Using Spatial Hints to Improve Policy Reuse in a Reinforcement Learning Agent

Bruno N. da Silva
Computer Science Department
University of British Columbia
201-2366 Main Mall
Vancouver, BC Canada
+1 604 822 3061
bnds@cs.ubc.ca

Alan Mackworth
Computer Science Department
University of British Columbia
201-2366 Main Mall
Vancouver, BC Canada
+1 604 822 3061
mack@cs.ubc.ca

ABSTRACT
We study the problem of knowledge reuse by a reinforcement learning agent. We are interested in how an agent can exploit policies that were learned in the past to learn a new task more efficiently in the present. Our approach is to elicit spatial hints from an expert suggesting the world states in which each existing policy should be more relevant to the new task. By using these hints with domain exploration, the agent is able to detect those portions of existing policies that are beneficial to the new task, therefore learning a new policy more efficiently. We call our approach Spatial Hints Policy Reuse (SHPR). Experiments demonstrate the effectiveness and robustness of our method. Our results encourage further study investigating how much more efficacy can be gained from the elicitation of very simple advice from humans.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *intelligent agents*.

I.2.6 [**Artificial Intelligence**]: Learning – *knowledge acquisition*.

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation – *Probabilistic computation*

General Terms
Algorithms, Design, Performance, Experimentation.

Keywords
Spatial hints, Policy reuse, Reinforcement learning, transfer learning.

1. INTRODUCTION
Reinforcement learning is a popular technique in the design of intelligent agents [2,6]. However, the large state- and action spaces of some domains are sometimes a severe limitation to the performance of the agent on a learning task. While traditional algorithms tended to explore the environment and exploit the information learned during exploration, new approaches attempt to overcome this limitation by using some input other than the current navigation of the agent and the collected rewards. The expectation is that this external source might assist the agent in reaching a satisfactory performance more effectively.

One such external source is to allow humans to guide the reinforcement learning agent. For example, by evaluating how real users provide biased feedback, Thomas and Breazeal [9] adapt the interpretation of the reinforcement signal to extract information about future explorations, as well as the evaluation of recent actions. Another possibility is to employ heuristics or expert knowledge to the function approximation method in order to incorporate information about preferred actions [3].

Another type of external source is the reuse of knowledge from the tasks that were learned in the past. This knowledge reuse is known as transfer learning [7]. For example, Taylor et al. [8] show that mappings linking states and actions of past tasks to states and actions of the current task allow the agent to find a good bias for the Q-value initialization. This approach (called Inter-Task Mappings) enables the agent to reach a satisfactory performance faster than when learning tabula rasa. However, Inter-Task Mappings require expert knowledge of both the past and current tasks, which might be very difficult or costly to elicit.

When the existing policies share the same space and action spaces with the current task, an alternative approach is direct experimentation with past policies in the new environment. This approach is called Policy Reuse [1], and its insight is that if an existing policy is adequate for a new task, then the reuse of this policy will lead to the collection of higher rewards. Therefore, Policy Reuse allows actions dictated by existing policies to guide the exploration in the new task, at the same time that the reinforcement signal is used to update the Q-values and construct a new policy. Like an N-armed bandit, the algorithm repeatedly samples from the set of policies to determine if one will be the reference for each episode. If no policy is selected, a Q-Learning episode is executed. The policy selection is biased towards those that yield higher rewards. Therefore, the algorithm converges to the use of a similar past policy (if any) with no human input other than the design of the reward function.

Clearly, there is a qualitative spectrum over the amount of human assistance in reusing past policies. While Inter-Task Mappings require a significant effort from humans, Policy Reuse needs none. Therefore, one open question remains regarding the right

Cite as: Using Spatial Hints to Improve Policy Reuse in a Reinforcement Learning Agent, Bruno da Silva and Alan Mackworth, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 317-324 Copyright © 2010, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

level of independence from humans; in other works, is there a method that improves the performance on the learning task without requiring too much human assistance?

Inspired by this reflection, we extend the Policy Reuse method to incorporate spatial hints from users. More concretely, instead of relying just on the set of past policies learned by the agent, we allow users to optionally specify reference points from the state space for each policy. This information allows our method to more efficiently integrate complementary policies that work well in different situations.

This paper is organized as follows: Section 2 contains the formal problem definition. Section 3 introduces our contribution, the Spatial Hints Policy Reuse (SHPR) algorithm. Experiments are described in Section 4. In Section 5 we present a general discussion while Section 6 ends with our conclusions.

2. PROBLEM DEFINITION

We define a Reinforcement Learning problem using a Markov Decision Process (MDP). An MDP is a tuple < S, A, T, R >, where S is the set of states, A is the set of actions, R: S x A → $\mathbb{R}$ is a reward function, and T: S x A x S → [0,1] is the transition function. T and R are unknown to the agent.

