COMBINING SPATIAL HINTS WITH POLICY REUSE IN A REINFORCEMENT LEARNING AGENT

by

Bruno Norberto da Silva

B.Sc., Universidade Federal Fluminense, 2005

M.Sc., Universidade Federal Fluminense, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

August 2010

Abstract

In this dissertation, 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.

Table of Contents

List of Tables

List of Figures

Glossary

RL	Reinforcement Learning
PRQL	Policy Reuse Q-Learning
MDP	Markov Decision Process
TD	Temporal Distance
TLITM	Transfer Learning via Inter-Task Mappings

Acknowledgments

I thank my advisor Alan Mackworth for being extremely supportive, friendly and amazingly open to very different research ideas. It is incredible how you manage to keep me on track without restricting the paths I choose to follow. Your generosity and helpfulness are a great lesson that I hope I can replicate one day with my own students. You are an idol.

I'm grateful for all my LCI colleagues, for the stimulating environment and support. I learned a lot from many of you. A special note goes to Prof Giuseppe Carenini who helped me with my PhD application and also contributed as second reader of this thesis.

All my family in Brazil, for all the emails, phone calls, thoughts. Thanks for all your patience, care, support. And I'm trying hard to find the time to go back and see you all again.

And most importantly, ma femme. Nao sei se devo repetir os votos ou se referencio minhas teses de graduacao e mestrado. Em todo caso, Helena fica aqui registrada como a motivacao e inspiracao deste e de todo meu trabalho.

Chapter 1

Introduction

One of the central goals of Artificial Intelligence is the engineering of competent behaviour in complex environments. The complexity of the environments that interest AI research may require different degrees of intelligent behaviour, ranging from those dealt with well by colonies of non-intelligent biological agents, passing through those which require human intelligence to cope with its challenges, and including environments where a society of humans (the ultimate intelligent agent to date) is necessary for a sufficiently good performance.

The biggest challenge for Artificial Intelligence is the impossibility of formally identifying intelligence. Even though intelligent behaviour is routinely recognized by intelligent agents (humans), the apparent impossibility to directly specify it impeded the synthetic creation of intelligent agents. Therefore, this design limitation leads to a research agenda that attempts to achieve desired behaviour indirectly by using recognition instead of specification. This area is known as Machine Learning.

In this dissertation, we focus on a sub-field of Machine Learning called Reinforcement Learning (RL). In RL, the recognition of the desired behaviour is passed to an agent in the form of a reinforcement signal. This signal follows the actions of a computational agent who is able to affect some environment. The higher the signal received by the agent, the better was the influence of the agent's actions on the environment. Therefore, if an agent is able to keep track of the states of the environment and of the actions taken between transitions of these states, then this agent can learn to maximize rewards received during a task by collecting statistics about the rewards received under certain states of the environment after some action was executed.

One formulation of RL is temporal difference learning. In this framework, the agent interprets the resulting statistic associated with a state and an action as the value of being in this state and performing this action. And the reward signal is regarded as the different between the true value of the pair state, action and the current value maintained by the agent. The agent can then update the value of state, action pairs after observing rewards using the equation:

$$Q(s_t,a) = Q(s_t,a) + \alpha[r + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a)],$$

where α is the learning rate, γ is the discount factor, and $Q(s,a)$ is the value of performing action a after observing the environment in state s. The collection $Q(.,.)$ is referred to as the Q-table.

This formulation is generic, in the sense that it can learn an optimal policy (mapping between states and actions) for every task that is expressible using a real-valued reward signal. However, the problem with this approach is that the time it takes the learning agent to reach a satisfactory performance on the task might be too long. This motivates the search for alternative RL methods.

Many approaches have been proposed to accelerate the acquisition of competent behaviour by a RL agent, notably to incorporate information other than the reinforcement signal to guide the learning agent. A popular approach is to try to reuse policies learned in the past and transform them into a useful bias for learning a new task. This approach is called Transfer Learning [11].

A fundamental challenge of Transfer Learning is to discover similarities between tasks learned in the past and the task faced in the present. However, because of the multiple ways in which similarities can occur, Transfer Learning is a very challenging problem. Existing approaches for Transfer Learning in RL domains vary considerably with regards to their settings. One feature of a Transfer Learning setting is the incorporation of human supervision of the execution of the work of the agent. Under this perspective, two important works from the literature illustrate different ways to bring humans into the loop. In [12], a human expert is allowed to create mappings between two tasks, one that was already learned and

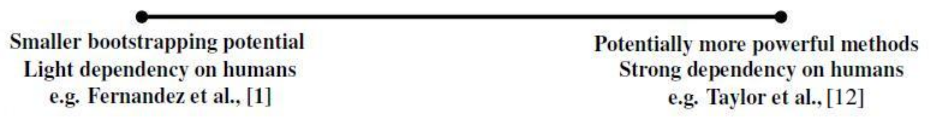
Smaller bootstrapping potential
Light dependency on humans
e.g. Fernandez et al., [1]
Potentially more powerful methods
Strong dependency on humans
e.g. Taylor et al., [12]

Figure 1.1: *A spectrum of Transfer Learning methods.*

one that is being faced by the agent. These mappings can connect states of the world between the two tasks, or they can connect actions that are equivalent in the two tasks. After eliciting this association between the tasks, the agent then initializes the Q-table of the current task based on the mapping created by the expert and the Q-table of the old task. For example, if both (s_1, a_1) and (s_2, a_2) in the learned task map to (s_3, a_3) in the current task, the value of $Q(s_3, a_3)$ is initialized to the average of (s_1, a_1) and (s_2, a_2).

In [1], the computational agent is more autonomous because no configuration of existing policy is required before the actual learning step begins. As the agent faces different tasks in its lifetime, it stores their respective policies in a library, and then uses this Library of Policies when a new task is to be learned. By combining random exploration of the environment with exploitation of both the existing policies and the policy that is being built for the current task, the learning process can be accelerated given the presence of a task in the agent's library that is sufficiently similar to the current learning task[1].

