\\
&= \sum_{e\in E} P^{M_1}(e|s)\log\left(\frac{P^{M_1}(e|s)}{P^{M_2}(e|s)}\right)\\
&\le \max_{e\in E}\log\left(\frac{P^{M_1}(e|s)}{P^{M_2}(e|s)}\right)\\
&= \max_{e\in E}\log\left(\frac{\exp(W_e^{M_1}x)/\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\exp(W_e^{M_2}x)/\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right)\\
&= \max_{e\in E}\left[\log\left(\frac{\exp(W_e^{M_1}x)}{\exp(W_e^{M_2}x)}\right) - \log\left(\frac{\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right)\right]\\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \log\left(\frac{\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right)\right]\\
&\le \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \log\left(\min_{e'\in E}\left(\frac{\exp(W_{e'}^{M_1}x)}{\exp(W_{e'}^{M_2}x)}\right)\right)\right]\\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \min_{e'\in E}\left(\log\left(\frac{\exp(W_{e'}^{M_1}x)}{\exp(W_{e'}^{M_2}x)}\right)\right)\right]\\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \min_{e'\in E}\left((W_{e'}^{M_1} - W_{e'}^{M_2})x\right)\right]\\
&\le \max_{e\in E}[\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1 + \max_{e'\in E}(\|W_{e'}^{M_1} - W_{e'}^{M_2}\|_1 \sup_s \|x(s)\|_1)]\\
&\le 2\max_{e\in E}(\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1)
\end{aligned}$$

The first step is from definition of *effect*. The second step is from the fact that weighted average of elements must be smaller than the largest one. The sixth step is from the property that if a_i and b_i are non-negative, then $\left(\sum_i a_i\right)/\left(\sum_i b_i\right) \ge \min_i(a_i/b_i)$. The seventh step is from monotonicity of logarithmic function.

By Pinsker's inequality,

$$\begin{aligned}
&\sum_{s'\in S} P^{M_1}(s'|s)\log\left(\frac{P^{M_1}(s'|s)}{P^{M_2}(s'|s)}\right)\\
&\ge \frac{1}{2}(\sum_{s'\in S}|P^{M_1}(s'|s) - P^{M_2}(s'|s)|)^2
\end{aligned}$$

which implies

$$\sum_{s' \in S} |P^{M_1}(s'|s) - P^{M_2}(s'|s)|$$

$$\leq 2\sqrt{\max_{e \in E}(\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1)}$$

Extending to all actions,

$$\sum_{s' \in S} |P^{M_1}(s'|s) - P^{M_2}(s'|s)|$$

$$\leq \max_{a \in A}\left(2\sqrt{\max_{e \in E}(\|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1)}\right)$$

$$= 2\sqrt{\max_{a \in A, e \in E}(\|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1)}$$

To complete the theorem, the following lemma (see Lemma 33 in [68]) is used without proof.

Lemma 1. *Let $M_1 = (S, A, P^{M_1}, R)$, $M_2 = (S, A, P^{M_2}, R)$ be two MDPs, and fixed discount factor γ. π_1 and π_2 are their optimal policies respectively. Let V_π^M be the value function of π in MDP M. If*

$$\sum_{s' \in S} |P^{M_1} - P^{M_2}|(s'|s, a) \leq \epsilon$$

for every state-action (s, a), then $|V_{\pi_2}^{M_1}(s) - V_{\pi_2}^{M_2}(s)| \leq \frac{\gamma V_{max}\epsilon}{1-\gamma}$ and $|V_{\pi_1}^{M_2}(s) - V_{\pi_1}^{M_1}(s)| \leq \frac{\gamma V_{max}\epsilon}{1-\gamma}$, for every $s \in S$.

It is clear that

$$\max_{s \in S}\left(V_{\pi_2}^{M_2} - V_{\pi_1}^{M_2}\right)$$

$$= \max_{s \in S}\left(V_{\pi_2}^{M_2} - V_{\pi_1}^{M_1} + V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2}\right)$$

$$\leq \max_{s \in S}\left(V_{\pi_2}^{M_2} - V_{\pi_2}^{M_1} + V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2}\right)$$

$$\leq \max_{s \in S}|V_{\pi_2}^{M_2} - V_{\pi_2}^{M_1}| + \max_{s \in S}|V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2}|$$