Definitions 1, 3 and 6 are from [1, pp. 721]:

Definition 1. A domain D is a tuple < S, A, T >, where S is the set of all states; A is the set of all actions; and T is a state transition function, T: S x A x S → [0,1].

This definition characterizes the invariants across the current task and all tasks that will be reused by the method. It enables a policy defined in one task to be executed for a different task in the same domain.

Definition 2. A policy π: S → A assigns one action for each member of the state space of some domain.

Definition 3. A task Ω is a tuple < D, R_Ω >, where D is a domain, and R_Ω is a reward function, R: S x A → $\mathbb{R}$.

Therefore, the only difference across tasks that are defined in the same domain is the different reward that an agent receives after executing actions in certain states.

Definition 4. A hint h is a pair < π_h, s_h >, where π_h is a policy defined on the domain D and $s_h \in S(D)$ is a state in the state space S of D. We call s_h the reference point of the policy π_h.

Definition 5. A hint library L is a set of n hints $\{h_1,\ldots,h_n\}$. ∃D, such that each hint $h_i \in L$ solves a task Ω = < D, R_Ω >.

Therefore, every hint in the library is defined over the same domain. This makes every policy and state across all hints in L to share the same domain as well. The library can have hints that share the same policy, or even hints that share the same reference point.

Alternatively, Policy Reuse (which we will use as a baseline) defines a policy library:

Definition 6. A policy library, L, is a set of n policies $\{\pi_1,\ldots,\pi_n\}$. Each policy $\pi_i \in L$ solves a task I =< D, $R_{\Omega i}$ >, i.e. each policy solves a task in the same domain.

We are interested in episodic tasks with absorbing goal states, i.e. p(s_{goal},a,s_{goal}) = 1, ∀a. By analogy with the scheduling problem we define an episode as follows:

Definition 7. A step t begins in a certain state s_t and ends when the agent executes an action a_t and receives a reward r_t for that action in that state. A slot σ is a sequence of k steps.

Definition 8. An episode is a sequence of slots $\{\sigma_0,\ldots,\sigma_{k-1}\}$, each of them containing the same number of steps (except possibly the last one). An episode ends after reaching the maximum number of slots k or when the goal state is reached.

As illustrated in Figure 1, our method schedules policies (or any algorithm that dictates which actions to take given a state) to slots in each episode, much like a pre-emptive scheduler of an operating system.

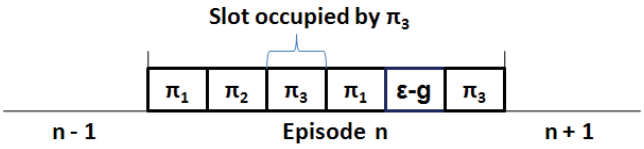

Figure 1. We define an episode as containing k slots. Each slot is occupied by a policy (or an exploration strategy such as ε-greedy) that will dictate the actions to be taken in the steps of this slot.

This definition is a generalization of the standard concept of an episode. In Policy Reuse, for example, the same policy is used across all slots of the same episode. Likewise, one could not use any existing policy and let a particular algorithm dictate actions for every slot of all episodes.

The evaluation metric is defined as the average reward per episode:

$$W(E) = \frac{1}{E}\sum_{e=0}^{E-1}\sum_{k=0}^{k_e-1}\sum_{t=0}^{t_k-1} \gamma^{k+t} r_{k,t,e}$$

where E is the number of episodes, k_e is the number of slots in episode e and t_k is the number of steps in slot k (either T or less if the goal state was reached). $\gamma \in [0,1]$ is the discount factor for future rewards, and $r_{k,t}$ is the reward received in step t of slot k of episode e.

3. APPLYING SPATIAL HINTS TO POLICY REUSE

This section explains how we use a hint library to better exploit existing policies and learn a new task. The challenge when reusing policies is to discriminate the states of the world in which some policy from the library should be reused from the states of the world in which no policy from the library would be useful (therefore requiring independent exploration).

While a library of hints is a good initial approach for this problem, it doesn't solve it altogether. Since a hint associates a policy with a single state, it would be excessive to ask users to specify the ideal policy for every state of the world. Therefore, some metric is needed to estimate how useful each hint could be for each state of the world.

Naturally, such a metric should consider the distance between the current state of the world and the policy's reference point. Hints that are farther away would be less likely to be useful than closer hints. Additionally, the quality the hints might not be uniform. While some existing policies might be more suitable to be executed in more states of the new task, some of them might be less useful, and therefore should exert a weaker influence on the learning agent.