These two approaches attack the same fundamental problem, but they lie on two extremes of a spectrum (as shown in Figure 1.1). While the work in [1] relieves the agent's designer from participating in the transfer of knowledge between tasks, in [12] the contribution of a human expert is a decisive factor for the efficiency of the method. At the same time that we need an efficient way to engineer competent behaviour, it is important to note that human contributions are a scarce resource in computational systems. Therefore, this motivates the question of whether one can design a method for transfer learning that find a balance in this spectrum, i.e. a method that takes human input to improve the performance of a learning agent, but without posing an excessive burden on the provider of information.

[1] [1] reports a reduction from 20 hours of training time to 20 seconds in order to keep the ball for 9 seconds in the Keepaway domain

1.1 Problem statement

In this dissertation, we focus on the transfer of knowledge between tasks defined on the same domain. Therefore, the policies learned in the past and the policy faced by the agent in the present share the same state and action spaces.

The goal is to use the reinforcement signal associated with the new task to generate a policy mapping states of the world into actions. Other than the reinforcement signal, humans contribute collaboratively with a computational agent in order to facilitate the learning process. This contribution takes the form of Spatial Hints, i.e. a human provides a map from the policies already learned by the agent to elements of the state space of the task. The respective state associated with a policy is called that policy's reference state, and it implies that the human believes that is the condition in which the old policy will be most similar to an ideal policy for the current task. A collection of existing policies with their reference state is called a *Library of Spatial Hints*. The central problem of this dissertation is thus how to use this Library of Spatial Hints to learn a new task in a more efficient way.

1.2 Hypotheses

Our working hypotheses observe the performance of our method using a Library of Spatial Hints. In this work, we used two baseline methods: the first is the traditional temporal-distance Q-learning, which does not incorporate any extrinsic information, only the exploration of the state space of the task. The second is PRQL [1], which uses a Library of Policies to improve learning of the new task. Our null hypotheses are the following:

$H_0a : \mu_1 = \mu_2$,

$H_0b : \mu_1 = \mu_3$,

Where μ_1 is the mean performance transfer learning method that uses a Library of Spatial Hints to perform transfer of knowledge between tasks, μ_2 is the mean performance of the PRQL method and μ_3 is the mean performance of the Q-learning method.

The alternative hypotheses are the following:

$H_1a : \mu_1 > \mu_2$,

$H_1b : \mu_1 > \mu_3$.

1.3 Premise

Our premise is that the interaction between the human who is constructing the Library of Spatial Hints and the learning agent is collaborative. In other words, while one can easily construct an adversarial input that will deteriorate the performance of our method, we assume that it is against the human's interest to do so. Nevertheless, we do not assume perfect human knowledge of the relation between the old tasks and the current one. In other words, we will address conditions in which the human provides associations between previous tasks and points in the state space in which the respective task will contribute with detrimental knowledge for the performance of the current task. Robustness to such conditions is important because otherwise the effort to engineer a good enough input could be overwhelming.

1.4 Research Method and Evaluation Metric

Our research method will consist of empirical evaluation of the proposed algorithm in different scenarios. We define a scenario as a specific construction of a Library of Spatial Hints or a Library of Policies, depending on the algorithm in question. Each scenario varies in the quality of the input, from one that is constituted of beneficial tasks that are very similar to the current learning task, to one is which the existing tasks are very different from the current task.

The evaluation of our method will be defined over the resulting policy produced by the learning agent. The metric considered will be the sum of discounted rewards accumulated during the learning period. This is a metric commonly used in the literature, and helps evaluate the final policy produced by the algorithm as well as the learning period. The evaluation of the learning period is especially relevant if one considers the learning stage as relevant to the performance of the agent, i.e. if the agent is not training in a fictitious environment, but already in a realistic one in which its actions are already relevant from the point of view of the agent's designer.

1.5 Outline

In Chapter 2, we will review the literature and highlight the most important work that is relevant to our contribution. Chapter 3 formally introduces our method and positions its contributions to the literature surveyed in the previous chapter. The empirical work that validates our contribution will be presented in Chapter 4, while Chapter 5 ends this dissertation with a general discussion and examines possibilities for future work.

Chapter 2

Related work

In this Chapter, we will discuss some of the relevant works from the literature, including contributions to the design of reinforcement learning agents, and different approaches to accelerate the generation of satisfactory policies.

One important contribution to the design of satisfactory high-level behaviour in complex environments is the Layered Learning approach introduced by [7] for the Robocup domain. Layered learning was developed as a hierarchical, learned solution to tasks that are sufficiently complex that cannot be solved by hand-coded policies. Features such as limited communication, real time decision making and noisy environments in a multi-agent setting all contribute to the impossibility of solving tasks with manually created policies, and to the difficulty of applying standard learning techniques that deal with a direct representation of data collected from the noisy sensors of the agent.

The layered approach consists of breaking the agent design into behavioural modules, allowing for independent development that avoids the full domain complexity. Layers are defined by the agent's designer, taking into consideration the domain of action and the Machine Learning techniques available.

The first challenge in the implementation of this technique is to define a bottom-up task decomposition that will enable Machine Learning algorithms to treat the output of the learning process in one abstraction to be used as a sub-routine in another abstraction of a higher level.

Table 1 (Table 4.2 in [7]) illustrates an example of the task decomposition for

Table 2.1: *The design of a layered task decomposition for the Robocup domain. Layers are defined bottom-up- to cope with the domain complexity. Policies defined in lower layers serve as primitives for the behaviour of higher-level layers.*

Layer	Strategic Behaviour	Behaviour Type	Example
1	robot-ball	individual	ball interception
2	one-to-one player	multi-agent	pass evaluation
3	one-to-many player	team	pass interception
4	team formation	team	strategic positioning
5	team-to-opponent	adversarial	strategic adaptation

Robocup. In this domain, agents must learn to play a soccer game in collaboration with 10 agents, against an adversarial team. The challenges of the task make it impossible to develop a satisfactory policy for the task manually. Therefore, this domain emerged as a very popular test bed for Machine Learning techniques.

Layers are defined bottom-up, in the sense that the agent designer first focuses on the most elementary aspects of the task, looking for the simplest behaviour that must be followed by a satisfactory policy. Next, these behaviours serve as guidance for the design of more abstract patterns of an ideal policy.

This bottom-up approach ensures that the agent can cope with the domain complexity. In each layer, the agent designer can make an independent choice of the specific technique that will better fit the conditions of the respective sub-task (e.g. a supervised learning technique, a reinforcement learning setting, or even a direct specification of the policy, in those lower-level layers where the sub-task is sufficiently simple.)