$$\leq \frac{2\gamma V_{max}\epsilon}{1-\gamma}.$$

The proof is therefore complete. □

References

[1] B. Boots, N. Hawes, T. Hester, G. Konidaris, T. Mericli, L. Riano, B. Rosman, P. Stone (Eds.), AAAI 2013 Workshop on Intelligent Robotic Systems, 2013.
[2] National Science Foundation, National robotics initiative, http://www.nsf.gov/pubs/2012/nsf12607/nsf12607.htm, accessed on 06 September 2013.
[3] U.S. DARPA robotics challenge, http://www.theroboticschallenge.org, accessed on 06 September 2013.
[4] N. Kroes, Robots and other cognitive systems: challenges and European responses, Philos. Technol. 24 (2011) 355–357.
[5] E. Guizzo, T. Deyle, Robotics trends for 2012, IEEE Robot. Autom. Mag. (2012) 119–123.
[6] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[7] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: a survey, Int. J. Robot. Res. 32 (2013) 1238–1274.
[8] A.Y. Ng, H.J. Kim, M.I. Jordan, S. Sastry, Autonomous helicopter flight via reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '03, 2003.
[9] T. Hester, P. Stone, TEXPLORE: real-time sample-efficient reinforcement learning for robots, Mach. Learn. 90 (3) (2013) 385–429.
[10] B.R. Leffler, M.L. Littman, T. Edmunds, Efficient reinforcement learning with relocatable action models, in: Proceedings of the National Conference on Artificial Intelligence, AAAI '07, 2007.
[11] T.A. Estlin, B.J. Bornstein, D.M. Gaines, R.C. Anderson, D.R. Thompson, M. Burl, R. Castaño, M. Judd, AEGIS automated science targeting for the MER opportunity rover, ACM Trans. Intell. Syst. Technol. 3 (2012) 50:1–50:19.
[12] M.E. Taylor, P. Stone, Transfer learning for reinforcement learning domains: a survey, J. Mach. Learn. Res. 10 (2009) 1633–1685.
[13] Q. Silver, D. Yang, L. Li, Lifelong machine learning systems: beyond learning algorithms, in: Proceedings of the AAAI Spring Symposium on Lifelong Machine Learning, Stanford University, CA, 2013.
[14] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[15] G. Konidaris, A.G. Barto, Building portable options: skill transfer in reinforcement learning, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '07, 2007.
[16] F. Fernández, J. García, M. Veloso, Probabilistic policy reuse for inter-task transfer learning, J. Robot. Auton. Syst. 58 (2010) 866–871.

