Learning in Non-Stationary MDPs as Transfer Learning

M. M. Hassan Mahmud
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
hmahmud42@gmail.com

Subramanian Ramamoorthy
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
s.ramamoorthy@ed.ac.uk

ABSTRACT

In this paper we present a learning algorithm for a particular subclass of non-stationary environments where the learner is required to interact with other agents. The behavior-policy of the agents are determined by a latent variable that changes rarely, but can modify the agent policies drastically when it does change (like traffic conditions in a driving problem). This unpredictable change in the latent variable results in non-stationarity. We frame this problem as a transfer learning in a particular subclass of MDPs where each task/MDP requires the learner to learn to interact with opponent agents with fixed policies. Across the tasks, the state and action space remains the same (and is known) but the agent-policies change. We transfer information from previous tasks to quickly infer the combined agent behavior policy in a new task after some limited initial exploration, and hence rapidly learn an optimal/near-optimal policy. We propose a transfer algorithm which given a collection of source behavior policies, eliminates the policies that do not apply in the new task in time polynomial in the relevant parameters using novel a statistical test. We also perform experiments in three interesting domains and show that our algorithm significantly outperforms relevant algorithms.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Intelligent Agents

General Terms

Algorithms

Keywords

Single agent Learning,Planning, Agent theories, Modeling other agents and self, Computational architectures for learning

1. INTRODUCTION

In this paper we introduce an algorithm for learning in non-stationary environments where the non-stationarity arise from the actions of other agents whose behavior policy may change over time. The behavior-policies are determined by a latent variable which does not change often, but when it does it may drastically alter the policies of the agents. That is, each value of the variable defines a regime, and the change in value causes a change in the regime. Domains that require interacting with robots or other agents who operate in accordance with distinct behavior-profiles are examples of tasks that fit into our formulation. For instance, in a driving problem, the behavior-profile of other drivers will depend on the latent variable 'traffic conditions'. Our algorithm uses standard and novel statistical tests to quickly determine whether any of the previous profiles are being used in the current task, and if so uses that fact to compute an optimal policy for the new task. We give sample complexity bounds for the algorithm to eliminate incorrect profiles when we use our novel statistical test and then demonstrate its efficacy in experiments.

Our approach is to model the above problem as transfer learning for reinforcement learning (TLRL) in MDPs where each MDP represents a particular regime. In TLRL we try to use solutions of MDPs solved at a prior point in time (the *source* MDPs) to solve a new but related MDP (the *target* MDP) much faster (see [11],[15] for an introduction reinforcement learning in MDPs and [16] for a survey on TLRL). Hence, in our case we use agent behavior in source MDPs (previous regimes) to infer their behavior in the target MDP (current regime).

We call a MDP that has this additional structure in the form of existence of K opponent agents with fixed policies a MDP-with-agents, (MDP-A in short form). Each MDP-A is defined like a MDP but now the transition and reward functions are dependent on the actions of the other agents. The policy of each opponent agent is fully determined by the state and action of the learner, which implies that the model is actually an MDP. In the transfer learning scenario, all the tasks in the sequence are defined over the same state and action space (both learner and other agent), and the only thing that varies are the policies of the agents, which we collectively refer to as the type of the task. We assume that these state and transition distributions are known and the only unknowns are the types. We derive an algorithm, which we call Type-Elimination, that based on some initial exploration of the target task quickly determines if the type of the target task is the same as the type of any of the source task. If so, then the source task may be used to compute an optimal policy for the target task and we get an optimal policy very quickly.

As a simple illustrative example, consider the following congestion example that we use in our experiments later on. Due to the plan of the city and movement of the residents of the city, the traffic may follow a predictable pattern, where a certain zone free of traffic implies that a certain other zone has a traffic jam. The type of traffic can be modeled as the behavior-policies of other agents, and hence the traffic pattern corresponds to a particular type. Furthermore, given that one zone is free of traffic, we may predict that a different zone has traffic, and use this information to plan more effectively.

We have briefly introduced our model and we now proceed as

Technical Report: *University of Edinburgh, 2013*

follows. In section 2, we discuss related work. Then we present the preliminaries for this paper in Section 3. After that we introduce our model and then we introduce our learning algorithm in Section 4. After that we describe three different domains and perform experiments on them in Section 5. After that we end with a conclusion in Section 6.

2. RELATED WORK

Our algorithm deals with environments that involve other agents. Typically, Markov games are used to model problems where the learner has to deal with other players. However, in Markov games the other agents may change their policies within a particular task whereas in our case the policy remains constant in a single task, but changes across different tasks. The fact that the policy remains fixed in a given task implies that each task in our model is indeed an MDP (rather than a Markov game). For exactly this reason, a MDP-A is different from I-POMDPs and other extension of MDPs that allow presence of other agents. While our problem can be posed in framework of the above models, which are more general and allow for intricate strategic interaction, we do not use them because learning (and even planning) in these models can be hard to intractable. An MDP, on the other hand, is polynomially (approximately) learnable [7], and is adequate for capturing the type of strategic interaction that we wish to address.

As mentioned above, we use the transfer learning formulation to address learning in a non-stationary environment. Unsurprisingly, it is also possible to frame the above problem as reinforcement learning in a particular type of non-stationary MDPs [10], [9], [18]. A non-stationary MDP is one where the transtion and reward function may change with time. Hence in our case, we can view the sequence of MDP-As as a single MDP where the reward and transition function change as the MDP-A changes. The previous methods for non-stationary MDPs do not quite apply in our setting. The papers [10] and [9] consider, respectively, planning and learning when the state and reward distribution changes are a-priori bounded, whereas in our case we make no such assumption.