For this reason, we associate with each hint h_i a variable $reach_i$ which estimates how good the policy π_i is around its reference state s_i. Policies that perform well around their reference state should have their respective reach increased to extract significant contributions from good existing policies. Likewise, hints that are not as good (e.g. whose policy is not a good alternative for that reference point) should not be reused as often. This has an analogy to the laws of attraction from physics. The reach of a hint can be considered its mass. Bodies with stronger mass exert a stronger force of attraction, just like bodies that are closer to the respective object. Therefore, we need a metric that is proportional to $reach_i$, and inversely proportional to the distance between the current and reference states. That's why we chose to assign policies to slots with probability proportional to w_i:

$$w_i = \frac{reach_i}{1 + distance(\ current\ state, reference\ state_i\ )}$$

where distance can be any metric defined over the state space. In this paper, we use the Manhattan distance.

After the assignment of a policy π_i to a slot, the execution of this slot starts. The selection of actions in this slot will be determined by two sources: the existing policy π_i and an ε −greedy procedure based on the Q-values of the current task:

$$\varepsilon - greedy(\pi_{new}) \begin{cases} best\ action, with\ probability\ \varepsilon \\ random\ action, with\ probability\ 1-\varepsilon \end{cases}$$

The action selection is initially determined by π_i. In subsequent steps, there is a probabilistic balance between actions by π_i and ε −greedy, until the end when ε −greedy dominates the action selection process (Figure 2).

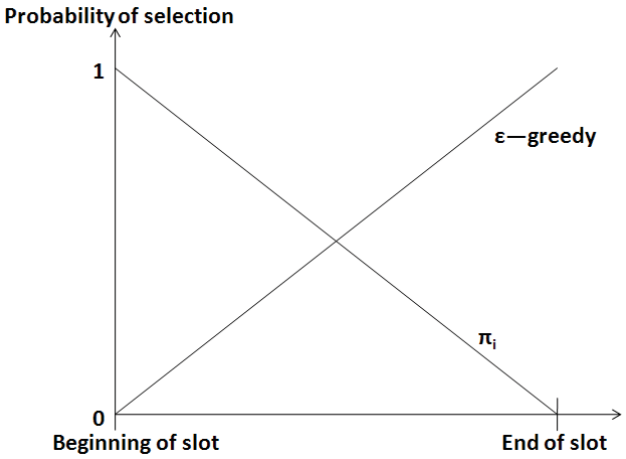

Figure 2. Action selection process in a slot combining an existing policy π_i and an ε −greedy procedure.

Table 1 describes the algorithm executed during each slot. After the policy for that slot π_i has been determined, it is passed together with the initial state of the slot to the algorithm. The basic iteration is repeated for the maximum number of steps T, or until the goal state is reached. The ε −greedy strategy starts at a specific level of randomness ε_0, and becomes greedier in every step by a factor of $\Delta\varepsilon$. The update of the policy is performed no matter where the action came from. Therefore, even if an action that was dictated by an existing policy is not appropriate in this new task, there is still useful information that is collected through this update of the Q table.

Table 2 contains the definition of an episode. Naturally, it consists of a sequence of calls to the slot algorithm. We start by considering the initial state, and based on this state, computing a probability distribution over the existing policies (in the library of hints).

This distribution is proportional to the past performance of each function ($reach_i$) and inversely proportional to how far that policy was referenced by the user (1+dist). After the chosen policy guides the agent in one slot, the current state s_{curr} changes, yielding a different distribution over policies.

Algorithm slot(π_i, $s_{initial}$)
$p := 1.00$
$s_{curr} := s_{initial}$
$\varepsilon := \varepsilon_0$
Repeat T times or until $s_{curr} = s_{goal}$
$action := \begin{cases} \pi_i(s_{curr}), with\ probability\ p \\ \varepsilon - \text{greedy}(\pi_{\text{new}}(s_{curr})), \text{with prob } 1-\text{p} \end{cases}$
Execute action, collecting s_{next} and reward r
$Q^{\pi_{new}}(s_{curr}, action) := (1-\alpha)\ Q^{\pi_{new}}(s_{curr}, action) + \alpha\ [\ r + \gamma \max_a Q^{\pi_{new}}(s_{next}, a)\ ]$
$p := p - {}^1/_T$
$s_{curr} := s_{next}$
$\varepsilon := \min(\ 1, \varepsilon + \Delta\varepsilon\ )$

Table 1. Slot exploration strategy

Algorithm episode($s_{initial}$, Library)
$s_{curr} := s_{initial}$
Repeat K times or until $s_{curr} = s_{initial}$
Let $w_i = \frac{reach_i}{1+distance(s_{curr}, s_i)}\ \forall i \in Library$
Select π according to distribution $p(\pi_i) = \frac{w_i}{\sum_j w_j}$
Execute slot(π, s_{curr})
Retrieve new s_{curr} from slot

Table 2. Episode definition strategy

The algorithm episode above defines how the library of hints is used to generate an episode of the learning task. Initially, this library of hints used in each episode is composed of those existing policies selected by the user. However, these policies might be undefined or very ineffective in some parts of the state space. Therefore, simply reusing the same set of hints in every episode might be too inefficient an approach.