With regards to Machine Learning techniques, our main focus in this thesis is on reinforcement learning. In this setting, an agent interacts with an environment in discrete steps. In each step, the agent observes a state of the world $s_t \in S$ and must choose an action $a \in A$. In response to an action, the environment produces a numerical reward r and the next state of the world $s_{t+1} \in S$.

The environment is an MDP consisting of a set of states S, a set of actions A, a transition function $P : S \times A \times S \rightarrow [0,1]$ defining the probability of seeing a given state $s_{t+1} \in S$ after executing an action $a \in A$ at state $s_t \in S$, and a reward function

$R_a(s_t, s_{t+1})$. The assumption is that the learning agent does not know the system dynamics P or the reward function R. The objective of a reinforcement learning agent is to generate a policy $\pi : S \to A$.

A common metric for the evaluation of reinforcement learning algorithms is the sum of discounted rewards $\sum_{t=0}^{\infty} \gamma^t r_t$. But this alone is not enough to illustrate the usefulness of a method. Aspects like the kind of extrinsic information used to improve learning, or the performance of the method in restricted durations are also of general importance. This motivates the introduction of different algorithms for the same problem. In this Chapter, we mention three that are distinguished mostly regarding the kind of input that is asked from users. In the Options framework [9], a generalization of the action space is introduced by allowing the agent designer to define sequences of actions. These sequences can be thought of as macros that an expert knows will be useful for a given task. Later, we will introduce Policy Reuse [1]. In this method, the agent is restricted to atomic actions in the environment, but is now allowed to consult policies learned in the past for different tasks. And finally the last method is called TLITM [12]. Here, the agent elicits from an expert a manual mapping between a pair of tasks. This map allows the agent to transfer knowledge from a task that was learned in the past into a new task that is being faced in the present. These three approaches are the most relevant existing work related to the contributions of this thesis.

2.1 The Options framework

The MDP framework introduced above models the influence of the agent in the environment as the set of actions A, and this imposes the use of a single level of abstraction for the entire learning problem. Unfortunately, this might be a considerable restriction for the design of the learning problem. Humans are used to thinking in multiple levels of abstraction, and if the MDP is created with low-level actions, it might take too long for the agent to adopt a satisfactory behaviour, while high-level actions can be adopted at the expense of the efficiency of the final policy.

This problem motivated a search for a more flexible representation of actions in a reinforcement learning domain. The Options framework introduced then a generalization of an MDP by replacing the set of actions A with a collections of options in the form $o \equiv < I, \pi, \beta >$, where $I \in S$ represents the states of the world in which the option o can start, $\pi : S \times A \rightarrow [0,1]$ represents the (stochastic) policy followed during o, and $\beta : S \rightarrow [0,1]$ is the probability that o terminates after the next state is revealed.

It is easy to see that the standard formulation presented above can be achieved by defining for each primitive action an option that follows that action in every state and always ends with certainty regardless of the next state. And since the integration of an MDP with options generalizes to a Semi-MDP, the value functions and Q-table can be defined in terms of states and options, instead of states and actions. However, this does not impede an agent who is making choices over options to learn the values of the actions that compose each option. If an option A is executed, then the following is observed:

$$s_0, a_0, r_0,\ s_1, a_1, r_1,\ \ldots,\ s_n, a_n, r_n$$

where $s_0 \in I_A$ and $a_0, a_1, \ldots a_n$ were generated by following π from the respective states. Therefore, the temporal distance update can be executed after each step of the option (and not only after it is finished), allowing the agent to learn the holistic value of the sequence, but also the individual value of each of its components.

The main contribution of the Options framework was a more flexible knowledge representation, allowing the designer of the learning agent to represent actions in different levels of temporal abstraction. On the other hand, there is still a burden on the designer to study the task and know which options to make available to the agent. Even though detrimental options will tend not to be used as the agent exploits its Q-table, they still degrade the performance of learning the new task.

2.2 Policy Reuse (PRQL)

A different approach to accelerating the performance of a reinforcement learner is Policy Reuse Q-Learning (PRQL) [1]. Unlike the Options framework, this work builds on a traditional MDP model. Additionally, it allows the agent to exploit policies learned in the past in order to accelerate learning of a new task. Here, it is assumed that all policies faced by the agent are defined in the same state space, and the same action space.

The idea is that if the existing policies are sufficiently similar to the task faced by the learning agent, then these policies can induce a more efficient exploratory behaviour, allowing the agent to experiment more with those actions that result in higher rewards.

Unlike the options framework, in PRQL there is no need for the agent designer to specify anything other than the basic MDP parameters. In each episode of this method, the learning agent makes a choice between exploiting one of the existing policies learned in the past or the knowledge already accumulated in its Q-table. Either way, a random exploration is combined with this choice to allow a temporal distance method to refine the Q values by executing actions and collecting rewards from the environment. In other words, even if the agent is following the actions dictated of an existing policy designed for a different task, the learning agent still tracks state transitions and rewards from the environment. Therefore, the existing policies can be seen as an exploration bias to the learning agent.

2.3 Transfer Learning via Inter-Task Mappings (TLITM)

One limitation of the PRQL method is the necessity to directly apply existing policies into the episodes of the new task. This limits the knowledge reuse between policies defined over the same state space and action space. However, the ultimate goal of transfer learning is to be able to transfer knowledge between arbitrary pairs of tasks that may not share any MDP component.

This is the main advantage of *Transfer Learning via Inter-Task Mappings (*TLITM*)* [12]. In this method, the association between two tasks is made manually by a specialist in the domains of both tasks, who maps states of the task which the agent still needs to learn to states of the task that was already learned before. Similarly, actions of the current task are mapped to equivalent actions of the previous task.

From this mapping, the agent has for each pair state-action a set of equivalent state-action pairs in the reference task. Therefore, the Q-table of the current task can be initialized based on this information, where the value of each entry is the average of the values of the equivalent entries of the already learned task.

What TVILM provides is in fact an initial bias for a learning task. The transfer will be beneficial if a learning algorithm that starts with this carefully generated bias learned more effectively than if another alternative initialized of the Q-table was used. After the initialization of the Q values, a Temporal Distance (TD) learning algorithm will exploit this bias and explore the environment, accumulating rewards and updating value estimations for states and actions, and consequently refining the policy for the current task[1].