The paper [18] on the other hand does assume that the reward changes arbitrarily but the state function changes remain within some bound (both change possibly adversarially and at every step). In this setting, the authors design their algorithms to not do much worse than the best policy in hindsight for *all* the previous tasks. This criteria is not appropriate for our case because we want the best policy for the current task we are solving. If the previous tasks are useful for this purpose, we would like to take advantage of it, but if not, we still want to be optimal in the current task. Correspondingly, the two algorithms, ORDP and Q-FPL and not appropriate for our setting, and in fact will fare poorly against algorithms considered in this paper (and of course, such a comparison would be unfair because they were designed for completely adversarial problems).

More related is the paper by [3] who consider MDPS with finite number of hidden 'modes' that modify the MDP parameters. They pose this problem as a POMDP and learn to solve it. While these modes are identical to our notion of regimes, we do not assume that the number of regimes is bounded. Furthermore, our solution method is within the framework of MDPs, thus endowing us the efficiency of learning algorithms in these models. Another work in a similar vein is [13]. However, this paper considers slow, gradual changes in parameters rather than the infrequent but drastic change that we are interested in.

It is interesting to contrast our general approach to handling multiple agents and non-stationarity in MDPs, both very difficult problems, with those of the models and algorithms discussed above. Instead of looking at general models that capture many possible agent interactions, or deriving universal algorithms that work everywhere with a certain worst case guarantee, we focus on deriving effective and efficient algorithms for MDPs with specific structures and justify those structures through applications.

In the context of transfer learning in reinforcement learning, the MDP-A model is novel. We now only discuss the works that are most relevant to us, and point the interested reader to [16] for a survey. The MDP-A model addresses the case where the task changes infrequently, but possibly drastically. Another work which address slow change in the task description is [12]. In this, the authors present a algorithm that uses decision trees to represent agent policies and incrementally adapts it (by pruning or extending the tree) to accommodate the changing environment. The main difference is that they address incremental change and we address drastic change and across tasks.

In terms of previous work on TLRL the algorithm that most applies to our model is Policy-Reuse [6], [5]. This is a heuristic algorithm that , given K different previous policies for a given domain, tries to determine the best way to combine and use them. In our case, for each source MDP, we get a policy for the target MDP under the assumption that the behavior profile in that source task is same as in the target MDP. The above algorithm can then be used to choose between these policies. In fact, in our experiments, we compare our algorithm against the Policy-Reuse algorithm. We show that in addition to having theoretical guarnatees, our algorithm can take advantage of the structure of MDP-As to rapidly (often within a single episode) reject the incorrect previous tasks. While for Policy Reuse, this may take a long time.

3. PRELIMINARIES

We use $\triangleq$ for definitions, Pr to denote probability and $\mathbb{E}$ for expectation. Given a sequence of observations of random variable X, the empirical estimate of $\mathbb{E}[X]$ is the average of all the observed values. Similarly, for a distribution D, its empirical estimate is the empirical probability constructed from $x_{1:t}$ observations drawn from D. We denote this distribution by $P_{x_{1:t}}$. The rest of this section is devoted to defining Markov decision processes and the value functions of their policies.

For an introduction to reinforcement learning using MDPs, see [11] [15]. A finite MDP $\mathcal{M}$ is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, R, \gamma)$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions and $\mathcal{R} = [l, u] \subset \mathbb{R}$ is the set of rewards. $P(s'|s, a)$ is a the state transition distribution for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ while and $R(s, a)$, the reward function, is a random variable taking values in $\mathcal{R}$. Finally, $\gamma \in [0, 1)$ is the discount rate.

A (stationary) policy π for $\mathcal{M}$ is a map $\pi : \mathcal{S} \to \mathcal{A}$. A policy π For a policy π, the Q function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of the policy is given by:

$$Q^\pi(s, a) = \mathbb{E}[R(s, a)] + \gamma \sum_{s'} P(s'|s, a) Q^\pi(s', \pi(s'))$$

The value function for π is defined as $V^\pi(s) = Q^\pi(s, \pi(s))$. An optimal policy π^* is defined as $\pi^* = \arg\max_\pi V^\pi$ – the Q function is given by

$$Q^*(s, a) = \mathbb{E}[R(s, a)] + \gamma \sum_{s'} P(s'|s, a) \max_a Q^*(s', a)$$

For the optimal policy the value functions is denoted by V^*. The goal of the agent is to estimate Q^* and then choose the action $\arg\max_a Q^*(s, a)$ at state s.

3.1 MDPs With Agents

In this section we describe MDP-As and our transfer learning problem. A MDP-A is an MDP with some additional structure in the form of K other agents with fixed policies operating in the environment. It is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, T, L, \gamma, \mathcal{A}', \tau)$, where $\mathcal{A}'$ is a joint action space of K different agents and τ is the joint behavior profile of the K different agents. For $a' \in \mathcal{A}'$, $a'(i)$, the action of the i^{th} agent might be $\emptyset$, indicating that the agent does not perform any action. Each τ is a distribution over $\mathcal{A}'$ indexed by s, a – i.e. $\tau(s,a)$ is a distribution over $\mathcal{A}'$. The transition distribution and reward random variable, respectively have the form $T(\cdot|s,a,a')$ and $L(s,a,a')$ where $s \in \mathcal{S}$, $a \in \mathcal{A}$, $a' \in \mathcal{A}'$. The joint-action a' is such that $a' \sim \tau(s,a)$. Since $\tau(s,a)$ depends only on s and a, each MDP-A is indeed a MDP. Note that we recover the standard form of the MDP transition and reward functions as follows:

$$P(s'|s,a) = \sum_{a'} T(s'|s,a,a')\tau(s,a)(a') \quad (1)$$

and

$$R(s,a) = \sum_{a'} L(s,a,a')\tau(s,a)(a') \quad (2)$$

where $\tau(s,a)(a')$ is the probability of joint action a' under $\tau(s,a)$.