Naturally, one solution to overcome the inadequacy of existing policies would be to use a traditional exploration/exploitation of the environment. We already do this by switching from the use of the existing policy and $\varepsilon-$greedy in each slot (Figure 2). However, it might still be the case that the area around the initial state of an episode might not be adequately covered by existing policies. Therefore, before the start of each episode, we artificially introduce an extra entry in the library of hints containing the $\varepsilon-$greedy policy having the initial state $s_{initial}$ as reference point, with different values of $reach_{\varepsilon-greedy}$. This constitutes an experiment, which tries to identify which is value of $reach_{\varepsilon-greedy}$ in $s_{initial}$ will better combine with the existing policies to lead to higher rewards.

Given that $distance(s_{initial}, reference_{\varepsilon-greedy}) = 0$, we can vary $reach_{\varepsilon-greedy}$ from a low to a high value to estimate more precisely how the episode performs when $\varepsilon-$greedy has a low probability and when it has a high probability of being assigned to one of the slots of the episode. Naturally, this is not a perfect experiment because future tests setting a higher value of $reach_{\varepsilon-greedy}$ will have benefitted from the knowledge acquired in past tests. This will create a bias towards higher values of $reach_{\varepsilon-greedy}$. However, this is not necessarily a bad thing. One of the purposes of introducing $\varepsilon-$greedy in the first place is to slowly get rid of existing policies, and with each episode to rely increasingly more on concrete knowledge about the current task. The exploration dictated by $\varepsilon-$greedy relies on this concrete knowledge, and is therefore beneficial to the learning agent in an advanced stage of the process. An abstract version of our algorithm is presented in Table 3.

Abstract version of SHPR
Repeat
Pick a state from the state space
Repeat the question
For this state, are the existing policies enough? If not, how strong should the $\varepsilon-$greedy policy be to jumpstart the collection of good rewards?
Design an experiment to answer this question, and…
Record an $\varepsilon-$greedy entry in the library of existing policies that's as strong as necessary.

Table 3. Abstract definition of SHPR

Finally, Table 4 presents the complete version of our algorithm. It is called SHPR, which stands for Spatial Hints for Policy Reuse. We start by repeating a number of times (E, an input parameter) the procedure equivalent to the first Repeat from Table 3. Next, the algorithm needs to determine how many times the question from the second Repeat from the abstract definition is going to be asked (REPETITIONS_PER_EPISODE is another input parameter). The experiment is equivalent to the while loop, where each iteration is a test with a different hypothesis.

Algorithm SHPR(Library)
For each $\pi_i \in Library$
$reach_i := INITIAL\_REACH$
Repeat E times
$s_{initial} := selectInitialState()$
$reach_{\varepsilon-\text{greedy}} := 1$
$maxReach := \max_{\pi_i \in Library} reach_i$
$\Delta reach := \frac{\lvert maxReach - reach_{\varepsilon-\text{greedy}} \rvert}{REPETITIONS\_PER\_EPISODE}$
$accRewards := 0$
accWReach := 0
While $reach_{\varepsilon-\text{greedy}} \leq maxReach$
tempLibrary := Library $\cup$ { $\varepsilon-\text{greedy}(\pi_{\text{new}})$ with reference at s_{initial}, reach = $\text{reach}_{\varepsilon-\text{greedy}}$}
Execute episode($s_{initial}$, $tempLibrary$)
Retrieve total discounted reward R from episode
For each $\pi_i \in Library$
$reach_i := reach_i + participation_i * R$, where $participation_i = \frac{\#\ slots\ from\ last\ episode\ using\ \pi_i}{\#slots\ from\ last\ episode}$
accWReach := accWReach + $reach_{\varepsilon-\text{greedy}} * R$
accRewards := $accRewards + R$
$reach_{\varepsilon-\text{greedy}} := reach_{\varepsilon-\text{greedy}} + \Delta reach$
$reach_{\varepsilon-\text{greedy}} := {accWReach}/{accRewards}$
Library := Library $\cup$ { $\varepsilon-\text{greedy}(\pi_{\text{new}})$ with reference at s_{initial}, reach = $\text{reach}_{\varepsilon-\text{greedy}}$}

Table 4. The Spatial Hints for Policy Reuse algorithm

There are two points where the reach table is updated. The first is immediately after an episode, where the method can estimate each policy's contribution to the recent reward, and update their future influence accordingly. And finally, after the experiment it is possible to evaluate the average of the $reach_{\varepsilon-greedy}$ over the different rewards (${accWReach}/{accRewards}$).