[17] A.A. Sherstov, P. Stone, Improving action selection in MDP's via knowledge transfer, in: Proceedings of the National Conference on Artificial Intelligence, AAAI '05, 2005.
[18] T.J. Walsh, L. Li, M.L. Littman, Transferring state abstractions between MDPs, in: ICML Workshop on Structural Knowledge Transfer for Machine Learning, 2006.
[19] V. Soni, S. Singh, Using homomorphisms to transfer options across continuous reinforcement learning domains, in: Proceedings of the National Conference on Artificial Intelligence, AAAI '06, 2006.
[20] M.E. Taylor, N.K. Jong, P. Stone, Transferring instances for model-based reinforcement learning, in: Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 5212, Springer-Verlag, 2008, pp. 488–505.
[21] A. Lazaric, Transfer in reinforcement learning: a framework and a survey, in: M. Wiering, M. van Otterlo (Eds.), Reinforcement Learning: State of the Art, Springer, 2011, pp. 143–173, Ch. 5.
[22] T.T. Nguyen, T. Silander, T.Y. Leong, Transfer learning as representation selection, in: Working Notes of the International Conference on Machine Learning Workshop on Representation Learning, 2012.
[23] T.T. Nguyen, T. Silander, T.Y. Leong, Transferring expectations in model-based reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '12, 2012.
[24] T.T. Nguyen, Z. Li, T. Silander, T.Y. Leong, Online feature selection for model-based reinforcement learning, in: Proceedings of the International Conference on Machine Learning, ICML '13, 2013.
[25] G. Konidaris, A.G. Barto, Autonomous shaping: knowledge transfer in reinforcement learning, in: Proceedings of the International Conference on Machine Learning, ICML '06, 2006.
[26] L. Xiao, Dual averaging methods for regularized stochastic learning and online optimization, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '09, 2009.
[27] H. Yang, Z. Xu, I. King, M.R. Lyu, Online learning for group lasso, in: Proceedings of the International Conference on Machine Learning, ICML '10, 2010.
[28] C. Boutilier, R. Dearden, M. Goldszmidt, Stochastic dynamic programming with factored representations, Artif. Intell. 121 (1–2) (2001) 49–107.
[29] J. McCarthy, Situations, actions, and causal laws, Tech. rep. Memo 2, Stanford Artificial Intelligence Project, Stanford University, 1963.
[30] L.J. Savage, Elicitation of personal probabilities and expectations, J. Am. Stat. Assoc. 66 (336) (1971) 783–801.
[31] A. Dawid, Statistical theory: the prequential approach, J. R. Stat. Soc. A 147 (2) (1984) 278–292.
[32] X. Zhu, Z. Ghahramani, J. Lafferty, Time-sensitive Dirichlet process mixture models, Tech. rep. CMU-CALD-05-104, School of Computer Science, Carnegie Mellon University, 2005.
[33] M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables, J. R. Stat. Soc., Ser. B, Stat. Methodol. 68 (2006) 49–67.
[34] R.S. Sutton, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, in: Proceedings of the International Conference on Machine Learning, ICML '90, 1990, pp. 216–224.
[35] A.W. Moore, C.G. Atkeson, Prioritized sweeping: reinforcement learning with less data and less time, J. Mach. Learn. 13 (1993) 103–130.
[36] A.L. Strehl, C. Diuk, M.L. Littman, Efficient structure learning in factored-state MDPs, in: Proceedings of the National Conference on Artificial Intelligence, AAAI '07, 2007.
[37] T. Hester, P. Stone, Generalized model learning for reinforcement learning in factored domains, in: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, AAMAS '09, 2009.
[38] D. Chakraborty, P. Stone, Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree, in: Proceedings of the International Conference on Machine Learning, ICML '11, 2011.
[39] C.G. Atkeson, A.W. Moore, S. Schaal, Locally weighted learning, J. Artif. Intell. Rev. 11 (1997) 11–73.
[40] C. Diuk, L. Li, B.R. Leffler, The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning, in: Proceedings of the International Conference on Machine Learning, ICML '09, 2009.
[41] T. Degris, O. Sigaud, P.-H. Wuillemin, Learning the structure of factored Markov decision processes in reinforcement learning problems, in: Proceedings of the International Conference on Machine Learning, ICML '06, 2006.
[42] S. Ross, J. Pineau, Model-based Bayesian reinforcement learning in large structured domains, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI '08, 2008.
[43] J.W. Robinson, A.J. Hartemink, Learning non-stationary dynamic Bayesian networks, J. Mach. Learn. Res. 11 (2010) 3647–3680.
[44] M. Kroon, S. Whiteson, Automatic feature selection for model-based reinforcement learning in factored MDPs, in: Proceedings of the International Conference on Machine Learning and Applications, ICMLA '09, 2009.
[45] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
[46] A.L. Strehl, M.L. Littman, Online linear regression and its application to model-based reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '07, 2007.
[47] T.J. Walsh, I. Szita, C. Diuk, M.L. Littman, Exploring compact reinforcement-learning representations with linear regression, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI '09, 2009.
[48] J.L.A. Celiberto, J.P. Matsuura, R.L. De Mantaras, R.A.C. Bianchi, Using cases as heuristics in reinforcement learning: a transfer learning application, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '11, 2011.
[49] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. Isbell, A. Ram, Transfer learning in real-time strategy games using hybrid CBR/RL, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '07, 2007.
[50] B.C. da Silva, G. Konidaris, A.G. Barto, Learning parameterized skills, in: Proceedings of the International Conference on Machine Learning, ICML '12, 2012.
[51] G. Konidaris, A.G. Barto, Efficient skill learning using abstraction selection, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '09, 2009.
[52] E. Brunskill, L. Li, PAC-inspired option discovery in lifelong reinforcement learning, in: Proceedings of the International Conference on Machine Learning, ICML '14, 2014.
[53] A. Wilson, A. Fern, S. Ray, P. Tadepalli, Multi-task reinforcement learning: a hierarchical Bayesian approach, in: Proceedings of the International Conference on Machine Learning, ICML '07, 2007.
[54] K. Doya, K. Samejima, K. Katagiri, M. Kawato, Multiple model-based reinforcement learning, J. Neural Comput. 14 (2002) 1347–1369.
[55] B.C. da Silva, E.W. Basso, A.L.C. Bazzan, P.M. Engel, Dealing with non-stationary environments using context detection, in: Proceedings of the International Conference on Machine Learning, ICML '06, 2006.
[56] H. Van Seijen, B. Bakker, L. Kester, Switching between different state representations in reinforcement learning, in: Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, AIA '08, 2008.
[57] E. Talvitie, S. Singh, An experts algorithm for transfer learning, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '07, 2007.
[58] A. Lazaric, M. Restelli, Transfer from multiple MDPs, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '11, 2011.
[59] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, J. Mach. Learn. Res. 6 (2005) 503–556.
[60] Y. Liu, P. Stone, Value-function-based transfer for reinforcement learning using structure mapping, in: Proceedings of the National Conference on Artificial Intelligence, AAAI '06, 2006.

[61] M.E. Taylor, G. Kuhlmann, P. Stone, Autonomous transfer for reinforcement learning, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '08, 2008.

[62] H. Ammar, D. Mocanu, M. Taylor, K. Driessens, K. Tuyls, G. Weiss, Automatically mapped transfer between reinforcement learning tasks via three-way restricted Boltzmann machines, in: Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, vol. 8189, 2013, pp. 449–464.

[63] N.K. Jong, P. Stone, State abstraction discovery from irrelevant state variables, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI '05, 2005.

[64] D. Calandriello, A. Lazaric, M. Restelli, Sparse multi-task reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems, NIPS '14, 2014.

[65] R. Chattopadhyay, Q. Sun, W. Fan, I. Davidson, S. Panchanathan, J. Ye, Multisource domain adaptation and its application to early detection of fatigue, ACM Trans. Knowl. Discov. Data 6 (4) (2012) 18:1–18:26.

[66] L. Ge, J. Gao, H. Ngo, K. Li, A. Zhang, On handling negative transfer and imbalanced distributions in multiple source transfer learning, in: Proceedings of the SIAM International Conference on Data Mining, 2013.

[67] S. Lee, S. Wright, Manifold identification of dual averaging methods for regularized stochastic online learning, J. Mach. Learn. Res. 13 (2012) 1705–1744.

[68] L. Li, A unifying framework for computational reinforcement learning theory, Ph.D. thesis, Rutgers, The State of University of New Jersey, 2009.