4. TRANSFER LEARNING IN MDP-A

Recall the goal of transfer learning is to use the information gained in previous N tasks, the source tasks, to solve the current task, the target task much faster. In our case, the source tasks are N MDP-As and the target task is another MDP-A (we use the terms task and MDP-A interchangeably). Our goal will be to use the types of agents seen in the source tasks to infer the type of the agents in the target task quickly after some initial exploration and hence solve the target MDP-A much quicker.

In particular, in the transfer setting we consider, we assume that the source MDP-As and the target MDP-A have identical $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, T, R, γ and $\mathcal{A}'$, which are all known, and the only difference between the tasks are the types τ_i. Hence we assume that when solving the target task, the information we are given is in the form of N different types $\tau_1, \tau_2, \cdots, \tau_N$, one corresponding each of N source MDP-As. Our goal is now to use these types to solve the target task much more quickly.

We denote the unknown type in the target task by $\bar{\tau}$. When solving the target task, intially the learner knows exactly the P and R functions (equations 1 2 respectively) at state-action pairs (s,a) for which $\bar{\tau}(s,a)$ is undefined (i.e. the other agents do not have an effect on these pairs) – we call these state-actions pairs the *known* pairs – we call these pairs the *unknown pairs*. It does not know the P and R function for all the other state-action pairs. As the agent explores, the P and R functions at the latter group of state-action pairs become known and the learner has a more accurate value function with which to plan.

Hence, our learner has a exploration-exploitation dilemma when solving the target task. On one hand the it would like to exploit its current model of the world to accumulate rewards, and on the other hand it needs to explore to determine as quickly as possible if the current task has type similar to previous types. Note that this problem is not quite the same as the standard exploration-exploitation problem in reinforcement learning. This is because the distributions $P(s'|s,a)$ and $R(s,a)$ are not entirely unknown, we know that they are a convex combination, respectively, of $T(s'|s,a,a')$, $a' \in \mathcal{A}'$ and $L(s,a,a')$, $a' \in \mathcal{A}'$. So the question is what do we do in this case. In the following we first present the non-transfer/pure RL algorithm four our setting, and from that derive our algorithm.

4.1 Pure Reinforcement Learning

A RL algorithm is defined by the action the learner takes at each step and the subsequent updates it performs. The algorithm we use is the R-Max [2] algorithm, a model based 'optimal' PAC-MDP algorithm [7] (see [14] for a survey). In particular, we use exactly the version of R-max algorithm that appears as Algorithm 1 in [14]. We briefly describe this algorithm and then discuss how it applies to our case.

At each step t with current state s, the R-Max algorithm takes the action $\arg\max_a \hat{Q}^*(s,a)$, where $\hat{Q}^*(s,a)$ is the current estimate of the optimal Q-function Q^*. $\hat{Q}^*$ is initially set to $U(s,a)$, an input to R-Max which has to be an upper bound on $Q^*(s,a)$. The learner takes actions and collects samples at each state-action pair (s,a) – i.e. the next state s' and reward r observed when action a is taken at state s. This is used to maintain empirical estimates of $\hat{P}(.|s,a)$ and $\hat{\mathbb{E}}_{R(s,a)}$. If at any step t the number of samples $n(s,a)$ for a pair (s,a) is above $m \triangleq CV_{\max}^2 \frac{|S|+\ln(|S||A|\delta)}{\epsilon_1^2(1-\gamma)^2}$, for a user given constant C, R-Max performs value iteration for $\lceil \frac{-\ln \epsilon_1 - ln(1-\gamma)}{1-\gamma} \rceil$ steps for all pairs that satisfy that the sample count is $\geq m$. This is the end of the algorithm.

Hence, the basic idea of R-Max is that initially it optimistically assumes that all pairs (s,a) have the maximum possible value - this encourages exploration. Then when m samples are observed, it updates their Q-value as that number is sufficient to guarantee that with high probability, the estimates are close to their true values. The guarantee of R-Max is that after polynomially many steps, with probability $1-\delta$, the value of the learned policy is ϵ-close to the value of the optimal policy (see [14] for details).

First, note that since a MDP-A is an MDP, the R-Max algorithm applies and can ignore the presence of other agents (see Sect. 3.1. We adapt this algorithm in our case in the following way. We set $U(s,a) \triangleq \frac{1}{1-\gamma} \max_{a' \in \mathcal{A}'} L(r|s,a,a')$. The constant C depends on the mixing rate of the policies of the underlying domain and during experiments we try to find the best value for it via trial and error. For the known state-action pairs, the known reward and transition functions are always used, and for these pairs, $n(s,a) > m$ condition is always true. For the unknown state-action pairs, we proceed as in R-max algorithm.

4.2 Transfer Learning

In the transfer case, we adapt the philosophy/idea behind the R-Max algorithm for our purposes. The goal of R-Max is to optimally mix exploration and exploitation. In out transfer learning case, the goal of the learner will be to optimally mix exploration of which of the previous types are being used in the current problem with exploiting the knowledge of which of the previous types are correct. So the goal of the learner will be to determine as quickly as possible which, if any, of the source types is the current type. Then if it turns out that none of the previous type is our current type, then it will just switch to pure R-Max. Otherwise, if only one type is left, then we compute the optimal plan given that the type is true and act ϵ-greedily with respect to that optimal policy. We act ϵ-greedily because it is possible that the current type is wrong and greedy behavior w.r.t. the type-optimal policy keeps us in a region of the MDP where we do not detect that it is defective.