4. EXPERIMENTS

The purpose of our experiment is to evaluate the performance of our algorithm against two baselines: the PRQL algorithm [1] and the Q-Learning algorithm [2,6,12]. We selected these specific contributions because we want to evaluate the relative performance of our method against approaches that require less human input. The PRQL algorithm is the method we are trying to

extend, and Q-Learning is a standard baseline, against which PRQL was first compared.

We tested each algorithm with two exploration strategies: the ε−greedy defined above and the Boltzmann strategy, where each action is chosen according to $P(a_i) = \frac{e^{\tau Q(s,a_i)}}{\sum_j e^{\tau Q(s,a_j)}}$; where τ is a parameter whose initial value τ_0 is increased by Δτ after each episode.

All three algorithms have been implemented and are available on the Web together with the supporting data. [1]

4.1 Domain

We selected the Robot Navigation domain. We made this choice in order to have a proper comparison with existing contributions. This is a standard evaluation domain in the transfer learning literature [4,5,11], and we used exactly the same specification as when the PRQL algorithm was originally evaluated [1].

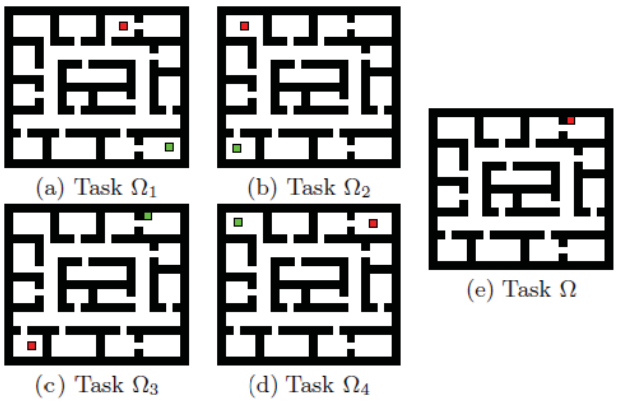

Figure 3. The tasks within the domain studied. Figures 3a-d represent the existing policies (past tasks) used by our agent. Figure 3e represents the current task we want to learn. In all of them, the red dot represents the goal state, and the green dot is the reference state (when applicable).

This domain is defined as a discreet $24\ x\ 21$ rectangle (Figure 3). The set of actions is {Left, Right, Up, Down}, which move the agent one position to the left, to the right, to north, and south, respectively. If the movement would crash the agent into a wall, then the action has no effect (i.e. the agent maintains its current position).

Five tasks are represented in Figure 3. In all of them, the red square represents the goal state of the task. Figure 3e represents the task we want to learn. Figures 3a-d represent the existing policies in our library. Their policy is the optimal policy necessary to reach their respective goal (i.e. for each state, to take the action that minimizes the shortest distance to the goal state). For these existing policies, the green square represents their reference state, whenever applicable.

It is noticeable that tasks Ω_1 and Ω_4 are very similar to the one we want to solve. Their goal state is closer to the goal of the new task, and therefore their optimal policy shares a greater percentage of decisions with the policy of Ω than the policy of the more divergent tasks, namely π_2 and more so π_3.

[1] http://people.cs.ubc.ca/~bnds/.

In all our experiments, the agent receives a reward of 1 when it reaches the goal state and 0 otherwise.

4.2 Parameter configuration

When selecting the parameters of the baseline algorithms, we used the same values as reported in [1]. Neither these nor those parameters of our algorithms are known to be optimal. In this line, a proper study exploring the parameter space of these methods in this domain remains the focus of future work.

For the Q-learning parameters, the discount factor γ = 0.95, the learning rate α = 0.05. The ε−greedy strategy was configured with ε_0 = 0.00 and Δε = 0.0005. The Boltzmann temperature was adjusted with τ_0 = 0 and Δτ = 5.

The PRQL algorithm was executed with the total number of episodes K = 2000, H = 100 steps per episode, the probability of choosing an existing policy φ = 1.00, and φ's decay rate υ = 0.95.

We configured our algorithm according to the configurations above. First, we set REPETITIONS_PER_EPISODE = 10. In order to have a fair comparison between algorithms, and since we are repeating each episode 10 times, we set our total number of episodes E = 200 (instead of the equivalent 2000 episodes of the remaining algorithms). For the same reason, since we set the maximum number of slots per episode $k = 10$, we configure the maximum number of steps per slot T = 10 (to be equivalent to the 100 steps per episode of the other algorithms).

We discuss the effects of repeating each episode in Section 5.

4.3 Results

This section reports the empirical results from our experiments. In each experiment, a different library of existing tasks was selected. For each of the algorithms evaluated (SHPR, PRQL, and Q-Learning), we initially confronted the two action selection strategies (ε−greedy and the Boltzmann strategy). However, the performance of ε−greedy dominated the Boltzmann strategy in the SHPR, and Boltzmann dominated ε−greedy with the remaining two methods. That's why we only report the dominating strategies below.