Of course, the success of this method depends heavily on the quality of the mappings between states and actions of the tasks. The robustness of this approach to errors made in this mapping is yet to be studied, but it seems like the effort necessary to provide this method with a sufficiently detailed mapping requires expert knowledge that may be too hard or costly to elicit from a human.

Chapter 4 includes experimental evaluation of our contribution. PRQL was chosen as a baseline method in the evaluation, and therefore empirical results are presented there together.

[1][12] demonstrates an improvement of 43% in the 5vs4 version of the Keepaway task, when transfering from policies generated from the 4vs3 and 3vs2 versions, in comparison to learning the 5vs4 directly *tabula rasa*.

Chapter 3

Spatial hints

3.1 Problem definition

In this chapter we present the main contribution of this thesis, called Spatial Hints from Policy Reuse. This is a method for transfer learning in Reinforcement Learning domains. In this sense, we envision an agent that learns new tasks characterized by a reinforcement signal.

In this thesis, we focus on transfer between tasks that share the same state- and action-spaces. This is applicable, for example, to an agent that acts in the same room, but must execute different activities across its lifetime.

This agent faces tasks that are presented through time, and it's important that once a new task is presented to the agent, they are learned as fast as possible. This means that the agent should be fast in generating a satisfactory policy for tasks given limited training time. For this reason, we want our agent to be able to use any resource that might accelerate the formation of such a policy. In this thesis, we explore the problem of policy reuse for transfer learning, therefore we want our agent to reuse the policies it learned in the past to generate a policy for the new task faster.

3.2 Formalization of the method

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, $R : S \times A \rightarrow \Re$ is a reward function, and $T : S \times A \times S \rightarrow [0,1]$ is the transition function. T and R are unknown to the agent.

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, $T : S \times A \times S \rightarrow [0,1]$.

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 $\pi : S \rightarrow A$ assigns one action for each member of the state space of some domain.

Definition 3. A task Ω is a tuple $< D, R_\Omega >$, where D is a domain, and R_Ω is a reward function, $R : S \times A \rightarrow \Re$.

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 $< \pi_h, s_h >$, 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 $h_1, \ldots, h_n$. $\exists D$, such that each hint $h_i \in L$ solves a task $\Omega = < D, R_\Omega >$.

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.

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

Definition 6. 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 7. 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

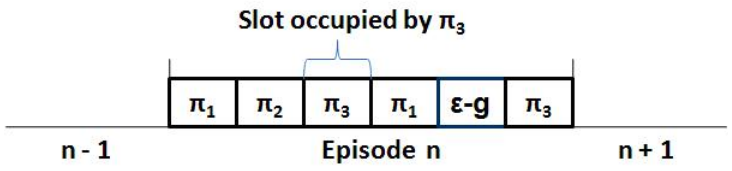

Figure 3.1: *We define an episode as containing k slots. Each slot is occupied by a policy (or an exploration strategy such as $\varepsilon - greedy$) that will dictate the actions to be taken in the steps of this slot.*

after reaching the maximum number of slots k or when the goal state is reached.

As illustrated in 3.1, 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 [10].

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:

$$W(E) = \frac{1}{E} \sum_{e=0}^{E-1} \sum_{k=0}^{k_e-1} \sum_{t=0}^{t_k-1} \gamma^{k+1} r_{k,t,e} \tag{3.1}$$

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 $r_{k,t,e}$ is the reward received in step t of slot k of episode e.

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 of 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:

$$w_i = \frac{reach_i}{1+distance(currentstate, referencestate_i)}$$

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 $\varepsilon - greedy$ procedure based on the Q-values of the current task:

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

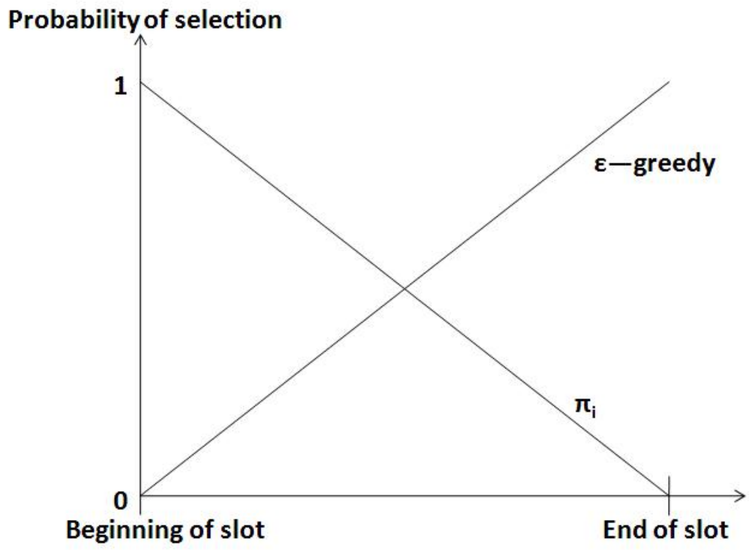

Figure 3.2: *Action selection process in a slot combining an existing policy π_i and an $\varepsilon - greedy$ procedure.*

The action selection is initially determined by π_i. In subsequent steps, there is a probabilistic balance between actions by π_i and $\varepsilon - greedy$, until the end when $\varepsilon - greedy$ dominates the action selection process (Figure 3.2). The rationale behind this choice is that the longer a policy has been in use, the farther away from its reference state the agent will be. Therefore, the learning algorithm gives increasingly more space to alternative knowledge sources.

Table 3.1 describes the algorithm executed during each slot. After the policy for that slot 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 $\varepsilon - greedy$ 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 3.1: *Slot exploration strategy.*

```
Algorithm $slot(\pi_i, s_{initial})$
$p = 1.00$
$s_{curr} := s_{initial}$
$epsilon := \varepsilon_0$
Repeat $T$ times or until $s_{curr} = s_{goal}$
    $action = \begin{cases} \pi_i(s_{curr}) & \text{with probability } p \\ \varepsilon - greedy(\pi_{new}(s_{curr})) & \text{with probability } 1-p \end{cases}$
    Execute $action$, collect $s_{next}$ 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 3.2: *Episode definition strategy.*

```
Algorithm episode( $_{initial}$, $Library$ )
$s_{curr} := s_{initial}$
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 $\pi$ according to distribution $p(\pi_i) = \frac{w_i}{\sum_j w_j}$
    Execute $slot(\pi, s_{curr})$
    Retrieve new $s_{curr}$ from slot procedure
```

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

Table 3.2 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.

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 3.2). 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 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 3.3.

Finally, Table 3.4 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 3.3. Next, the algorithm needs to determine how many times the

Table 3.3: *Abstract definition of SHPR.*

```
Abstract version of SHPO
Repeat
 Pick a state from the state space
 Repeat the question
  For this state, are the existing policies good 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.
```

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.

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 ($accWReach/accRewards$).

3.3 Discussion

As mentioned before, our motivation for integrating past policies with spatial hints is that we believe it is easy for humans to associate pairs of tasks by their spatial relevance. In other words, it should be easy to pinpoint a specific element of the state space in which an existing policy and the new task are most similar.

The introduction of spatial reference points allows our algorithm to combine it with a measure of success for each hint ($reach_x$) to create a probability distribution over hints. Naturally, this concept is technically independent of humans, and an algorithm could be employed to learn automatically the appropriate initialization of the reference points. On the other hand, it's possible that the overhead of searching for the appropriate reference points might be too much compared with the jumpstart provided to the actual learning of the new task. The answer to this question, however, is left as future work. Regarding this thesis, we note that a good

Table 3.4: *The Spatial Hints for Policy Reuse algorithm.*

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

initialization provided by a human will allow the agent to reason about the relevant of different hints given a state of the world, and to focus its computational power on learning the actual task.

The options framework replaces the reuse of existing policies with the application of macro actions (options). This is a good choice of knowledge representation, but it is more sensitive to the problem known as the knowledge acquisition bottleneck. Instead of relying on an engineer to provide sequences of actions to the design of an agent, we choose a more natural source of abstraction over actions: past policies. Moreover, since by design the execution of past policies is broken down into fixed-length slots, these slots can be seen as fragments of the existing policies, and therefore as macro actions too. And since we control and update the estimated value of each policy around the existing policy's reference point, we can say that our method learns the useful options that emerge from existing policies and their reuse throughout the lifetime of the computational agent. For a knowledge engineer to manually examine these policies and identify the useful ones would be a tremendous effort due to the infinite number of different possible options that can be constructed from existing policies.

The knowledge acquisition bottleneck also helps to illustrate the different between our contribution and the PRQL and TLITM methods. We already discussed in previous chapters how they differ in their use of human knowledge. The difficulty in eliciting expert knowledge highlights the importance of finding ways to extract better performance from methods that rely less on human collaboration.

Chapter 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 [3, 8, 14]. 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 $\varepsilon - greedy$ strategy defined before 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 $\Delta\tau$ after each episode.

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 [5, 6, 13], and we used exactly the same specification used when the PRQL algorithm was originally evaluated.