We denote the set elements of the target MDP with a bar on top (like $\bar{P}$). Now let VT_t be the set of types that has not been eliminated as incorrect at step t. For each $\tau_i \in VT_t$, define the reward

function:

$$R_i(s,a) = \mathbb{E}_{\tau_i(s,a)}[L(s,a,a')]$$

and then define the state transition function

$$P_i(s'|s,a) = \mathbb{E}_{\tau_i(s,a)}[T(s'|s,a,a')]$$

Let π_i^* be the optimal policy for the MDP with reward functions R_i and transition functions P_i on the state-action pairs for which τ_i is defined and $\bar{P}$, $\bar{R}$ on all the other state-action pairs. Let Q_i^* be the Q-function for π_i^*. Then at step t, if the learner is at state s and there all pairs s, a are known, it chooses the action:

$$a_t \triangleq \arg\max_i Q_i^*(s,a)$$

For a state that has an unknown action (i.e. (s,a) is unknown), we need to choose the action that will eliminate the type most efficiently. This is what we discuss now.

4.3 Choosing Actions At Unknown States

Let (s,a) be an unknown state-action pair and let $x_{1:t}$ be the actions $a' \in \mathcal{A}'$ observed when a is taken at s in the last t times we visited this pair. The question is how do we use this to eliminate previous types from τ_i. This is the classical hypothesis testing scenario and we present two approaches. The first is the chi-squared test and the second is novel based on Hoeffding bounds. For the second case, we are able to present results on how many samples it takes to eliminate the types with high probability. We present each statistical test in turn, but now we present our method for using the statistical tests.

The usage of the test takes the following form. The test is a function that, given a distribution $\tau(s,a)$ predicted by the type τ that we want to test, and a sample $x_{1:t}, x_i \in \mathcal{A}'$, observed at a particular state s, a, outputs a real $f(x_{1:t}, \tau(s,a))$. If $f(x_{1:t}, \tau(s,a)) > k$, then we can reject $\tau(s,a)$ as the generating distribution with probability α_k, with k and α_k depending on the test. The way this is used to choose actions is given in step 7 in Algorithm 1. We now describe our two tests.

4.3.1 The Chi-Squared Test

This is a standard test from statistics texts, which takes the following form in our case:

$$\chi(x_{1:t}, \tau(s,a)) = \sum_{i=1}^{|\mathcal{A}'|} \frac{(c_i - E_i)^2}{E_i} \quad (3)$$

where c_i is the empirical frequency of action a_i' in $x_{1:t}$ and $E_i = \tau(s,a)(a')$. The significance level α is given by a standard table for chi-squared test.

4.3.2 The Hoeffding Bound Based Test

We introduce a new Hoeffding bound based test, which we can use to give sample complexity bounds for our learning algorithm. The statistic used here is the empirical distribution. As above let $x \triangleq x_{1:t}$ be the sample observed at a particular state-action pair (s,a) and let τ be the type we want to check. The Hoeffding based statistic f is simply

$$f_H(x_{1:t}, \tau) = 1 - \exp[2K - t(\|Pr_x - \tau(s,a)\|_1)^2] \quad (4)$$

with $K = \ln(2^{|\mathcal{A}'|} - 2)$ and Pr_x is the empirical distribution of $x_{1:t}$ (see Section 3). f_H is a valid test in the following sense:

LEMMA 1. *$f_H(x_{1:t}, \tau)$ is the probability that after t steps τ will not have generated a sample y such that $\|Pr_y - \tau(s,a)\|_1 > \|Pr_x - \tau(s,a)\|_1$.*

PROOF. Consider the distribution $\tau(s,a)$ – from [17] (via an application of Hoeffding bound), we know that a sample $y \triangleq y_{1:t}$ generated by $\tau(s,a)$ will satisfy the folllowing:

$$Pr(\|Pr_y - \tau(s,a)\|_1 \geq \epsilon') \leq \delta \quad (5)$$

where, Pr_y is the empirical distribution of y (see section 3) and $\delta = \exp[2K/t - \epsilon'^2]$ with $K = \ln(2^{|\mathcal{A}'|} - 2)$. Hence, by setting $\epsilon = \|Pr_x - \tau(s,a)\|_1 \triangleq \epsilon_1$, we get that

$$Pr(\|Pr_y - \tau(s,a)\|_1 \geq \epsilon_1) \leq \exp[2K - t\epsilon_1^2]$$

and so

$$Pr(\|Pr_y - \tau(s,a)\|_1 \leq \epsilon_1) > 1 - \exp[2K - t\epsilon_1^2] = f_H(x_{1:t}, \tau)$$

Of course, $Pr(\|Pr_y - \tau(s,a)\|_1 \leq \epsilon_1)$ is simply the probability in the statement of the lemma, and so this completes the proof. □

So if $f_H(x_{1:t}, \tau) > \alpha$, with probability α, τ would have generated a sample less than $\| \Pr_x - \tau(s,a)\|_1$ and we can reject τ with probability α. When we have n different types τ_i to test, by the union bound, we need that:

$$\sum_i (1 - f_H(x_{1:t}, \tau_i)) \leq (1-\alpha) \Rightarrow \sum_i f_H(x_{1:t}, \tau_i) \geq n - \alpha \quad (6)$$

for the guarantee above to hold for all the types simultaneously.

4.4 Algorithm Description and Analysis

Our algorithm is presented in 1. It takes as input $\tau_1, \cdots \tau_N$ the set of source types, T, the number of steps for which to run and the test statistic function f, and its range value α. The main variable is VT_t, which are the set of types that have not been ruled out as potential types at the beginning of step t.

In line 3, the main loop of the algorithm starts, and it continues for T steps. In line 5, the algorithm checks to see if the current state is unknown for same action a (i.e. $\bar{\tau}(s,a)$ makes a prediction). If it is, in line 6 we eliminate the types that do not predict anything (and hence cannot be the true type $\bar{\tau}$). In line 7-8 we act conservatively and take the action that maximizes the minimum possible change in our chosen test and update the counts. Finally, we eliminate the types that fail the statistical test.

Otherwise, in line 10 onwards we address the case that the state is known for all the actions (i.e. $\bar{\tau}$ does not make a prediction). We start by removing all types that do make a prediction. Then in lines 12-15, if only one type is left, we act-ϵ greedily with respect to that type. Otherwise, we act greedily according to the type with best predicted future reward.