The first experiment (4.3.1) illustrates the behaviour of our method using a favourable library configuration. Only good existing policies were selected, thus contributing to a better performance of our algorithm. In 4.3.2, we introduce a bad existing policy into our library. In 4.3.3, we present the case when all policies in the library are not useful. The results are presented through the metric introduced in Equation (1). All results displayed are an average of 10 executions of each method under the same conditions.

4.3.1 Library = {Ω_1, Ω_2, Ω_4}

This experiment uses the policies Ω_1, Ω_2, and Ω_4, from those tasks represented in Figures 3a, 3b and 3d, respectively. It is noteworthy that tasks Ω_1 and Ω_4 are very similar to the task we are learning. In theory, this benefits both our algorithm and PRQL.

Results of this experiment are presented in Figure 4. An entry at this graph, say at (200, 0.02) means that the average reward of the respective algorithm after it ran the first 200 episodes was 0.02 (see Equation 1). Our algorithm collected the highest average rewards in this situation no matter what the action selection strategy. However, ε−greedy demonstrated to be the best strategy for our method. This is unlike the other algorithms, where the Boltzmann strategy always yielded better results.

In this case where the library of policies contains favourable entries, PRQL is able to outperform Q-Learning when using the Boltzmann strategy. The use of the ε−greedy strategy makes PRQL even worse than Q-Learning using that same strategy.

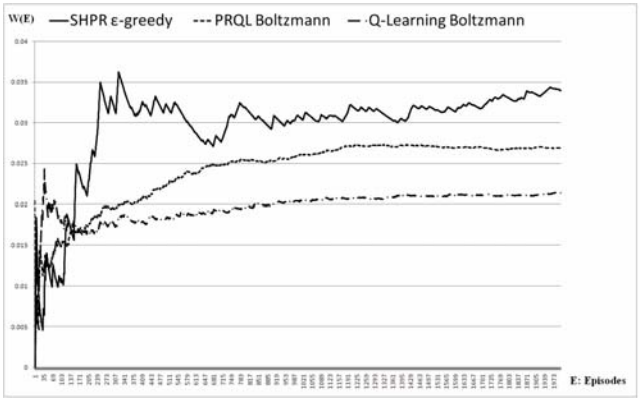

Figure 4. Comparison using existing policies π_1, π_2, and π_4. This library contains only favourable entries for both SHPR and PRQL. The legend displays the entries in order of final accumulated reward, from highest to lowest.

4.3.2 Library = {Ω_1, Ω_2, Ω_3, Ω_4}

In this experiment, we added one entry to the previous library, Ω_3, which is detrimental to SHPR and PRQL learning the current task. This entry has its goal state in the opposite corner of the state space, and therefore most of the actions it dictates are bad decisions for the learning agent. This experiment tests the behaviour of the algorithms in this more realistic scenario.

Figure 5 presents the results of this experiment. In this scenario, again SHPR performed better than the other methods. Interestingly, although it did take longer to converge, SHPR reached the same average as it did without the bad policy. The algorithm correctly tended to ignore the bad entry in the library, in favor of the better alternatives, causing hardly any damage to the final policy. This suggests robustness against bad elements in the input library of hints.

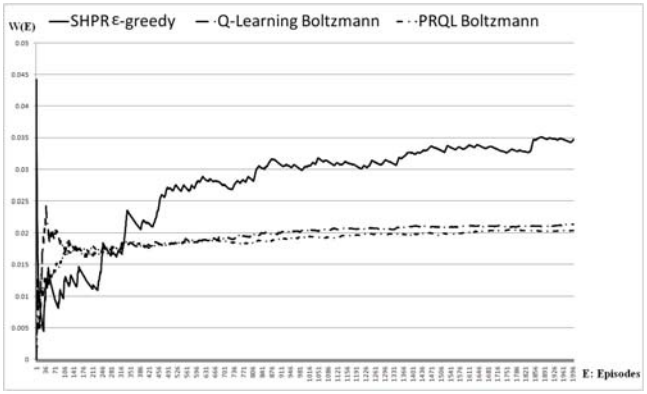

Figure 5. Comparison using existing policies π_1, π_2, π_3, and π_4. This library contains entry 3 which should be detrimental to SHPR and PRQL. The legend displays the entries in order of final accumulated reward, from highest to lowest.

Just like in the previous experiment, the use of ε−greedy yielded better results than the Boltzmann selection strategy. And again, this is the opposite of what happens with the other methods. Q-Learning managed to outperform PRQL by a small difference using Boltzmann and by a significant difference when using ε−greedy.