This domain is defined as a discreet 24×21 rectangle shown in Figure 4.1. 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

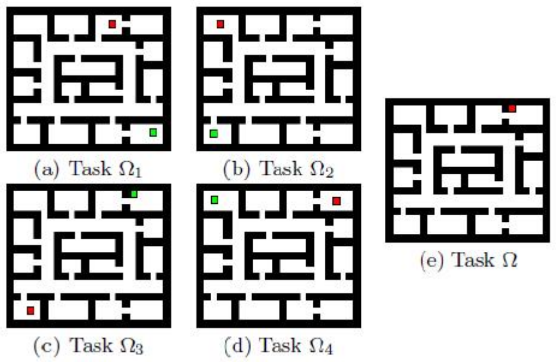

Figure 4.1: *The tasks within the domain studied. Figures 4.1a-d represent the existing policies (past tasks) used by our agent. Figure 4.1e represents the current task we want to learn. In all of them, the red dot represents the goal state, and the green dot is the reference state (when applicable).*

position).

Five tasks are represented in Figure 4.1. In all of them, the red square represents the goal state of the task. Figure 4.1e represents the task we want to learn. Figures 4.1a-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.

In all 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 started evaluating the values reported in [1]. For the Q-learning parameters, the discount factor $\gamma = 0.95$, the learning rate $\alpha = 0.05$. The $\varepsilon - \textit{greedy}$ strategy was configured with $\varepsilon_0 = 0.00$ and $\Delta\varepsilon = 0.0005$. The Boltzmann temperature was adjusted with $\tau_0 = 0$ and $\Delta\tau =$

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 $\varphi = 1.00$, and φ's decay rate $\upsilon = 0.95$.

We configured our algorithm according to the configurations above. First, we set $REPETITIONS\_PER\_EPISODE = 10$[1]. 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).

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 ($\varepsilon - greedy$ and the Boltzmann strategy). However, the performance of $\varepsilon - greedy$ dominated the Boltzmann strategy in the SHPR, and Boltzmann dominated $\varepsilon - greedy$ with the remaining two methods. Thats why we only report the dominating strategies below.

The first experiment 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 20 executions of each method under the same conditions.

[1]Naturally, the configuration of this and other parameters depends on the domain in which the agent will act. A detailed study of different configurations and their effect on the learned policies is left as future work

4.3.1 *Library* $= \{\Omega_1, \Omega_2, \Omega_4\}$

This experiment uses the policies Ω_1, Ω_2, and Ω_4, from those tasks represented in Figures 4.1a, 4.1b and 4.1d, 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 4.2. 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 3.1). Our algorithm collected the highest average rewards in this situation no matter what the action selection strategy. However, $\varepsilon - greedy$ 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 $\varepsilon - greedy$ strategy makes PRQL even worse than Q-Learning using that same strategy.

In order to test the significance of these results, we performed a Wilcoxon signed rank test with continuity correction. We observed statistical significant at the $P < 0.01$ that our method outperforms the two baselines.

Figure 4.3 shows the number of times each existing policy was selected by the Spatial Hints methods. This picture shows, for each of the test runs, the number of times each policy has been activated. Intuitively, the most favourable existing policy from the library (π_4) should be reused more often than the other ones. Indeed, we tested this hypothesis with the Wilcoxon signed rank test with continuity correction, and confirmed it with confidence values $p < 0.01$.

4.3.2 *Library* $= \{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$

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.

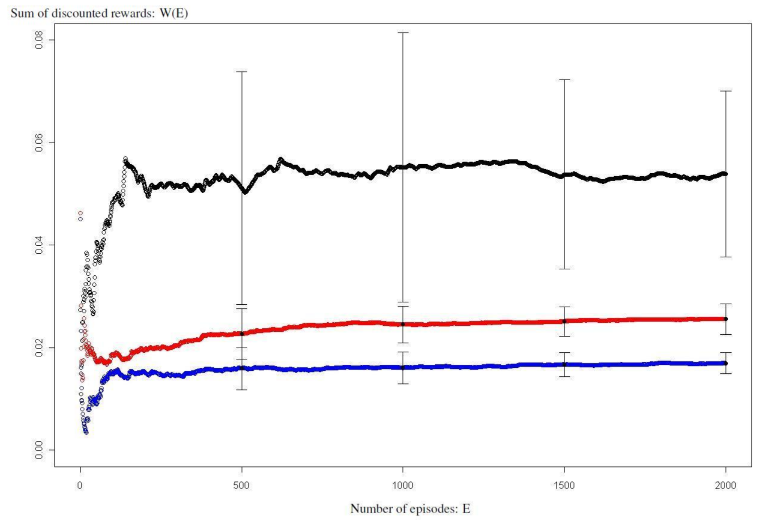

Figure 4.2: *Comparison using existing policies π_1, π_2, and π_4. This library contains only favourable entries for both SHPR and PRQL. The black curve represents SHPR using $\varepsilon -$ greedy, the blue curve represents $Q -$ learning using the Boltzmann strategy, while the red curve represents PRQL using the Boltzmann strategy.*

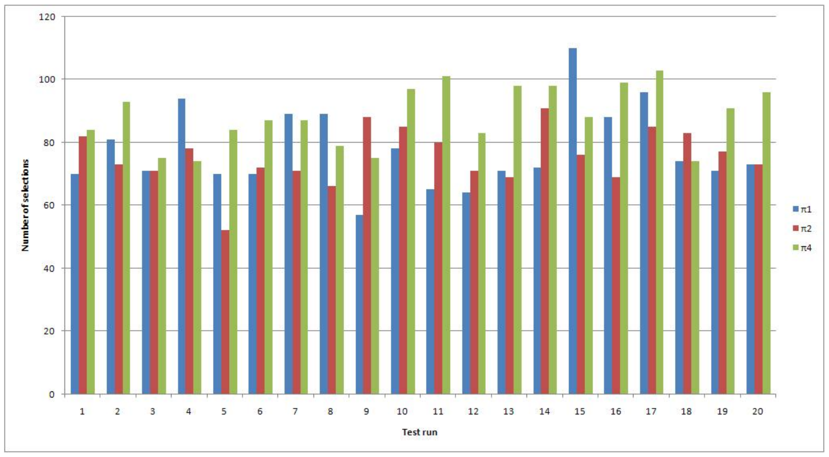

Figure 4.3: *Distribution of the use of each existing policy for each of the test runs of this Section's experiment.*

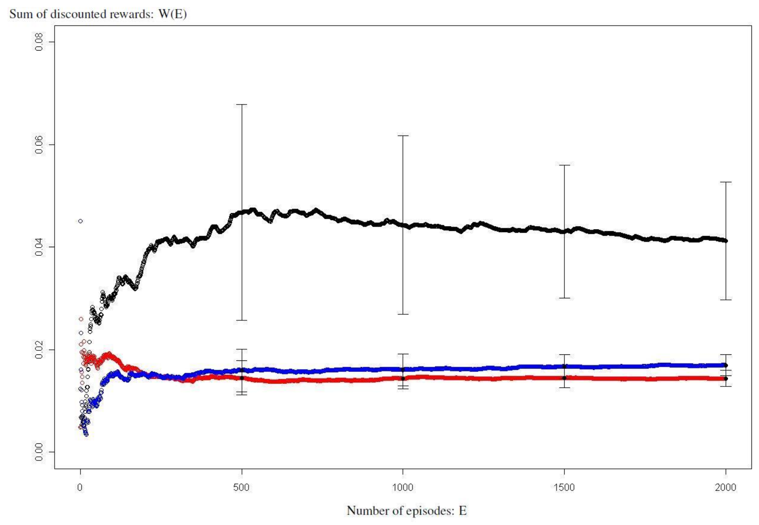

Figure 4.4: *Comparison using existing policies π_1, π_2, π_3, and π_4. This library contains entry 3 which should be detrimental to SHPR and PRQL. The black curve represents SHPR using $\varepsilon - greedy$, the blue curve represents $Q - learning$ using the Boltzmann strategy, while the red curve represents PRQL using the Boltzmann strategy.*

Figure 4.4 presents the results of this experiment. In this scenario, again SHPR performed better than the other methods. Just like in the previous experiment, the use of $\varepsilon - greedy$ yielded better results than the Boltzmann selection strategy. And again, this is the opposite of what happens with the other methods. QLearning managed to outperform PRQL by a small difference using Bolzmann and by a significant difference when using $\varepsilon - greedy$.

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. This result is demonstrated in Figure 4.5, where we display for each test run of the experiment, for number of times each existing policy has been selected by the algorithm for reuse. In evaluating this scenario, we tested whether our method was indeed selecting the less favourable policy π_3 less often than the others. However, the only statistically significant result in this matter was observed when comparing whether

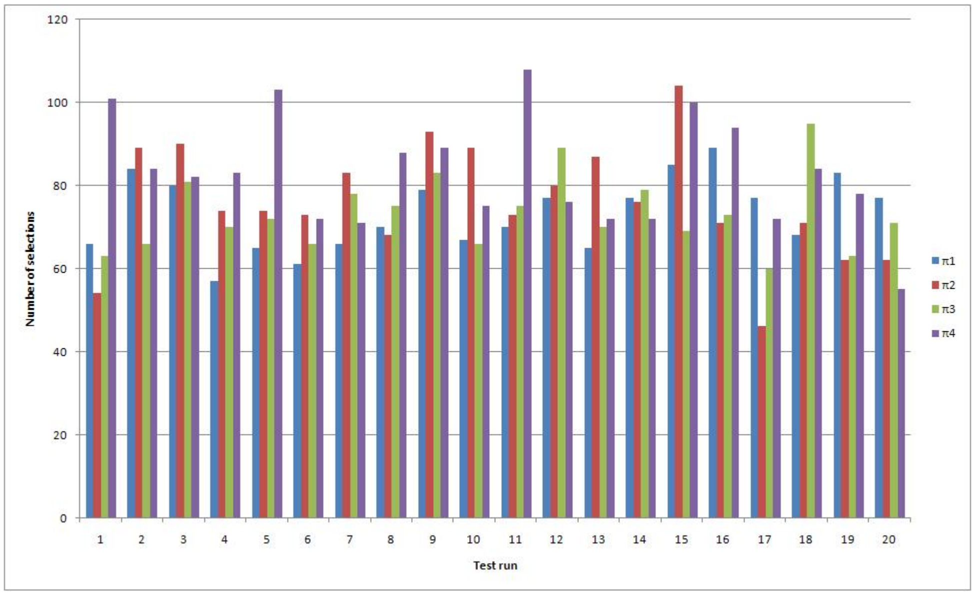

Figure 4.5: *Distribution of the use of each existing policy for each of the test runs of this Section's experiment.*

π_4 was used more often than π_3, at the level $p < 0.01$. On the other hand, it is also possible to attest that π_4 was used more often than π_1 ($p < 0.05$), but not that π_4 was used more often than π_2 ($p < 0.16$).

4.3.3 $Library = \{\Omega_3, \Omega_{3a}, \Omega_{3b}, \Omega_{3c}\}$

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 Ω_3, whose goal is somewhat opposed to the goal of our learning task. Figure 4.6 depicts this library. Figure 4.6a is simply a repetition of Ω_3, while Figures 4.6b-d introduce the same task with different reference points.

Results from this experiment are represented in Figure 4.7. 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, $\varepsilon - greedy$ performed better than Boltzmann in our algo-

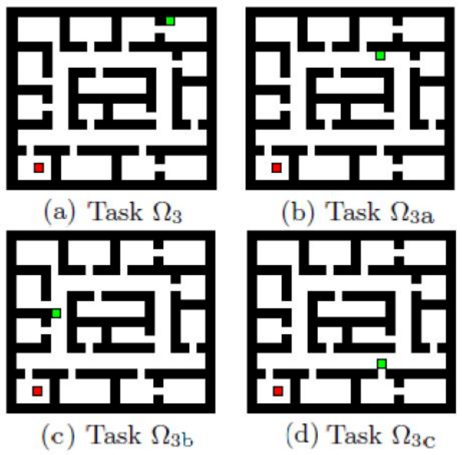

Figure 4.6: *A new library of bad policies used in the experiment. We removed all the good entries from the library and create three extra copies of the bad policy, with reference states covering different areas of the world.*

rithm, but worse in the other methods. PRQL performed worse than Q-Learning, and converged to very similar results, no matter which strategy is used.

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 thats opposite to where it should go. SHPR is able to reuse the subset of the policies thats beneficial to the current task.

Figure 4.8 depicts the use of each existing policy in each trial of this last experiment. Naturally, no policy from the library is highlighted as of special use to the method. However, we were interested in testing whether they would be considered useful in inverse proportion to the proximity of their reference state and the goal state of the task π. However, no correlation was found to be statistically significant in this respect.

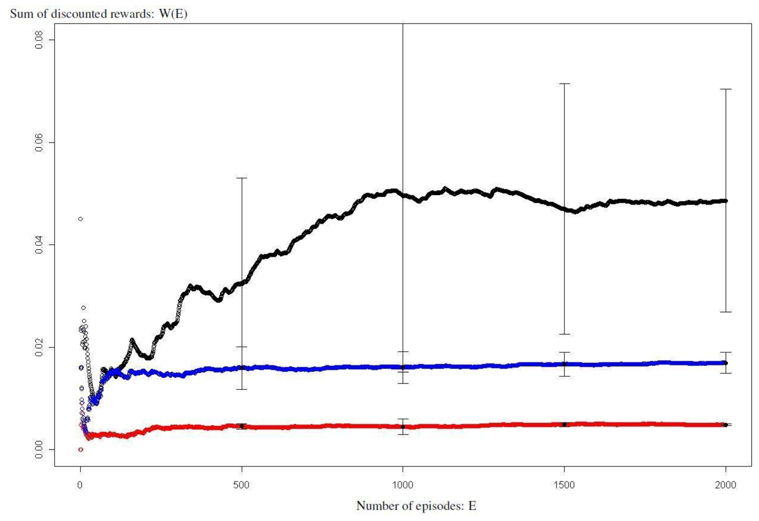

Figure 4.7: *Comparison using existing policies π_3, π_{3a}, π_{3b}, π_{3c}. This library contains only entries that should not help SHPR or PRQL. These hints reuse the dissimilar task Ω_3 with different reference points. The black curve represents SHPR using $\varepsilon-greedy$, the blue curve represents $Q-learning$ using the Boltzmann strategy, while the red curve represents PRQL using the Boltzmann strategy.*

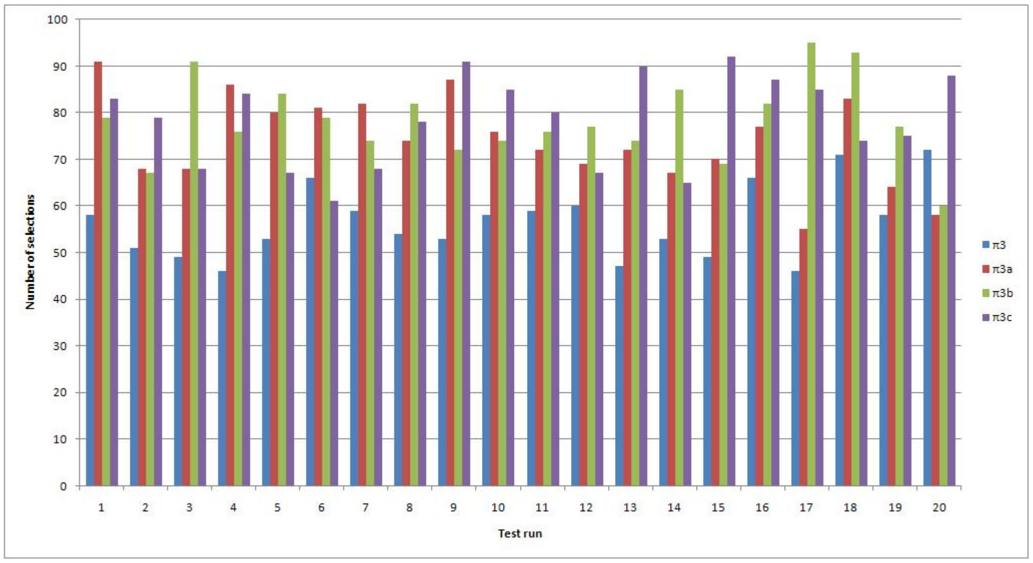

Figure 4.8: *Distribution of the use of each existing policy for each of the test runs of this Section's experiment.*

Chapter 5

Conclusion

In this thesis, we presented a new algorithm for transfer learning in reinforcement learning agents. Our method allows an agent to take advantage of tasks learned in the past to accelerate the elaboration of good policies for new learning tasks. In addition to a library of policies, we created a scheme where humans are allowed to give spatial hints to the learning agent. These hints consist of an element of the state-space for each policy in the library where the respective policy is most similar to an ideal policy for the current task. We believe that this kind of information is easily elicitable from non-expert users, and we demonstrated the impact is makes on the performance of the learning agent.

Our algorithm applies the concept of time-sharing to break episodes into different slots, allowing the agent to repeatedly sample from the library of policies during the exploration of the environment. Since the sampling is from a distribution proportional to the past success of policies, and inversely proportional to the proximity to the reference point of a policy, different positions in the state-space will lead to different choices of existing policies, and therefore to a good use of the user contribution and the successful existing policies.

In order to validate our approach, we conducted an empirical study testing the algorithm across different sets of existing policies. We also varied the type of user contribution, considering cases that characterize mistakes from the user in providing a hint that should indicate similarity between the existing policy and the current learning task. As demonstrated in Chapter 4, our algorithm is robust to inputs of different qualities, and also is able to extract similarities from tasks that at first might not be obvious to the user.

5.1 Discussion and future work

We introduced this thesis by mentioning the interest in extracting competent behaviour from learning agents. Because of this, it might be possible that the greedy nature of this agenda might produce policies that are not optimal for the learning task. That is why one important future test for our algorithm would be the study of its performance when using a set of existing tasks that are not optimal, or perhaps ill-defined in same subset of the state-space. This would certainly be a natural scenario even for other contributions to reinforcement learning where exploration is somehow limited by factors such as number of episodes or total training time.

One obvious approach to modeling the scenario with ill-defined existing policies is to replace calls for actions in areas of the state not considered by the existing policy by a call to the ε-greedy method. Althought simple, this approach might require a log of the regions actually covered by the agent when learning the past policies (i.e. the value function would not distinguish unexplored regions with choices of equivalent values). Nevertheless, it is possible that more elaborated solutions might yield better results for this problem.

One other possibility of future empirical work is the design of experiments in different domains. Since we are considering spatial hints from users, a metric over the state-space will be directly relevant to the sampling of existing policies, and therefore will have a significant effect on the overall performance of the method. Therefore, it might be interesting to test the robustnes of our algorithm in domains where such a metric would include non-linearities, or simply irregularities across the domain. One simple example is a grid-world like that considered in Chapter 4, but where each would have a heat generator that changes the temperature locally. Therefore, since walls would block the effect of heat propagation, at some parts of the state-space a manhattan model like the one considered in our experiments would produce faithful results, but not in others.

Naturally, another interesting empirical study is the evaluation of the behaviour of the algorithm in stochastic environments, where the perception of the agent and the effects of actions might be more noisy. Currently, we are testing this method in the Robocup domain [4], a domain which represents very well the challenge of stochastic environments.

From a different perspective, since we motivated our work with the idea of alleviating the burden on user of providing complicated input to the agent, it might be interesting to investigate methods that learn good spatial hints automatically, given the library of tasks and some description of the current task. Indeed, this would spare the user from providing information that is very important for the final performance of the learning agent. However, this would add to the learning task the burden of learning an important parameter of the learning algorithm. While current research demonstrates the feasibility of automatic algorithm configuration in different settings [2], it is not clear that this would be a successful approach in a reinforcement learning setting. One would naturally have to consider the overall performance of such a hypothetical method and somehow compare it with the cost of eliciting the knowledge directly from the user. Since this comparison is conceptually difficult to execute, it highlights the importance of considering human contribution to learning algorithms.

Furthermore, the high level goal of our agenda is to learn tasks fast by taking advantage of user contribution, but without burdening the knowledge contributor. It is possible that other aspects of the learning environment can be used in this respect, and not only policies learned before by the agent. A user might provide something other than points where pairs of tasks are more similar, or might contribute in a more different setting where some other type of knowledge is considered, instead of (only) tasks learned in the past.

Bibliography

[1] F. Fernandez, J. Garcia, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7): 866–871, 2010. → pages 3, 4, 9, 10, 23, 24

[2] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009. → pages 35

[3] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. → pages 23

[4] A. K. Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory and Applications*, pages 1–13. World Scientific Press, Singapore, 1993. Reprinted in P. Thagard (ed.), Mind Readings, MIT Press, 1998. → pages 35

[5] M. Madden and T. Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21:375–398, 2004. → pages 23

[6] A. A. Sherstov and P. Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005. → pages 23

[7] P. Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, Carnegie Mellon University, 1998. → pages 7

[8] R. Sutton and A. Barto. MIT Press, 1998. ISBN 0-262-19398-1. → pages 23

[9] R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999. → pages 9

[10] A. Tanenbaum. Prentice Hall, third edition, 2008. → pages 15

[11] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1): 1633–1685, 2009. → pages 2

[12] M. Taylor, P. Stone, and Y. Liu. Transfer learning for inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1): 2125–2167, 2007. → pages 2, 3, 9, 11, 12

[13] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995. → pages 23

[14] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Kings College, 1989. → pages 23