Finally, if all the types are eliminated, we switch to R-Max in line 21 by initializing it with our observed counts so far.

We now establish how quickly this algorithm rejects the incorrect source types τ with high the desired degree of probability. Toward that end, we need a particular mixing time.

DEFINITION 1. *Define T_m to be the time such that any policy will have visited each of the unknown state-action pair at least m times.*

This type of mixing time is used in bounding time-to-convergence of PAC-MDP and many other algorithms giving performance guarantees for MDPs (see for instance, [7] or [2]. In fact, the notion we use is likely to be much weaker than the ones used in those papers – that is in applications we expect T_m to be much smaller than the mixing times presented previously. This is because the mixing times used in the earlier papers bound the number of steps it takes

for a policy to converge to its true returns, whereas we just want the number of visits to the unknown states. We can now show the following.

THEOREM 1. *Define* $\Delta_{\min} = \min_{\tau_i} \min_{s,a} \|\bar{\tau}(s,a) - \tau_i(s,a)\|_1$ *and for fixed* δ*, define* $m \triangleq (2K - \ln(\delta/2n))/(\Delta_{\min}/2)^2$*. Then with probability* $1-\delta$*, after* T_m *steps, the Type-Elimination when using the Hoeffding test* f_H*, eliminates all the incorrect types.*

PROOF. After $m = (2K - \ln(\delta/2n))/(\Delta_{\min}/2)^2$ samples (K defined in section 4.3.2) are collected at a pair (s,a), the empirical distribution Pr_x of the samples is within $\Delta_{\min}/2$ of $\bar{\tau}(s,a)$ with probability at least $1-\delta/2N$ (this is from [17] via an application of the Hoeffding bound). By definition of $\Delta_{\min}$, for each incorrect type τ_i, all states satisfy $\|\tau_i(s,a) - \bar{\tau}(s,a)\|_1 \geq \Delta_{\min}$. We can pick N such state-action pairs and the probability that the samples $x^i, 1 \leq i \geq n$ all simultaneously satisfy $\|Pr_x - \bar{\tau}(s,a)\|_1 > \Delta_{\min}/2$ is at least $1 - n\delta/2N = 1 - \delta/2$.

Now consider an instantiation z of x^i that does satisfy $\|Pr_z - \bar{\tau}(s,a)\|_1 > \Delta_{\min}/2$. Plugging $t = m$ and $\Delta_{\min}/2$ as the distance into the Hoeffding test definition in 4) we get $f_H(z,\tau_i) = 1 - \exp[2K - m(\Delta_{\min}/2)^2] = 1 - \delta/2N$. With probability at least $1-\delta/2N$, the Hoeffding test rejects z. Now the variable x^i is independent of the random variable y in Lemma 1 and hence by the union bound we have that with probability at least $1 - 2\delta/2N = 1 - \delta/N$ the Hoeffding test rejects the random sample x^i. Since the distributions $\tau(s,a)$, $\tau(s',a')$ are independent whenever $(s,a) \neq (s',a')$, the union bound again applies and we have that with probability $1-\delta$ all N x^i are rejected by the Hoeffding test, completing our proof. □

5. EXPERIMENTS

In this section we describe experiments with our algorithm on three different domains and compared with Policy-Reuse [6]. For each domain we chose 10 source types and transferred to three targets. The first target (target 0 in the sequel) was with type identical to one of the sources; the second target was identical to a target profile for only half of the unknown state-action pairs (target 1 in the sequel); and the third (target 2) was totally dissimilar (target 2 in the sequel) – in this case no transfer was possible. We tested our algorithm against the policy reuse algorithm because, as we discussed in Section 2 this is the only relevant algorithm that we are aware of – please see the next section for more detail. We used R-Max as the base-line RL algorithm. All our results are presented averaged over 10 trials. The results we present are the number of time steps it took for the agent to reach the goal state. Since the reward structure in the domains were fully delayed, where we gave a reward of -1 for all actions and a positive reward only when the learner reached the goal state, this is appropriate. We excluded error bars for the sake of clarity. The results are presented in the Figures 1 to 9. In the legend of the figures, TEi stands for Type-Elimination with target task i and similarly PRi stands for Policy-Reuse with target task i. RMX stands for R-Max. It will be noticed that all the figures are missing the curve for TE2 – this is becuase our algorithm worked as expected and is discussed below in more detail. For the Type-Eliminiation algorithm we only show the results when using our Hoeffding test – the Chi-squared version tended to reject all the types immediately and simply switch to R-Max.

5.1 Policy Reuse

The policy resue algorithm is described in detail in [6]. It is a heuristic algorithm that when given a collection of policies for a particular domain, uses a soft-max criteria to choose between the different policies. A default member of the collection is the Q-learner policy. When chosen, each policy is run for an episode and the weight of each policy is the total discounted reward obtained by running that policy, averaged over the episodes in which it was run. There is a very simple way to adapt this algorithm for our case. Given the N source types τ_i, the policy library consists of $\{\pi_i : \pi_i(s) = \arg\max_a Q_i^*(s,a)\}$ – i.e. greedy with respect to the Q-function of the source types. In the setting we consider, the main advantage of our algorithm is that it quickly, within a single episode able to eliminate many incorrect source types, while for the policy reuse case, it may need to many episodes to stop using the wrong policies.

Algorithm 1 Type-Elimination

Input: Previous types $\tau_j, 1 \leq j \leq N$, initial state s_i, test f and constant α.
Initialize: Valid types $VT_0 = \{\tau_i : 1 \leq i \leq N\}$, $t = 0$, $s_0 = s_i$, $t = 0$.
for $t = 1$ to T and $|VT_t| > 0$ **do**
  Set $V_{t+1} = V_t$
  if (s_t, a) unknown for some a **then**
    Eliminate all τ_i from VT_t such that $\tau_i(s,a)$ is undefined.
    Take action $a_t \triangleq \arg\max_a \min_{a', \tau_i \in VT_t}[f(x_{1:t}a', \tau_i(s,a)) - f(f(x_{1:t}, \tau_i(s,a)]$, where $x_{1:t}$ is the sample collected at s, a.
    Observe s_{t+1}, a'_{t+1}, r_{t+1} and update empirical distributions and rewards $\hat{P}(.|s_t,a_t)$ and $\hat{\mathbb{E}}_{R(s_t,a_t)}$
    Eliminate all τ_i from VT_t such that $f(x_{1:t}, \tau_i) > \alpha_f$ where $x_{1:t}$ is the sample collected at s_t, a_t.
  else
    Eliminate all τ_i from VT_t such that $\tau_i(s,a)$ is defined.
    if $|VT_t| = 1$ **then**
      With probability ϵ, choose a_t to be an action chosen at random, and other choose the action $a_t = \arg\max_a \bar{Q}^*(s_t,a)_{VT_t}$.
    else
      Choose the action $a_t = \arg\max_a \bar{Q}^*(s_t,a)_{VT_t}$.
    end if
    Take a_t observe s_{t+1}, r_{t+1}, update next state and reward distributions for (s_t, a_t).
  end if
end for
if $t < T$ **then**
  Run RMax for $T - t$ steps, initialized with known $\bar{P}$ and $\bar{R}$ at known states and $\hat{P}(.|s,a)$ and $\hat{\mathbb{E}}_{R(s,a)}$ at the unknown but visited states.
end if

5.2 Cleaning Robots Domain

In the cleaning robot task the learner needs to learn to go from a target location to a goal location. However, there are other agents in the domain who are performing tasks in particular locations, in this case cleaning the floor. A particular task consists of a set of locations that the robots are cleaning at. If the learner attempts to enter an area where a robot is currently cleaning, the learner is blocked by that robot. If the learner attempts to enter an area, he may be blocked, but not with certainty. Hence, a type consists of the set of locations where the robots are cleaning and will block or semi-block the learner. Depending on the time of the day, the type changes, and the current type may or may not be the same as previous types. Our algorithm will help the learner take advantage if a particular location pattern repeats itself.

More precisely, our domain is a 30×30 gridworld. The state of the MDP is the (x, y) coordinate and the actions of the learner are movements in the 4 cardinal directions, each of which succeed with probability 0.7. There are 10 robots and for a given MDP, there are 10 different rectangular blocks that the learner is prevented blocked and semi-blocked, one for each of the robot. If the learner attempts to move into a blocked region, then the learners probability of moving in the desired direction is 0. If the learner tries to move into a semi-blocked region, the probability of him moving in the desired direction is 0.3. Finally, the sources we used assigned varying probabilities to the actions (chosen at random) and at random locations.

The results are presented in Figures 1 to 3. The second two figure each zooms in on three of the curves. For target 0 and 1, Type-Elimination eliminated all the incorrect types within the first episode and thereafter followed only the correct type or half-correct type (target 0 and 1 respectively). For these types, the corresponding policy found in the source was accurate enough so that it performed quite well. Also, for target 2 Type-Elimination almost within the first episode rejected all the incorrect types and switched to R-Max. For this reason its learning curve is missing from the graphs. This is good because recall that target 2 is completely different from all of the source and should be rejected. Type-Eliminiation outperformed Policy-Reuse quite handily.

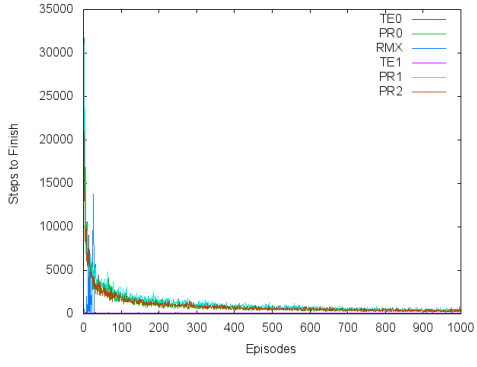

Figure 1: Cleaning Agents domain: Combined Results

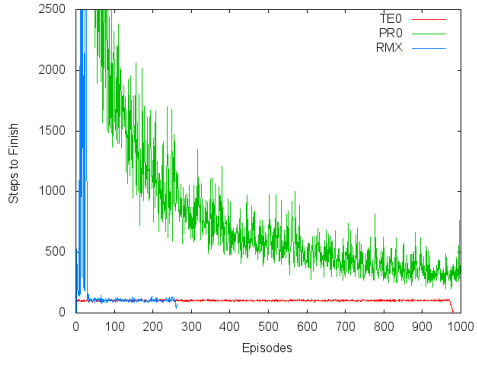

Figure 2: Result for Cleaning Agents Domain: zoomed in.

5.3 Congestion Domain

In this task, the goal of the learner is to navigate around a part

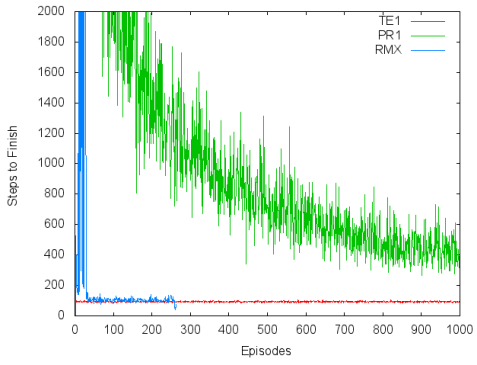

Figure 3: Result for Cleaning Agents Domain: zoomed in.

of a city and go from a start location to a goal location as quickly as possible. In this case, the other agents are the other cars in the road and the learner needs to avoid the traffic jams. The location and size of the traffic jams and free regions will depend on the current time of the day, and each such period of time consists of a task and the pattern of traffic and free road is our type – this is a type because this determines whether the other agents are allowing or impeding the progress of the learner. In this case, the transfer learning should help the learner determine from its experiencing traffic at a particular location whether there will be a jam in some other location.

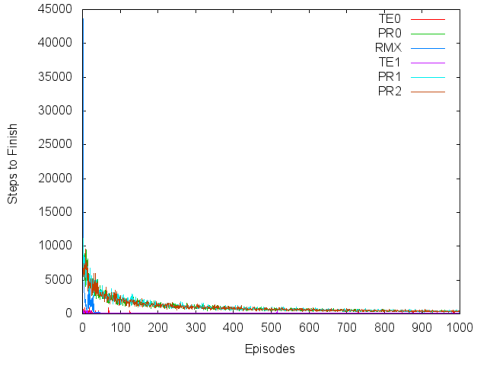

Figure 4: Results for the Congestion domain.

This domain is a 30×30 gridworld. The learner has three actions, turn left and turn right, move forward and has three state variables, orientation, which can take 4 values, and the x, y location. The turning action changes the orientation, and while the move forward action changes the x, y location as determined by the orientation. Each cell represents a particular cluster of city blocks and has either normal, mild traffic, medium traffic or heavy traffic. The condition of the traffic is only available when the agent arrives at that location and tries to move. In each case, there is a probability that the other agent will prevent the action the agent is trying to perform. In a normal traffic, the probability is 0, while in the mild, medium, heavy and crawl traffic setting the probability is respectively $0.5, 0.6, 0.8$ and 0.9 respectively. As before the sources types were assigned the location of their agents at random.

The results for this task are presented in Figures 4 to 6 (again the latter to zooming in). The tale being told by this graph is the

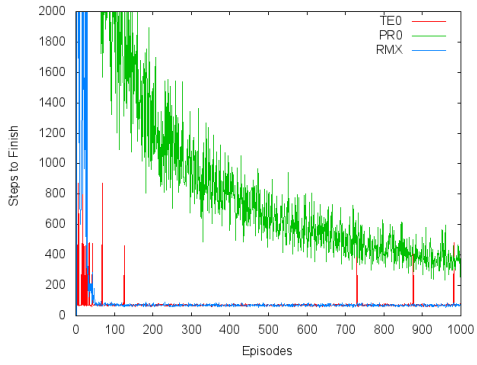

Figure 5: Results for the Congestion domain: zoomed in.

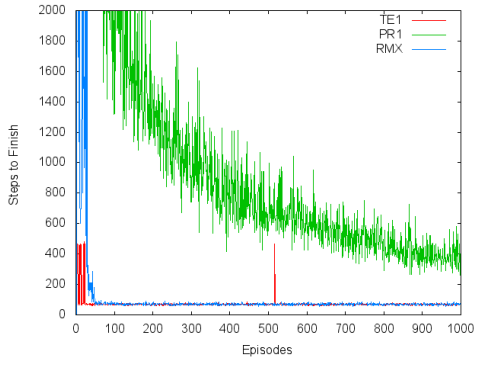

Figure 6: Results for the Congestion domain: zoomed in.

same as the in the Cleaning Agent domain, with Type-Elimination outperforming Policy-Reuse and rapidly rejecting incorrect source types, but keeping the useful source types. The only anomaly here are the spikes – we believe this is simply because we needed to average over more trials.

5.4 Niche Factory Floor

In this problem we consider the learner operating in a factory that performs flexible, niche production. This domain was inspired by [1] [8], as an analogue to and extension of the the Kiva systems [4] in the context of these new type of factories. In this case, instead of producing some specific items, the factory is able to construct customized products to meet the specialized needs of a niche market, and the specific needs can vary quite rapidly compared to traditional markets. We consider a factory making industrial and household robots and at any point in time, there are M different types of robots being constructed by M different collection of agents which share common machines and warehouses for components needed for assembling the robot. The learner controls one such collection of agents and its goal is to assemble its own assigned product using the shared resources. As mentioned above, due to varying market conditions, the exact needs of the other groups vary based on what the customers are requesting at that point in time. For efficiency reasons we want production to be performed in a distributed fashion, and so the learner has to determine through interaction what types of schedule of access to resources will work given the current demands of the other agent groups.

More specifically, to assemble its assigned robot, the learner needs to procure 5 components and then put them together. The components may be put together in any order. But the warehouse the items being procured from and the assembly machines are shared and so access to them is not guaranteed. The state is defined by 5 ternary features, one for each component. A value of 0, 1 and 2 means, respectively, that the corresponding component has not been procured, procured and assembled respectively. Hence, the learner starts off with features $0, 0, 0, 0, 0$ and finishes the production cycle with component features $2, 2, 2, 2, 2$. The factory floor has 5 warehouses, each housing certain combinations of components and 5 assembly stations, each of which can combine certain components. The goal of the learner is to procure the components from the warehouse, and then take them to the assembly stations, at which point they may be assembled. As mentioned above, the other agents are also sharing this resource. Ordinarily when no other agent is competing for these resources, each of these actions succeed with probability 0.9. However, when there are other agents, the actions may succeed with probability 0.5, 0.3 0.1 depending on how much demand there is from the other agents. This demand manifests as various three actions, one for each possible chance of success.

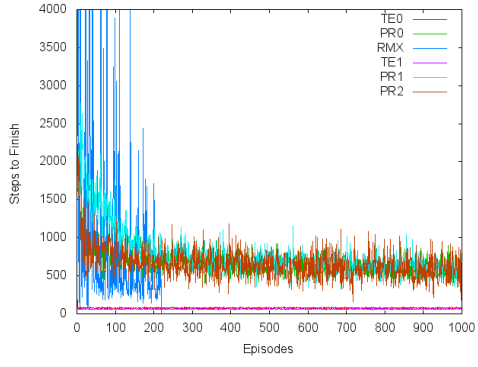

Figure 7: Results for the Niche Factory Floor domain.

Hence, each type is determined by the pattern of demand at the various warehouses and stations. The domain is a 10×10 grid-world and the 20 locations mentioned above are scattered around the domain. Therefore, the state of the learner is defined by 8 features. The first 5 were mentioned above. The next 2 are the location on the grid, and final one indicates the type of the cell (i.e. floor, warehouse, assembly station). So in total there are 24300 states for this task. The learner has 4 actions, each for movement in each of the cardinal directions, and 1 action for requesting procurement and another action for requesting assembly. The learner gets a reward of 1 for every successful procurement and assembly operation and a reward of 10 when it reaches the shipping point with all the features $2, 2, 2, 2, 2$. The learner receives a reward of -1 for every time step, which means it needs to construct its batch of robot as quickly as possible. The sources were again constructed randomly.

The results for this task are presented in Figures 7 to 9. The qualitative result of this graph is the same as before so we do not elaborate on it further. The only exception here is that the variance of R-Max is extremely high in this domain, so the fact that Type-Eliminiation immediately gives is only a partial win. We discuss this further in the Conclusion.

6. CONCLUSION

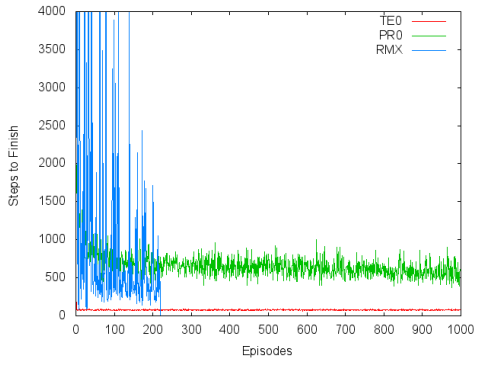

Figure 8: Results for the Niche Factory Floor domain: zoomed in.

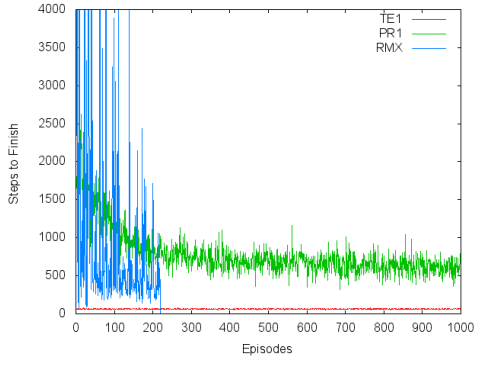

Figure 9: Results for the Niche Factory Floor domain: zoomed in.

In this paper we present an algorithm for a class of non-stationary environments where the non-stationarity arises from the presence of other decision making agents whose behavior changes discretely, slowly but greatly. We pose this problem as transfer learning in a novel sub-class of MDPs, so that learning to deal with new agent behavior can be posed as transferring information from a previously solved MDP to a new one. We present an algorithm in this setting which uses a novel statisitcal test of our formulation to rapidly eliminate previously solved MDPs that will not be useful for the current problem. We present some theoretical performance guarantees for this algorithm and then demonstrate its empirical properties by performing experiments on three domains with different characteristics. The results show that the algorithm performs as promised, in that it is able to stop using previous tasks if there is reasonable disagreement between the model implied by the previous task and the observed world. However, this also points toward a potential shortcoming of our algorithm which may be relevant in some domains, which is that it does not try to make use of previous tasks that are partially correct. This will be one possible future extension of this algorithm. Another possible criticism might be that instead of evaluating previous types based on their predicted disitrubtion over agent actions we should simply evaluate based on the observed distriubiton over states. However, this would violate our core motivation of the existence of types - a notion that enjoys compelling support in many different domains including game theory and cognitive science. Our current approach suggests that we may be able to extend this basic idea to extensive form games and other related models of inter-dependent decision making.

Acknowledgements

This work has taken place in the Robust Autonomy and Decisions group within the School of Informatics. Research of the RAD Group is supported by the UK Engineering and Physical Sciences Research Council (grant number EP/H012338/1) and the European Commission (TOMSY Grant Agreement 270436, under FP7-ICT-2009.2.1 Call 6).

7. REFERENCES

[1] A third industrial revolution. *Economist*, April 21^{st}, 2012.

[2] R. Braffman and M. Tennenholtz. R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[3] S. Choi, D.-Y. Zhang, and N. L. Zhang. Hidden-mode markov decision processes for nonstationary sequential decision making. *Sequence Learning – Paradigms, Algorithms and Applications*, 2000.

[4] R. D'Andrea. Guest editorial: A revolution in the warehouse – a retrospective on the kiva systems and the grand challenges ahead. *IEEE Transactions on Automation Science and Engineering*, 9(4), Ocotber 2012.

[5] F. Fernandez, J. Garcia, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58:866–871, 2010.

[6] F. Fernandez, J. Garcia, and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, 206.

[7] M. J. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3), 1999.

[8] P. Marsh. *A New Industrial Revolution*. Yale University Press, 2012.

[9] J. Morimoto and K. Doya. Robust reinforcement learning. *Neural Computation*, 17:335–359, 2005.

[10] A. Nilim and L. E. Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[11] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[12] J. Ramon, K. Driessens, and T. Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of The Eighteenth European Conference on Machine Learning*, 2007.

[13] B. C. Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the* 23^{rd} *International Conference on Machine Learning*, 2006.

[14] A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, 10:2413–2444, 2009.

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

[16] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[17] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. J. Weinberger. Inequalities for the L1 deviation of the empirical distribution. Technical Report HPL-2003-97R1, Hewlett-Packard Labs, 2003.

[18] J. Y. Yu and S. Mannor. Arbitrarily modulated markov decision processes. In *Proceedings of the IEEE Conference on Decision and Control*, 2009.