4.3.3 Library = { Ω_3, Ω_{3a}, Ω_{3b}, Ω_{3c} }

The goal of this last experiment is to evaluate our algorithm using a library consisting of less favourable entries. While before we had entries that made up for a bad entry in the library, now the library consists of repetitions of policy π_3, whose goal is somewhat opposed to the goal of our learning task. Figure 6 depicts this library. Figure 6a us simply a repetition of Ω_3, while Figures 3b-d introduce the same task with different reference points.

Results from this experiment are represented in Figure 7. The most intriguing result is the persistent good performance of our method relative to the others. Naturally it performed worse than with the previous two libraries, but SHPR still outperforms the other two algorithms, no matter which action-selection strategy was employed. Still, ε−greedy performed better than Boltzmann in our algorithm, but worse in the other methods. PRQL performed worse than Q-Learning, and converged to very similar results, no matter which strategy is used.

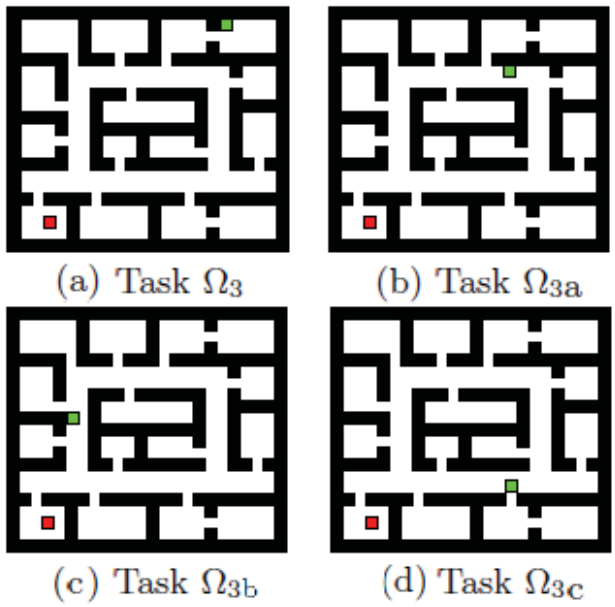

Figure 6. A new library of bad policies used in the experiment described in 4.3.3. In this experiment, we removed all the good entries from the library and create three extra copies of the bad policy, with reference states covering different areas of the world.

This last experiment really suggests a strong resistance to the use of policies that are significantly different from the current task. As is the case with the policies in this library, the only actions that benefit the agents are those inside the rooms that do not contain any of the goal states. But whenever the agent enters the corridor, these tasks take it to the direction that's opposite to where it should go. SHPR is able to reuse the subset of the policies that's beneficial to the current task.

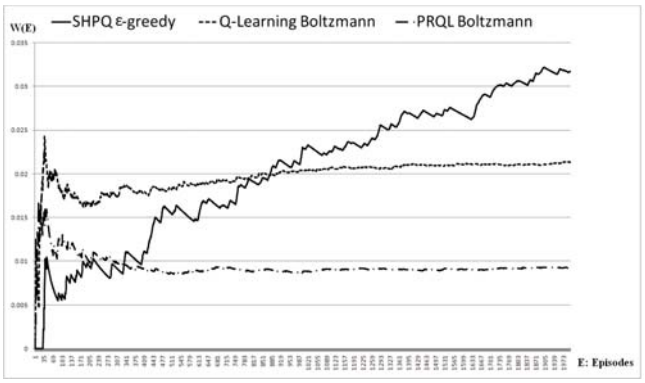

Figure 7. Comparison using existing policies π_3, π_{3a}, π_{3b}, π_{3c}. This library contains only entries that should not help SHPR or PRQL. These hints reuse the dissimilar task Ω_3 with different reference points. The legend displays the entries in order of final accumulated reward, from highest to lowest.

5. GENERAL DISCUSSION

In SHPR, one of the main departures from standard exploration of the environment is the repetition, in a sequence of episodes, of the same initial state. Naturally, if the agent must learn from a limited number of episodes, this feature of our method restricts the coverage of the state space by the agent. However, as the agent is allowed to explore the environment for a longer period of time, this deficiency fades away. For a long enough training sequence, the only difference this strategy makes (against one that randomly picks a new starting state every episode) is the order in which the state space is covered. Moreover, the subsequent episodes benefit from the more robust knowledge collected from previous episodes in our method.

As we mentioned in Section 1, our motivation for this work comes from the different methods of reusing information from past policies to learn a new task. One of the ways to classify these methods is with regards to how much information they require from humans. In this respect, it is important that our method achieves a considerable improvement in performance over others without imposing a significant burden to humans. The information we ask for is easily elicited from humans, who can reason about high-level task differences and make sense of the relation between different tasks. This kind of abstract reasoning, together with supporting task information, is what lacks current learning algorithms. Therefore, this framework seems promising to enable successful collaborations between humans and intelligent agents.

Even though we claim that the input we request from users can be easily collected, to the best of our knowledge, no metric exist which measures how difficult some piece of information can be collected from humans. Of course, this is a very difficult problem, and seems to be very challenging to define even in an ad hoc manner. However, since we are studying trade-offs between solution qualities and additional input, advances in this area might help better evaluate future contributions.

Nevertheless, it is also noteworthy how robust our algorithm is to input that in a significant subset of the state space provide a bad policy to be executed. This was demonstrated especially in Section 4.3.3. This also alleviates humans from having to construct detailed and accurate models in order to provide good input to the algorithm.

The introduction of several slots composing an episode is similar to the options framework introduced by Sutton et al. [7]. While before the agent would reuse a set of sub-policies, and follow a member of that set for the entire length of the respective sub policy, now we reuse entire policies learned in the past, and our method has the ability to interrupt the policy executed in a given slot even if that policy could provide actions for the subsequent states observed by the agent. However it is straightforward to reduce the slot scenario we presented here to the options framework.

All of our experiments demonstrated the superiority of the ε−greedy action-selection strategy over Boltzmann in SHPR. This is in contrast to the baseline methods, where Boltzmann performs consistently better. The Boltzmann strategy determines in the initial stages a uniform distribution over actions, and then converges to a more biased behaviour towards those actions that relate to higher Q values. ε−greedy, however, adopts a binary behaviour. It either selects the best action available or a uniform distribution over all actions. It might be the case that, since our method already switches from exploitation to exploration inside each slot (Figure 2), our approach is complemented by the ε−greedy behaviour. However, a careful analysis of this behaviour remains the object of future study.

The empirical studies reported here were conducted with the objective of demonstrating how our method can outperform the best known configurations of PRQL. However, neither the configuration of our SHPR nor that of PRQL is known to be optimal. This is specially the case with our algorithm, whose configuration was determined by the search for a fair comparison with the reported best configuration of PRQL, as the first principle. Therefore, an empirical study evaluating the behavior of SHPR with different parameters is the object of our current efforts.

Another important empirical question has to do with the question of convergence of our methods. In all our experiments, it is not clear what would be the long term performance of SHPR, had longer trials (or episodes) been recorded. From initial results, it seems like stagnation is not reached with the small number of episodes reported here, but a definite conclusion to this question cannot be reached yet.

Another topic of future work is the evaluation of the method in different environments. We implemented this specific robot navigation domain because of previous results reporting the success of PRQL in this problem. However, the application of learning algorithms in noisy domains is of enormous practical and theoretical importance. These and other questions are all topics for future work.

6. CONCLUSIONS

This paper introduced Spatial Hints Policy Reuse (SHPR). This is an algorithm that learns to perform a task using policies from tasks that were learned in the past in the same domain. Our main contribution lies in showing how, by allowing users to specify one or more reference points for past policies, a significant improvement in performance can be reached. This is a very simple type of information that can be elicited with little cost from users. We demonstrated how our method is robust to unfavourable input, and how it extracts information from past policies that share some degree of similarity with the current learning task.

7. REFERENCES

[1] Fernández, F., and Veloso, M. 2006. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems AAMAS '06* (Hakodate, Japan). ACM Press, New York NY USA, 720-727. DOI= http://doi.acm.org/10.1145/1160633.1160762.

[2] Kaelbling, L., Littman, M., and Moore, A. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research (JAIR)*, Vol 4 237-285 DOI= 10.1613/jair.301.

[3] Maclin, R., Shavlik, J., Torrey, L., Walker, T., and Wild, E. 2005. Giving Advice about Preferred Actions to Reinforcement Learners via Knowledge-Based Kernel Regression. In *Proceedings of the Twentieth National Conference on Artificial Intelligence AAAI '05* (Pittsburgh, PA USA). AAAI Press 819-824

[4] Madden, M., and Howley, T. 2004. Transfer of Experience between Reinforcement Learning Environments with Progressive Difficulty. *Artificial Intelligence Review*, Vol 21 375-398

[5] Sherstov, A., and Stone, P. Improving Action Selection in MDP's via Knowledge Transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence AAAI '05* (Pittsburgh, PA USA). AAAI Press 1024-1029

[6] Sutton, R., and Barto, A. 1998. Reinforcement Learning. MIT Press, Cambridge MA USA

[7] Sutton, R., Precup, D., Singh, S. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence* 112:181-211

[8] Taylor, M., and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research (JMLR)*, Vol 10(1) 1633-1685

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

[10] Thomaz, A., and Breazeal, C. 2006. Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence AAAI '06* (Boston, MA USA). AAAI Press 1000-1006

[11] Thrun, S., and Schwartz, A. 1995. Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems 7 NIPS '05* (Whistler, BC Canada). MIT Press

[12] Watkins, C. 1989. Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK