ABSTRACTION IN REINFORCEMENT LEARNING IN PARTIALLY OBSERVABLE ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

ERKİN ÇİLDEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

FEBRUARY 2014

Approval of the thesis:

ABSTRACTION IN REINFORCEMENT LEARNING IN PARTIALLY OBSERVABLE ENVIRONMENTS

submitted by **ERKİN ÇİLDEN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Faruk Polat
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Halil Altay Güvenir
Computer Engineering Department, Bilkent University _____

Prof. Dr. Faruk Polat
Computer Engineering Department, METU _____

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ERKİN ÇİLDEN

Signature :

ABSTRACT

ABSTRACTION IN REINFORCEMENT LEARNING IN PARTIALLY OBSERVABLE ENVIRONMENTS

Çilden, Erkin

Ph.D., Department of Computer Engineering

Supervisor : Prof. Dr. Faruk Polat

February 2014, 82 pages

Reinforcement learning defines a prominent family of unsupervised machine learning methods in autonomous agents perspective. Markov decision process model provides a solid formal basis for reinforcement learning algorithms. Temporal abstraction mechanisms can be built on reinforcement learning and significant performance gain can be achieved. If the full observability assumption of Markov decision process model is relaxed, the resulting model is partially observable Markov decision process, which constitutes a more realistic but difficult problem setting. Reinforcement learning research for partial observability focuses on techniques to reduce negative impact of perceptual aliasing and huge state-space. In the broadest sense, these studies can be divided into two categories. Model based approaches assume that the state transition model is available to the agent. In the model free approaches, states are completely hidden from the agent.

In this thesis, we propose methods to generalize a known sequence based automatic temporal abstraction technique –namely, extended sequence tree method– to partial observability. We attack the problem in both model based and model free approaches, showing that our methods accelerate well known representatives of each perspective. Effectiveness of our methods are demonstrated by conducting experimentation on widely accepted benchmark problems.

Keywords: Reinforcement Learning, Partially Observable Markov Decision Process, Temporal Abstraction, Extended Sequence Tree

ÖZ

KISMİ GÖZLEMLENEBİLİR ORTAMLAR İÇİN PEKİŞTİRMELİ ÖĞRENMEDE SOYUTLAMA

Çilden, Erkin

Doktora, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. Faruk Polat

Şubat 2014 , 82 sayfa

Pekiştirmeli öğrenme, özerk etmen bakış açısıyla, makine öğrenme yöntemleri arasında önde gelen bir yönlendirmesiz yöntem ailesi tanımlar. Markov karar süreci modeli, pekiştirmeli öğrenme algoritmaları için sağlam bir biçimsel temel oluşturur. Pekiştirmeli öğrenme yöntemlerinin üstüne zamansal soyutlama mekanizmaları inşa edilerek başarımlarında kayda değer artış elde edilebilmektedir. Eğer Markov karar süreci modelinin tam gözlemlenebilirlik varsayımı esnetilirse, ortaya çıkan kısmi gözlemlenebilir Markov karar süreci modeli, daha gerçekçi, ancak zor bir problem alanı tanımlar. Kısmi gözlemlenebilirlik altında pekiştirmeli öğrenme araştırmaları, algısal aynılık ve çok büyük durum uzayı sorunlarının yol açtığı olumsuz etkileri azaltacak tekniklere odaklanmıştır. Genel olarak, bu çalışmalar iki kategoriye ayrılabilir. Model tabanlı yaklaşımlar durum geçiş modelinin etmen tarafından erişilebilir olduğu varsayımına dayanır. Modelden bağımsız yaklaşımlarda ise durum bilgileri etmenden tamamen saklıdır.

Bu tezde, bilinen bir sıralama tabanlı otomatik zamansal soyutlama tekniğini (genişletilmiş dizi ağacı metodu) kısmi gözlemlenebilir problemler için genelleştiren yöntemler önerilmektedir. Probleme hem model tabanlı, hem de modelden bağımsız bakış açısıyla yaklaşılmış, önerilen yöntemlerin her iki bakış açısının önde gelen temsilcilerinde hızlanma sağladığı gösterilmiştir. Yöntemlerin etkinliği, yaygın kabul gören problemler üzerinde deneylerle gösterilmiştir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Kısmi Gözlemlenebilir Markov Karar Süreci, Zamansal Soyutlama, Genişletilmiş Dizi Ağacı

Dedicated to my dear wife, Evren.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor, Dr. Faruk Polat, for his support and proper guidance since the very beginning of my academic journey. I have always been inspired by his academic knowledge, ethical stance and enthusiasm. I also appreciate his friendly attitude that felt like working with a colleague, rather than a supervisor.

I had learned a lot from former and current attendees of the regular meetings lead by Dr. Faruk Polat. I would like to specially thank Sertan Girgin, for providing me the source codes of extended sequence tree algorithm. I appreciate very useful discussions with Utku Erdoğdu, Utku Şirin, Tuğcem Oral and many other friends.

I would also like to thank my brother, Emre. I appreciate his humorous advice, which made me remember the fact that it is possible to orchestrate or imitate randomness in a computing environment, and indirectly helped me to quickly solve a series of implementation issues.

I am grateful for the definitive support of my mother and father, Şeyda and Şinasi Çilden. As music educators and artists, they had guided me exceptionally wisely for every educational and career decision I made, and showed that this is no exception. Since I was a little kid, they have always evaluated my capabilities carefully and objectively, and taken my plans and ideas seriously. Without their wisdom and support, I would not have succeeded.

Finally, I would like to thank my colleague, best friend, and wife, Evren. Her unconditional support and confidence in me were the primary drivers of my decision to resume academic studies after a relatively long pause. We had very inspiring discussions on my studies, including possible alternative paths on deadlock situations, and made decisions together on many other technical and administrative issues throughout the whole process. Her patience, encouragement and support made this thesis possible.

This work is partially supported by the Scientific and Technological Research Council of Turkey under Grant No. 113E239.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AMDP	augmented Markov decision process representation
BEST	belief based extended sequence tree
CTS	conditionally terminating sequence
DBN	dynamic Bayesian network
EST	extended sequence tree
FSR	forbidden sub-policy repository
HA-EST	history aware extended sequence tree
HAM	hierarchical abstract machines
K-S	Kolmogorov-Smirnov
MDP	Markov decision process
MLS	most likely state
MSR	misleading sub-policy removal
NSM	nearest sequence memory
POMDP	partially observable Markov decision process
PWLC	piecewise linear and convex
RNN	recurrent neural network
SE	state estimator
SMDP	semi Markov decision process
TD	temporal difference
USM	utile suffix memory
VAPS	value and policy search

CHAPTER 1

INTRODUCTION

Besides its strong psychological and cognitive foundations on learning of mammals, reinforcement learning has a short but solid historical algebraic background in machine learning literature, which is based on trial-and-error oriented methods of learning (learning automata, classifier systems, etc.) and optimal control (dynamic programming). Reinforcement learning problem is commonly described along with the notion of a learning *agent*. This is probably not only because of the inspired branches of science, but also since agency is an appropriate model for problems for which direct external supervision is not available, where the learner is required to adapt the unknown environmental conditions alone.

There are a number of definitions for agency, depending on the level of generalization or the specific domain of problems attacked. In the broadest sense, [51] defines an agent as *"anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors"* (Figure 1.1).

[69] distinguishes two different usages of the term "agent". Weak notion of agency is defined to involve relatively more concrete properties like *autonomy*, *social ability*, *reactivity* and *proactiveness*. Weak notion of agency has its foundations mainly through software engineering perspective, and is thus used in the emerging discipline of *agent-based software engineering*. Strong notion of agency, on the other hand, has its foundations on *artificial intelligence* research, extending weak notion of agency with more *mentalistic* properties such as *knowledge*, *belief*, *intention* and *obligation* as in [53], or even some *emotional* features as in [4].

Through the agent perspective, artificial intelligence research focuses on development of methods for an agent to achieve its goals, given its beliefs. This perspective centralizes the *rationality* property, although there may be situations where there is no rational action to do, but the agent still has to come with a meaningful decision [62].

Machine learning, as a sub-discipline of artificial intelligence, focuses on building up a certain level of autonomy via methods for extracting rational (or near-rational) behaviour patterns through experience. A major distinction in machine learning is whether this experience is gathered in a supervised or unsupervised manner. In supervised learning, the agent tries to develop a behaviour map through the correct instances provided by an external expert. In

Figure 1.1: An agent interacting with the environment [51].

unsupervised learning, on the other hand, agent has no prior classified instances, thus it tries to find the regularities in the input [1].

Reinforcement learning, as a machine learning method, perfectly fits the agent based approach and lies between supervised and unsupervised methods. Since it is an on-line method, a proper definition should couple its two important aspects: a way to learn through experience (exploration) and an execution mechanism for the learned solution (exploitation).

The basic idea of reinforcement learning presumes a learner (*agent*) that interacts with its surroundings (*environment*) via its activator mechanisms (*actions*), and receives feedback (*reward*) from environment. In the general sense, what reinforcement learning methods do is, to maintain an internal situation-action mapping function by using realized reward (or punishment) information repetitively gathered from the environment, in order to make the agent more *adaptive* by means of optimizing rewards expected in the future. More formally, a reinforcement learning algorithm tries to find a policy of actions an agent ought to take in an environment so as to maximize an objective function on reward.

Reinforcement learning problems are often modelled as Markov decision processes (MDPs), under the assumption that states of the environment have *Markov property*, which means that any state retains all relevant information to derive (learn) an optimal policy of actions. In other words, decision of an agent for its next action given the current state, shall be independent of all previously experienced states. Following the agent oriented view presented in Figure 1.1, Figure 1.2 gives a gentle insight to a reinforcement learning agent in general.

Semi-Markov decision process (SMDP) model is an important extension of MDP, defining a process where the single step action assumption of MDP is relaxed. In other words, an action may take indefinite number of discrete time steps, instead of one. Fortunately, reinforcement learning methods can safely be extended to include SMDP model.

More realistic problem settings require some ground rules of MDP model to be relaxed. Partially observable Markov decision process (POMDP) is a generalization of MDP where states and state transition dynamics are no longer fully observable by the agent. POMDP defines a difficult problem category for reinforcement learning algorithms, since agent is equipped with limited *observations* instead of full state information. In other words, exact state information is *hidden* from the agent. Referring again to Figure 1.2, for MDP model, "observation"

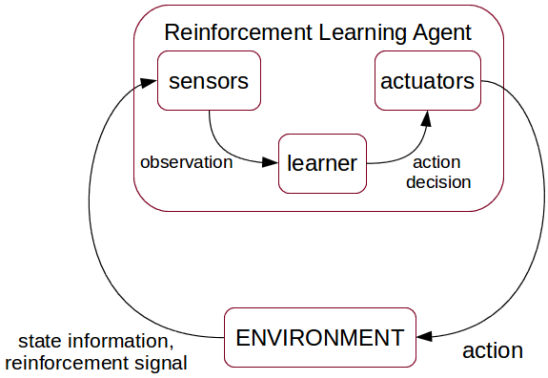


Figure 1.2: A generalized view of a reinforcement learning agent.

component of the control flow is equal to the exact state information, under the unrealistic assumption that sensors are capable of taking a complete picture of the environment's current state. For POMDP model, however, observation semantics depicts some limitations of the sensors of the agent, providing a partial and limited perception of the environment.

For many problems, there is no optimal (or even near-optimal) policy over observation semantics alone, where the agent is supposed to learn the best action to select given any observation. This kind of policies are called *memoryless* or *reactive*, and have a certain significance among researchers [15, 27, 32, 34, 45]. An obvious alternative to overcome difficulties of memoryless reinforcement learning methods is incorporating some form of memory in order to derive internal state estimations to discriminate perceptually aliased observations, which constitute another extensively studied area of research [12, 26, 38].

As a special treatment in the problem design of a POMDP learning task, the agent can be provided with the underlying MDP model. If this information is available in advance, partial observability is modelled by an internal *belief state*, which is nothing but a probability distribution over state space [28, 29]. Belief state representation of model based approach is semantically richer than observations alone. However, the resulting belief state space is very large to cope with. Existing research on belief based partially observable reinforcement learning mainly focuses on state space reduction and state prediction heuristics, since classical methods may fail to find a reasonable solution in a reasonable time, or can not find a solution at all.

Although there is no commonly agreed naming convention in the literature [15], we will refer to reinforcement learning methods for which the underlying MDP model is not available as *model free* within the POMDP context. Using the same convention, we will say that belief based reinforcement learning methods are *model based*.

Advances in reinforcement learning research raised ways to improve learning performance by means of heuristic method extensions for the fully observable setting. One of the methods is *temporal abstraction* of atomic responses of the agent, in the form of state-action sequences.

In many problems, a learning agent tries to solve a task that is in fact composed of various sub-tasks, which usually constitute a hierarchical structure. Most of the time, each sub-task repeats many times at different regions of the solution space, but the agent tries to learn each instance independently by exploring similar situations again and again, negatively affecting the learning performance, and making it difficult to converge to optimal behaviour in a reasonable time. In that sense, the main aim of temporal abstraction idea is learning faster, rather than learning a better solution.

A unified view proposes the notion of *options* [63], a widely accepted temporal abstraction model, which formulates the problem of temporal abstractions as an SMDP model and extends the theory of reinforcement learning to include temporally extended actions with an explicit interpretation in terms of the underlying MDP.

Temporal abstraction heuristics for reinforcement learning initiated by providing the agent with the abstraction information as a clue *prior* to the learning procedure [16, 43]. The idea was then extended to include *automated* ways to discover the abstraction patterns in parallel with the reinforcement learning process, which is obviously more consistent with the "on-line learning" philosophy of reinforcement learning [21, 25, 36, 39, 40, 55, 58].

Recently, few researchers attempted to use temporal abstractions to improve reinforcement learning agent to solve POMDP problems. Very few studies became frontiers of abstraction in belief based partially observable reinforcement learning [18, 64, 68], mainly focusing on non-automated or semi-automated abstraction strategies, mostly attacking the problem through sub-goal identification. Automated abstraction of memoryless partially observable reinforcement learning algorithms is even less explored [70]. To our knowledge, there are no automated abstraction studies designed for memory based partially observable reinforcement learning methods.

In this thesis, we attempt to develop ways to invoke a fully automated automatic abstraction to reinforcement learning for partially observable problems. For this purpose, we identified three main-stream families in reinforcement learning research for POMDP model, which are model based (belief based) methods, memoryless (or reactive) model free methods and memory based model free methods. We build our new constructs on a well established automatic temporal abstraction algorithm, namely *extended sequence tree* method [21], which falls into *sequence based*, or *direct* automatic abstraction category.

The outline of the thesis is as follows. Following this introduction chapter (Chapter 1), Chapter 2 summarizes the literature on which our studies are based on. Chapter 3 presents how the extended sequence tree method can be used to improve the model based methods. In Chapter 4, we switch to model free methods, and focus on enhancement of learning reactive policies through automatic abstraction mechanism. In Chapter 5, we extend the method proposed in Chapter 4 to improve a memory based model free algorithm, namely *Utile Suffix Memory* (USM) [38]. Finally we give our concluding remarks and future work in Chapter 6.

CHAPTER 2

BACKGROUND

This chapter summarizes the necessary background required to formally set the problem attacked by this thesis, highlighting the relevant topics and pointing the related publications in the literature. The decision process models are defined first; namely Markov decision process, semi-Markov decision process and partially observable Markov decision process. Afterwards, the reinforcement learning algorithms used in the thesis are summarized, including Q-Learning, Replicated Q-Learning, Linear Q-Learning, SARSA(λ) and Utile Suffix Memory. Finally, the literature on temporal abstractions is summarized, and notable methods on reinforcement learning in the partially observable case are discussed.

2.1 Markov Decision Processes

When a problem consists of several *decision problems* presented in a *sequence*, the agent needs to decide on the current action by taking into account its effects on the solution of future stages of the sequence. Moreover, the effects of the current decision is not known by the agent in advance, which introduces *uncertainty* [54].

It is often easier and more intuitive to model a decision process with the assumption of a single step dependency to the previous state. In other words, each state in the model summarizes everything about the current status of the world, without any need to refer to any of the previous states. If a process has this special characteristic, it is said to possess *Markov property* [29]. A *Markov decision process* (MDP) defines a formal framework for the sub-class of *sequential decision problems under uncertainty* having Markov property, and is formally defined as follows:

Definition 2.1. *An MDP is a tuple* $\langle S, A, T, R \rangle$*, where*

- S *is a finite set of states,*
- A *is a finite set of actions,*
- $T : S \times A \times S \rightarrow [0, 1]$ *is a state transition function such that* $\forall s \in S$*,* $\forall a \in A$*,* $\sum_{s' \in S} T(s, a, s') = 1$*, and*

- $R : S \times A \rightarrow \Re$ *is a reward function.*

$T(s, a, s')$ denotes the probability of making a transition from state s to state s' by taking action a. $R(s, a)$ is the immediate expected reward received when action a is executed in state s. ■

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action.

For an MDP, *policy* is defined to be the set of all functions π linking states of S to actions of A:

$$\pi : s \in S \rightarrow \pi(s) \in A \tag{2.1}$$

A policy defines which actions to undertake at each step of the decision process and for every possible state reached by the agent.

When a sequential decision problem can be formulated as an MDP, finding a solution turns out to be determination of the best policy. Although the methods vary according to the selected performance measure of reward accumulation (i.e. how best sequence is defined based on a function of sequence of rewards), two widely used ways to seek for good policies are *value iteration* and *policy iteration*.

Both methods are based on *value function* concept, which defines a mapping from any state s to the expected reward return, under either *total*, *discounted* or *average* criterion. Value iteration method solves the MDP problem by computing a sequence of functions converging toward the optimal policy. Policy iteration method, on the other hand, generates a sequence of improving policies.

By these methods, search for an optimal policy can be directly transformed into an optimization problem expressed in terms of value functions [54].

The MDP formalism and solution techniques based on value functions provide a basis for reinforcement learning methods, which is summarized in Section 2.4

2.2 Semi-Markov Decision Processes

MDP formalism assumes that action model consists of single step discrete actions. When this assumption is relaxed to invoke actions with temporal extension (i.e. when an atomic action may take variable amount of time) the model is named *semi-Markov decision process* (SMDP).

Formally, an SMDP generalizes the definition of MDP as follows.

Definition 2.2. *An SMDP is a tuple $\langle S, A, T, R, F \rangle$, where*

- *S is a finite set of states,*
- *A is a finite set of actions,*
- *$T : S \times A \times S \to [0, 1]$ is a state transition function such that $\forall s \in S$, $\forall a \in A$, $\sum_{s' \in S} T(s, a, s') = 1$,*
- *$R : S \times A \to \Re$ is a reward function, and*
- *F is a function giving probability of transition times for each state-action pair.*

$T(s, a, s')$ denotes the probability of making a transition from state s to state s' by taking action a. $F(t|s, a)$ denotes the probability that starting at s, action a completes within time t. $R(s, a)$ is the expected reward that will be received until next transition when action a is executed in state s; it allows rewards be received during a transition from one state to another, and computed as

$$R(s, a) = k(s, a) + \int_0^\infty \int_0^t \rho(s, a, t) dt dF(t|s, a) \tag{2.2}$$

where $k(s, a)$ is a fixed reward received upon executing action a at state s, and $\rho(s, a, t)$ is a reward rate given that the transition takes t time units [7]. ■

Obviously, MDP is a special case of SMDP with a step function having a jump at 1.

For almost any realistic MDP problem, the problem domain naturally imposes situations where the agent *switches* into a sub-policy region (i.e. skill or sub-goal) where there is no new opportunity for decision making until a number of steps passes. Importance of SMDP formalism is its ability to model such *skills* or *abstract actions* on top of MDP, by means of transition time probability function F. Most of the time, an abstract action is defined over a set of *primitive actions*. For example, in robotic soccer domain, action of *passing the ball* is composed of a series of *kick* primitive actions to position the ball and *accelerate* primitive actions to accelerate in the desired direction [59].

The temporal annotation to MDP provided by SMDP gives rise to performance improvement opportunities via temporal abstractions, or task hierarchies, as summarized in Section 2.5

2.3 Partially Observable Markov Decision Processes

It is often the case that the agent does not have enough information to infer the real current state of the decision process. It observes the process but does not know its exact state.

Partially observable Markov decision process (POMDP) model is designed to cope with this problem as a further generalization of an MDP.

Definition 2.3. *A POMDP is a tuple* $\langle S, A, T, R, \Omega, O \rangle$*, where*

- S *is a finite set of states,*
- A *is a finite set of actions,*
- $T : S \times A \times S \to [0, 1]$ *is a state transition function such that* $\forall s \in S$, $\forall a \in A$, $\sum_{s' \in S} T(s, a, s') = 1$*, and*
- $R : S \times A \to \Re$ *is a reward function.*
- Ω *is a finite set of observations the agent can experience of its world, and*
- $O : S \times A \to \mathcal{P}(\Omega)$ *is the observation function, which gives, for each action and resulting state, a probability distribution over possible observations.* $O(s, a, o)$ *denotes the probability of making observation* o *given that the agent took action* a *and landed in state* s*.*

■

Informally, a POMDP is an MDP in which the agent is unable to perfectly observe the current state. Instead, it produces an observation based on the action and resulting state [28]. In the agent perspective, POMDP framework can be viewed to provide a way to define a limited sensor model, so that the agent can only grasp the environment through certain features of the state space.

Clearly, MDP is a special case of POMDP where $\Omega = S$ and $\forall s \in S, O(s, a, s) = 1$. Flexibility of the observation function in POMDP model gives its generalization characteristics, however, this expressive power comes with its drawbacks. The model is so general that, even very restrictive or less informative observation functions can be defined. Thus, observation semantics alone is usually not sufficient to solve a given POMDP problem.

Although the agent lacks the exact knowledge of its state, it can enrich its awareness by collecting information such as history of observations and performed actions. In fact, the following information is sufficient for the agent to build a *complete information state* so that the problem satisfies the Markov property:

- the initial probability distribution over state s_0
- the history composed of all past observations and of the current observation $(o_0, ..., o_t)$
- the history of past actions $(a_0, ..., a_{t-1})$

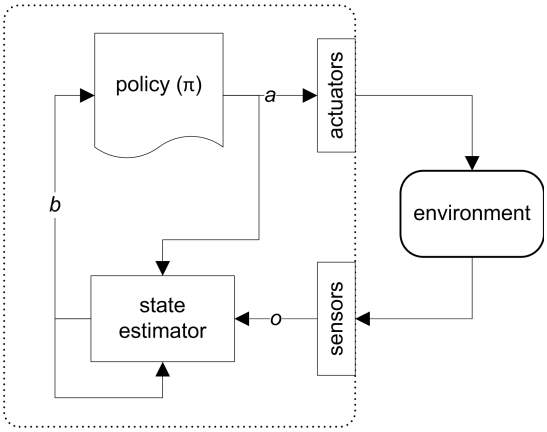


Figure 2.1: A general structural view of a belief based POMDP agent, which can be decomposed into a state estimator (SE) deriving a belief state, and a policy (π).

If an agent has the above knowledge, the information state space becomes complete and the problem turns out to be an MDP. However, in practice, the size of this information state space grows at each time step of the process, and the overall state representation quickly becomes intractable. Moreover, obviously, this solution is not applicable for processes of infinite horizon.

A widely used solution to this problem is maintaining a *belief state* as in Figure 2.1. The learning agent makes decisions based on observations and acts accordingly, satisfying the observation oriented POMDP semantics. However, it keeps an internal belief state, b, to represent all of its previous experience. The state estimator (SE) component updates the current belief state information using the last action, the recent observation, and the previous belief state. π is a conventional policy function. However, π is now a function of belief state, rather than the exact state of the world [28].

For this model to effectively preserve Markov property of the process, the belief state description should provide *sufficient statistics* relative to the control of the process, so that the complete information state requirement is fulfilled.

Belief states ensure sufficient statistics by effectively fusing the past history and the initial belief state of the agent. In other words, given the agent's current belief state, no additional data about its past actions or observations would supply any further information about the current state of the world. This makes a decision process over belief states Markovian [28, 54].

The policy component π in Figure 2.1 of POMDP agent should map the current belief state into an action. Since the belief state is a sufficient statistic, the solution of POMDP turns out to be the solution of the following special construct:

Definition 2.4. *A belief-MDP is a tuple $\langle \mathcal{B}, A, \tau, \rho \rangle$, where*

- *$\mathcal{B}$, the set of belief states, comprise the belief state space*

- *A, the set of actions, as in Definition 2.1,*
- *$\tau(b, a, b')$ is the state transition function, which is defined as*

$$\tau(b, a, b') = Pr(b'|a, b) = \sum_{o \in \Omega} Pr(b'|a, b, o) Pr(o|a, b) \tag{2.3}$$

where

$$Pr(b'|b, a, o) = \begin{cases} 1 & if SE(a, b, o) = b' \\ 0 & otherwise \end{cases} \tag{2.4}$$

- *$\rho(b, a)$ is the reward function on belief states, constructed from the original reward function on world states:*

$$\rho(b, a) = \sum_{s \in S} R(s, a) \tag{2.5}$$

■

An optimal policy for the continuous belief-MDP defined in Definition 2.4 will provide optimal behaviour for the original POMDP. What remains then, is to solve this MDP in a reasonable time. Unfortunately, solving the continuous belief-MDP is known to be very difficult in the general sense [28].

Fortunately, there is an important property of the value function for the belief-MDP, that it is piecewise linear and convex (PWLC) [57]. Thanks to this property, the value function can be represented with only a finite number of parameters, making it possible to construct a *parsimonious representation* of the value function. A famous method for constructing parsimonious representation of value functions is the WITNESS algorithm [10]. By this way, size of the continuous space MDP can effectively be reduced and exact value iteration methods can be applied.

Still, however, the above theoretical works can only be applied to toy problems and have limited impact on practical situations. Thus, algorithms that look for approximate solutions to POMDPs have been designed.

In machine learning perspective, notable studies are based on processing of the belief state to get an MDP approximation. In [23], some widely used approximation methods are summarized. Among those stated methods are grid based value interpolation/extrapolation methods, curve fitting approximations, grid based linear function methods etc., all of which can effectively be used to "downsize" the POMDP problem to a corresponding MDP approximation and make it possible to apply MDP learning methods for a nearly optimal solution.

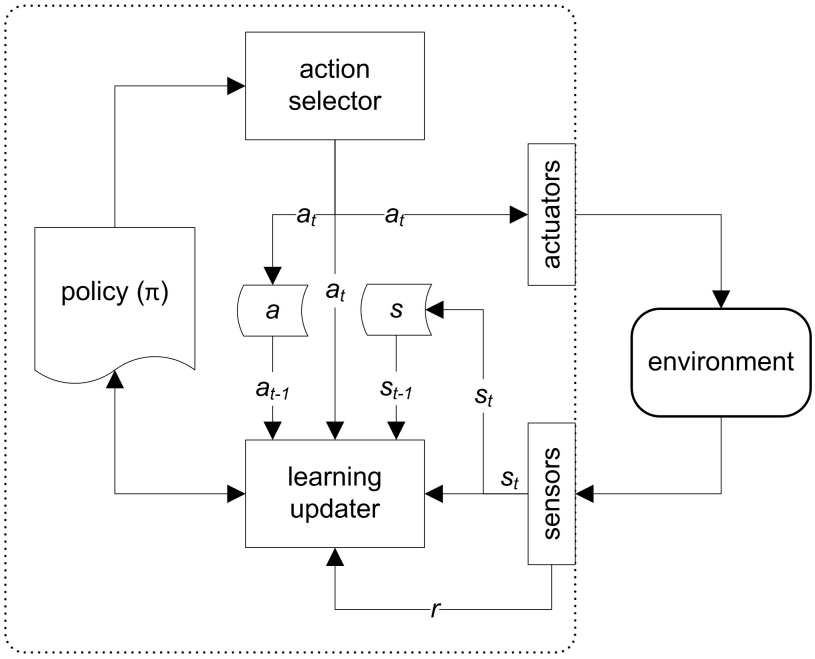


Figure 2.2: A structural view for a general reinforcement learning algorithm through a learning agent perspective.

2.4 Reinforcement Learning under Full Observability

Reinforcement learning algorithms resemble simple principles borrowed from the study of animal or human cognition, such as the increased tendency to perform an action in a context if its consequences are generally positive in that context [54].

Mathematical formalization of reinforcement learning, in machine learning point of view, is based on optimal control framework [5], through the notion of MDP [62].

The structural view of a classical reinforcement learning algorithm is given in Figure 2.2. More or less, all reinforcement learning algorithms follow this structural pattern, where the learning functionality (bounded by the dotted rectangle) is equipped with some *learning update* mechanism coupled with an *action selection* module, together with an internal *policy* database which is regularly updated through iterated invocation of these functions. Learning update engine is supplied with state (s) and reward (r) information from the environment, and the decided action (a) from the action selection mechanism. Agent interacts with outer world (*environment*) through its *sensors* and *actuators*.

Solving MDPs via the methods described in Section 2.1 (namely policy iteration and value iteration) is usually impractical for large scale and complex MDPs, due to time and space complexity issues to cope with the computing and storing of the huge state transition matrices (usually referred as *curse of modelling* and *curse of dimensionality*, respectively).

State-of-the-art reinforcement learning framework is a synthesis of topics from four different

fields: classical dynamic programming, artificial intelligence (temporal differences), stochastic approximation (simulation), and function approximation (regression, Bellman error, and neural networks) [22].

Although there are researchers avoiding to discriminate dynamic programming and reinforcement learning methods, and prefer to handle them as different instances of the same MDP solution method, some others categorize reinforcement learning to stand somewhere between dynamic programming and machine learning, inheriting characteristics of both sides. From either point of view, the main distinguishing feature of reinforcement learning from dynamic programming towards machine learning is that, reinforcement learning does not assume knowledge of problem dynamics.

In other words, a reinforcement learning agent does not know the transition and reward function of the underlying MDP in advance, so it tries to find the optimal solution by incremental estimation of the *value function* (V). Value function gives the value of being in a state on the way to goal. Alternatively, a function named Q representing the value of each action in each state can also be used. In fact, reinforcement learning attempts to formulate and solve the very same MDP problem through a different perspective, which is much more realistic for many real life problem areas, such as robotics and intelligent agent systems.

A simple way of performing the value function estimation consists of using the average cumulated reward over different trajectories obtained by following a policy π. If $R_k(s)$ is the expected utility in state s along trajectory k, then an estimation of the value function V is s based on the average after $k+1$ trajectories is

$$\forall s \in S, V_{k+1}(s) = \frac{R_1(s) + R_2(s) + \cdots + R_k(s) + R_{k+1}(s)}{k+1} \tag{2.6}$$

To avoid storing all the rewards, this computation can be reformulated in an incremental way:

$$\forall s \in S, V_{k+1}(s) = V_{k+1}(s) + \frac{1}{k+1}[R_{k+1}(s) - V_k(s)] \tag{2.7}$$

To get $V_{k+1}(s)$, one just needs to store $V_k(s)$ and k. We can even avoid storing k, using a more generic formula:

$$\forall s \in S, V_{k+1}(s) = \alpha[R_{k+1}(s) - V_k(s)] \tag{2.8}$$

where α is positive and should decrease with time. This iterative formula converges to an optimal value function:

$$\lim_{k \to \infty} V_k(s) = V^{\pi}(s) \tag{2.9}$$

The incremental estimation (Equation 2.8) is the heart of most reinforcement learning methods [54]. In the following sub-sections, reinforcement learning algorithms that have gained significant attention are described.

2.4.1 Temporal Difference Method

Temporal difference (TD) concept [60] is probably the most central idea of modern reinforcement learning methods. Under *discounted* reward assumption, temporal difference method rephrases the incremental estimation formula (Equation 2.8) to the following update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.10}$$

where α is called the *learning rate* while γ is the *discount rate*. Note that the total reward R is no longer in the scene. Instead, at time $t+1$, the agent immediately makes a useful update using the observed reward r_{t+1} to the discounted value estimation $V(s_{t+1})$, with a correction $-V(s_t)$, which is the temporal difference part. Now the agent does not need to wait for "collecting" all the rewards through the trajectory up to the goal state (or to the end of the episode). It makes updates while *learning* the qualities of visited states.

2.4.2 Q-Learning Algorithm

Q-Learning algorithm is based on the TD method, with two differences. First, it operates on state-action pairs rather than states (i.e. Q function). Second, it is not necessary to determine the action at the next step to calculate updates. The update rule for Q-Learning is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.11}$$

In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed.

Making use of this update rule, Q-Learning is one of the most important breakthroughs in reinforcement learning research [66]. Q-Learning is probably the most widely used reinforcement learning algorithm due to its simplicity. The pseudo-code of the algorithm is given in Algorithm 1.

Intuitively, Q-Learning algorithm imposes its Q-values to be stored in tables. However, for state spaces with excessive number of spanning features, this method becomes intractable, where function approximation techniques such as *artificial neural networks* can be considered instead [30].

Algorithm 1 Q-Learning

Initialize $Q(s, a)$ arbitrarily
repeat
    Let s be the current state
    repeat
        Choose a from s using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observe r and the next state s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$
    until s is terminal
until some convergence criterion is met

2.5 Temporal Abstractions in Reinforcement Learning

After gradual saturation of the reinforcement learning theory and its applications, researchers focused on finding ways to make improvements on aspects like quality of solutions, learning speed, memory space requirements, and so on. One of the notable improvement opportunities in learning speed is the determination of frequently occurring patterns in solution space, so that the agent does not need to discover the same useful sub-policy over and over again.

Although it is not easy to point a commonly agreed naming convention in the literature, the term *temporal abstraction* is used frequently to address this notion, since a sub-policy in the solution space can be viewed as an abstract action (or a macro action) executed through a certain amount of consequent discrete time steps.

Figure 2.3 is a summary of temporal abstraction methods in reinforcement learning, through a perspective that categorizes methods according to how the abstraction information is obtained by the learning agent. Before the overview of the literature on these concepts, an important definition, namely *options framework*, is provided in Section 2.5.1. Section 2.5.2 focuses on methods in which abstraction clue is provided by the designer of the agent. Finally, Section 2.5.3 describes learning of abstractions. All of the sections mentioned assume reinforcement learning in the MDP formalism.

2.5.1 Options Framework

As summarized in Section 2.2, in SMDP model, the actions are treated as black boxes, or indivisible flows of execution, which are used as they are, irrespective of their internals. All temporal abstraction mechanisms make use of state-action sub-sequences that can be used as an indivisible package, as if it is an ordinary action.

Options framework [63] formalizes this idea by extending reinforcement learning through embedding a discrete-time SMDP over MDP to include temporally extended actions with an

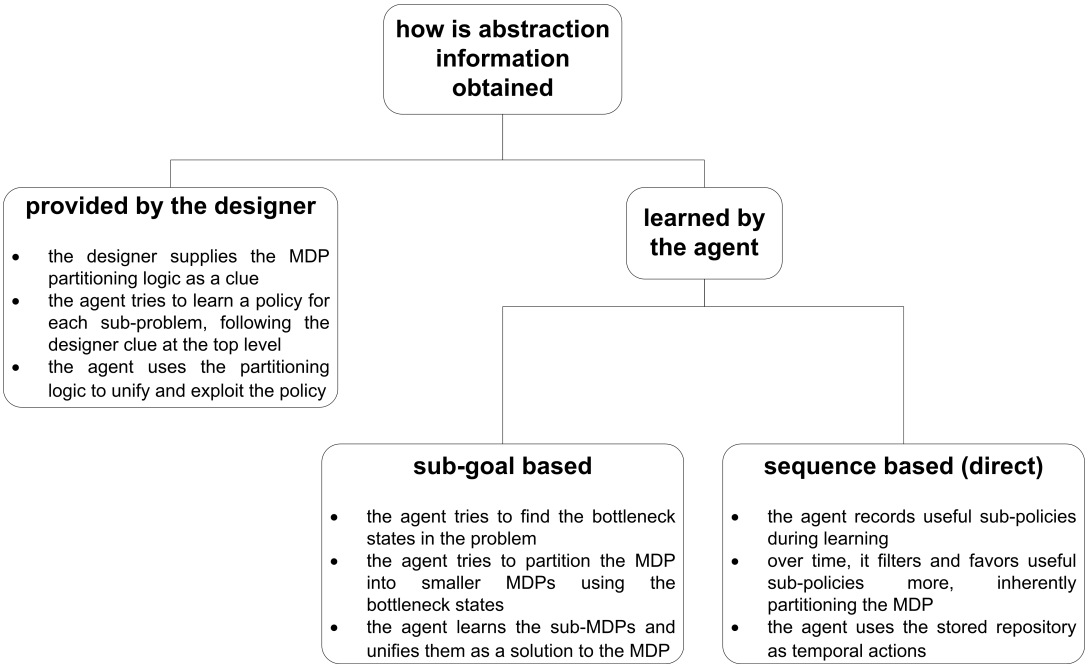


Figure 2.3: A summary of temporal abstraction methods in reinforcement learning, based on the perspective of how abstraction information is obtained by the learning agent.

explicit interpretation in terms of the underlying MDP. While keeping the unit time transition dynamics of MDPs, actions are generalized in the sense that they may last for a number of discrete time steps and referred to as *options*. An "option" is a temporally extended counterpart of an "action".

Definition 2.5. *An option is a tuple* $\langle \mathcal{I}, \pi_{option}, \beta \rangle$ *where*

- $\mathcal{I}$ *is the initiation set, which is a set of states that the option can be initiated at,*
- π_{option} *is the local policy of the option, and*
- β *is a probability distribution induced by the termination condition.*

Once an option is initiated by an agent at a state $s \in \mathcal{I}$*, option's local policy* π *is followed until the option terminates at a specific condition determined by* β*.* ■

An interpretation of option may assume that action selection inside the option is made only based on the current state, which is called *Markov option*. On the other hand, an alternative interpretation may relax this rule, so that π_{option} and/or β are allowed to depend on all prior events that occurred since the beginning of the option, which is called *semi-Markov option*. Definition 2.6 provides a basis for formalizing consecutive events, required by all methods based on options framework.

Definition 2.6. *A history is defined as*

$$h_{t\tau} = s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, ..., r_\tau, s_\tau$$

where states, actions and rewards observed by the agent are listed starting from time t until time τ. ■

Given the set of all possible histories, when β and π are defined over this set instead of S, it is possible to induce an SMDP where each action of the SMDP is an option. By this way, reinforcement learning methods applicable to SMDP can be employed by replacing the action set with the option set.

An option is essentially a fixed policy. In the options framework, the system is restricted to a set of options in every disjoint subset of the state space. In any state, one has to choose an option from the set of options available in that state. When the system enters a new subset of the state space, a new set of options becomes available. Since each option is like a fixed policy, it is composed of *primitive* actions. In a more general sense, the underlying goal in such a setting is to search over a restricted set of policies in each subset of the state space.

The Q-Learning update rule for selecting an option, θ, in a state s is as follows:

$$Q(s,\theta) = Q(s,\theta) + \alpha[r + \gamma^t \max_{\theta' \in \Theta_s} Q(s',\theta') - Q(s,\theta)] \tag{2.12}$$

where Θ_s is the set of options that can be initiated at s, $\theta \in \Theta$ (Θ being the set of all available options), and other symbols are as in basic Q-Learning. This version of Q-Learning converges to optimal Q-values for all $s \in S$ and $\theta \in \Theta$ under conditions similar to those for the Q-Learning algorithm [20].

Importance of the options framework is that it provides a formal basis for temporal abstraction methods for reinforcement learning. Furthermore, some automated temporal abstraction methods make use of the formalism constructed by options.

2.5.2 Hierarchical Reinforcement Learning

From MDP point of view, it is already known that some problems have a special structure making them amenable to a partitioning so called *hierarchical decomposition* [19]. With such a decomposition, one can essentially solve MDPs with smaller solution spaces at a lower level which supply solutions to a control optimization problem at a higher level. In other words, the overall MDP can effectively be divided into disjoint sets where an MDP is solved in each set without or sometimes with consideration of the MDP in another set. These solutions form the input to a higher level problem for which it is now not necessary to make decisions at the lower level. Considerable research has been performed in developing hierarchical methods for reinforcement learning. In fact, this idea appears to be a challenging frontier of reinforcement learning research [22].

Due to historical naming conventions, we will refer to the temporal abstraction methods that invoke the abstraction strategy as a design clue to the agent prior to the learning process

as *hierarchical reinforcement learning* algorithms. Although not directly relevant with our dissertation, in order to take a complete picture of the temporal abstraction strategies in reinforcement learning, two notable hierarchical reinforcement learning algorithms are briefly described in the following two sub-sections.

MAXQ MAXQ is a hierarchical reinforcement learning method based on decomposing the target MDP manually into a hierarchy of smaller MDPs and decomposing the value function of the target MDP into an additive combination of the value functions of the sub-MDPs of the hierarchy. It is based on the assumption that the programmer can identify useful sub-goals and define sub-tasks that achieve these sub-goals. By defining such sub-goals, the programmer constrains the set of policies that need to be considered during reinforcement learning, thus decreasing learning time.

MAXQ method makes use of a decomposition procedure that takes a given MDP M and decomposes it into a finite set of sub-tasks (or sub-MDPs) $\{M_0, M_1, ..., M_n\}$ with the convention that M_0 is the root sub-task (i.e., solving M_0 solves the entire original MDP M). The resulting graph is called the MAXQ graph. The solution is obviously a *hierarchical policy* which is a set of policies $\pi = \{\pi_0, \pi_1, ..., \pi_n\}$, one for each sub-task, that indicate how each node should choose its actions.

MAXQ method modifies Q-Learning algorithm to invoke the MAXQ graph to learn hierarchical policy, named MAXQ-Q, and is proved to converge [16].

Hierarchical Abstract Machines In the *hierarchical abstract machines* (HAM) framework, in addition to the system state and actions, one considers a hierarchy of machines, which is essentially a program, when executed by an agent in an environment, constrains the actions that the agent can take in each state. Machines for HAM are defined by a set of states, a transition function, and a start function that determines the initial state of the machine. There are four machine state types:

- *action* states execute an action in the environment,
- *call* states execute another machine as a subroutine,
- *choice* states non-deterministically select a next machine state,
- *stop* states halt execution of the machine and return control to the previous call state.

The next machine state is determined by the transition function after an action or call state as a function of the current machine state, and depending on the resulting environment state [43].

HAM method gives rise to the opportunity of reducing the problem complexity by using a formalism based on the fact that, an MDP annotated with a HAM under certain constraints

can be used to define a reduced SMDP with an optimal solution.

Thus, HAM can be used in cases where the effort required to obtain a solution typically scales too high with the size of the problem. By focusing on exploration of the state space, HAM constraints can reduce the negative impact of the *blind search* phase that reinforcement learning agents usually suffer when learning a new environment. Moreover, operating in a reduced state space can effectively accelerate learning [44].

2.5.3 Learning Temporal Abstractions

As the immediate successor of hierarchical reinforcement learning studies (Section 2.5.2), some researchers have focused on ways of *learning abstractions*, in other words, automated derivation of temporal abstractions along with the underlying reinforcement learning procedure [3]. One track of research in this area is *sub-goal based*, meaning that it focuses on identification of sub-goals, and trying to achieve a useful partitioning scheme [25, 36, 39, 55, 58]. Methods of the other track, called *sequence based* or *direct* methods, invoke common subsequence analysis of multiple successful histories, without identification of sub-goals [21, 40].

The following two sub-sections summarize the related literature, with an emphasis on sequence based methods, which is the focus of attention in this thesis.

2.5.3.1 Sub-goal Based Methods

[58] presents a statistical extension to classical reinforcement learning methods by discovering *options* automatically. In this method, a reinforcement learning agent is first allowed to explore the environment and gather statistical data. The gathered data is then used to identify potential sub-goals and initiation states. Then, the agent learns the internal policies of identified options by classical reinforcement learning methods. Finally, the agent exploits the learned behaviour. Algorithm is based on the intuition that, if states occur frequently on trajectories that represent solutions to random tasks, then these states may be important, and can be used to derive options. This approach is probably the simplest implementation for the *bottleneck state* idea. A drawback of the method is that there is an implicit assumption of tabular form representation for value function.

[39] presents a *data mining* method using which a reinforcement learning agent can discover useful sub-goals automatically by mining an ensemble of behavioural trajectories accumulated by the learning agent as it interacts with its environment. This study focuses on discovery of sub-goals by searching for *bottlenecks* in the search space. Inspired by *multiple-instance learning problem* as defined in [17], it adopts the concept of *diverse density* of [37] to discover bottleneck regions. The key observation here is the similarity of diverse density concept and the bottleneck region concept of sub-goals in reinforcement learning. A mapping is made between the maximum diverse density and the bottleneck region which the agent passes through

on multiple successful trajectories and not on unsuccessful ones.

[36] attempts to use clustering techniques on the state space for sub-task discovery in reinforcement learning. The basic idea behind the method is that, instead of taking clusters of states into account as ordinary states, they use them as *intermediate stages* in the learning process, thus define an option to be a *sub-policy* that allows the agent to efficiently shift from one cluster of states to another. The method consists of a learning part and a clustering part.

[55] attempts to identify temporal abstractions in a reinforcement learning problem by using the notion of *access states*. It defines an access state as a sub-goal state that allows the agent to transition to a part of the state space that is otherwise unavailable or difficult to reach from its current region (or *bottleneck states* as frequently named in the literature, like a doorway between two rooms). The method detects this transition using the concept of *relative novelty*, which is defined to be the measure of short-term novelty a state introduces to the agent. For this purpose, a method called *relative novelty algorithm* is defined, which identifies a sub-goal and creates a temporally extended activity that takes the agent effectively to this state. Novelty is defined as how frequently a state is visited since a designated start time. Relative novelty algorithm relies on the relative novelty of a state in a transition sequence, which is defined to be the ratio of the novelty of states that followed it (including itself) to the novelty of the states that preceded it. The key observation here is that, target states have higher relative novelty scores than the others. One shortcoming of the method is the empirical and heuristic setting of parameters of relative novelty algorithm depending on the problem domain.

[25] is a good example for task hierarchy construction via decomposition of state space into a number of nested sub-MDP regions. Decomposition is possible when

- some of the variables in the state vector represent features in the environment that change at less frequent time intervals,
- variables that change value more frequently retain their transition properties in the context of the more persistent variables, and
- the interface between regions can be controlled.

Proposed algorithm, HEXQ, uses the state variables to construct a hierarchy. For deterministic shortest path problems HEXQ will find a globally optimal policy and with stochastic actions HEXQ, as MAXQ, is recursively optimal. However, HEXQ lacks a way to automatically combine exits that result in the same next state.

2.5.3.2 Sequence Based Methods

Sequence based or *direct* way of learning temporal abstractions is a relatively less explored area, and has its basis in the notion of options framework (Section 2.5.1). Notable methods of this approach are *acQuire-macros* and *extended sequence tree* methods.

acQuire-macros One of the remarkable studies in direct learning of options is the *acQuire-macros* algorithm [40], which tries to identify common action sequences at regular intervals by

- detecting frequently occurring successful trajectories,
- eliminating similar results, and
- creating options for the remaining trajectories.

acQuire-macros algorithm is based on a special case of semi-Markov option, named *conditionally terminating sequence* (CTS). A CTS is defined as a sequence of n ordered pairs $\sigma = \langle C_1, a_1\rangle \langle C_2, a_2\rangle$ ... $\langle C_n, a_n\rangle$, where $C_i \subseteq S$ is the *continuation set* and $a_i \in A$. At step i, a_i is selected and executed, and the sequence advances to the next step if current state s is in C_i, otherwise the sequence terminates.

The most important feature of CTSs is that they can be used to compactly represent frequently occurring and useful patterns of actions in a reinforcement learning problem, that can eventually be used as options for the underlying reinforcement learning algorithm.

However, since a CTS represents a linear flow of execution, it cannot be used to represent situations in which different courses of actions may be followed depending on the observed history of events. Many real-life problems, however, inherently contain sub-tasks for which reuse of learned abstractions build-up a solution hierarchy for the overall problem, obviously through a number of decision points.

Extended Sequence Tree Method Extended sequence tree (EST) method [21] improves the acQuire-macros algorithm, by transforming useful histories into a tree data structure, in order to make it possible to incorporate conditional branching in action selection, and make use of available abstractions in a more compact and effective way.

The method is based on memorization of successful sub-policies, derived from recorded histories. A typical simplified outline of a reinforcement learning algorithm augmented by the EST abstraction method is given in Algorithm 2.

There are three main components of EST.

- The first one is the EST data structure, which serves as an option repository in the form of a tree whose edges are labelled with actions, and nodes contain continuation sets. The formal definition of EST data structure is as follows (although it is usually explicitly distinguished throughout in this thesis, the term "extended sequence tree" can sometimes be used to mean either the whole method, or the data structure only, depending on the context):

Algorithm 2 REINFORCEMENT-LEARNING-WITH-EST

initialize T to an empty EST
initialize policy π ▷ set initial policy arbitrarily
repeat
  observe state s and append it to empty history e
  repeat
    $a \leftarrow$ SELECT-ACTION(s, T) ▷ modified to handle both EST and underlying reinforcement learning algorithm
    perform a, observe s' and r ▷ r is the immediate reward
    $\pi \leftarrow$ REINFORCEMENT-LEARNING-UPDATE(s, s', a, r)
    append a, r and s to e
    $s \leftarrow s'$
  until s is terminal
  $T \leftarrow$ UPDATE-SEQUENCE-TREE(T, e)
until some convergence criterion is met

Definition 2.7. *An extended sequence tree is a tuple $\langle N, E \rangle$, where N is the set of nodes and E is the set of edges. Each node represents a unique action sequence that is used to reach that node; the root node, denoted by $\emptyset$, represents the empty action set. If the action sequence of node q can be obtained by appending action a to the action sequence represented by node p, then p is connected to q by an edge with label $\langle a, \psi \rangle$; it is denoted by the tuple $\langle p, q, \langle a, \psi \rangle \rangle$. ψ is the eligibility value of the edge to indicate how frequently the action sequence of q is executed. Furthermore, q holds a list of tuples $\langle s_1, \xi_{s_1}, R_{s_1} \rangle, \ldots, \langle s_k, \xi_{s_k}, R_{s_k} \rangle$ stating that action a can be chosen at node p if current state observed by the agent is in $\{s_1, \ldots, s_k\}$, which is called the continuation set of node q, denoted $cont_q$. R_{s_i} is the expected total cumulative reward that the agent can collect by selecting action a at state s_i after having executed the sequence of actions represented by node p. ξ_{s_i} is the eligibility value of state s_i at node q and indicates how frequently action a is actually selected at state s_i.* ■

The EST data structure is designed as a repository of valuable histories. Every stored history represents a state-action sequence starting with a distinguished initiation state. At this point, it is useful to redefine *history* (Definition 2.6) to be initiated by a state and to be generated by a policy π, as follows:

Definition 2.8. *A history is called a π-history of state s if it starts with state s and it is obtained by following a policy π until the end of an episode (or between designated conditions, such as reaching a reward peak).* ■

Briefly, a procedure tries to update an -initially empty- EST, by adding successful π-histories, at the end of each episode. By this way, an alternative tree of solution routes that are previously proven to be successful becomes available during execution of underlying reinforcement learning algorithm, ready to be invoked whenever it is beneficial to do so.

Figure 2.4 illustrates a sample EST data structure. The problem domain is the fully observable version of the tiny navigation environment defined in [33], and summarized in Section 3.3. The tree is constructed automatically by execution of EST method together with Q-Learning. Starting from the node immediately below the root node, every node and the edge that connects it to the parent node defines an exploitation step in the option. The current node and its parent edge is interpreted as "if one of the states in the current node is observed by the agent, the agent shall invoke the action given in the parent edge." After an exploitation step, the control passes to the node level defined by the children of the current node. The idea is to run an exploitation sequence using the tree from the root node to a leaf node. Note that each path in the tree is a successful sub-policy of the solution space, embodying a potential option.

- The second component is the action selection procedure (called by Algorithm 2 at line 6) which is a modified action selection mechanism in such a way that it can switch the control flow between the action selection of the underlying reinforcement learning algorithm and that of the EST method, by comparing the expected value of the current situation calculated by the policy function of the underlying reinforcement learning algorithm, and the expected value of the experiences accumulated in the EST data structure.

  The modified action selection mechanism is designed to make the proper decision, whether to trigger an option execution or not, at each discrete time step. If an option is not initiated, it executes the regular reinforcement learning action invocation procedure. If an option is initiated, on the other hand, it passes the control flow to follow the EST data structure beginning from the root node. At each time step, the control flow is passed to the child node that contains the continuation set element which best represents the currently observed situation, executing the action labelling the parent edge.

  This routine is repeated until either a leaf node is reached, or the current node does not represent the current situation well. In either case, the option execution is aborted. Meanwhile, after every action execution through the EST trace, a regular single step update is executed by the underlying reinforcement learning algorithm, in order to update the reinforcement learning policy throughout the experienced option exploitation.

- The third component is the tree update mechanism (Algorithm 3) regularly called upon achievement of a goal state (or a reward peak, alternatively) (Algorithm 2, line 12). First, all possible promising π-histories are extracted from a full length successful history by means of state equivalences (Algorithm 3, line 1). Then, all derived sub-histories are fused into the EST data structure representing the sequences in a compact manner, where each path from the root to a leaf represents a successful sub-policy (Algorithm 3, lines 2-4). Update mechanism is also responsible for pruning of relatively unused paths from the tree (Algorithm 3, line 5), handled by a decay and a threshold mechanism keeping track of eligibility values ψ and ξ in EST data structure.

The resulting annotated reinforcement learning algorithm discovers and utilizes useful tem-

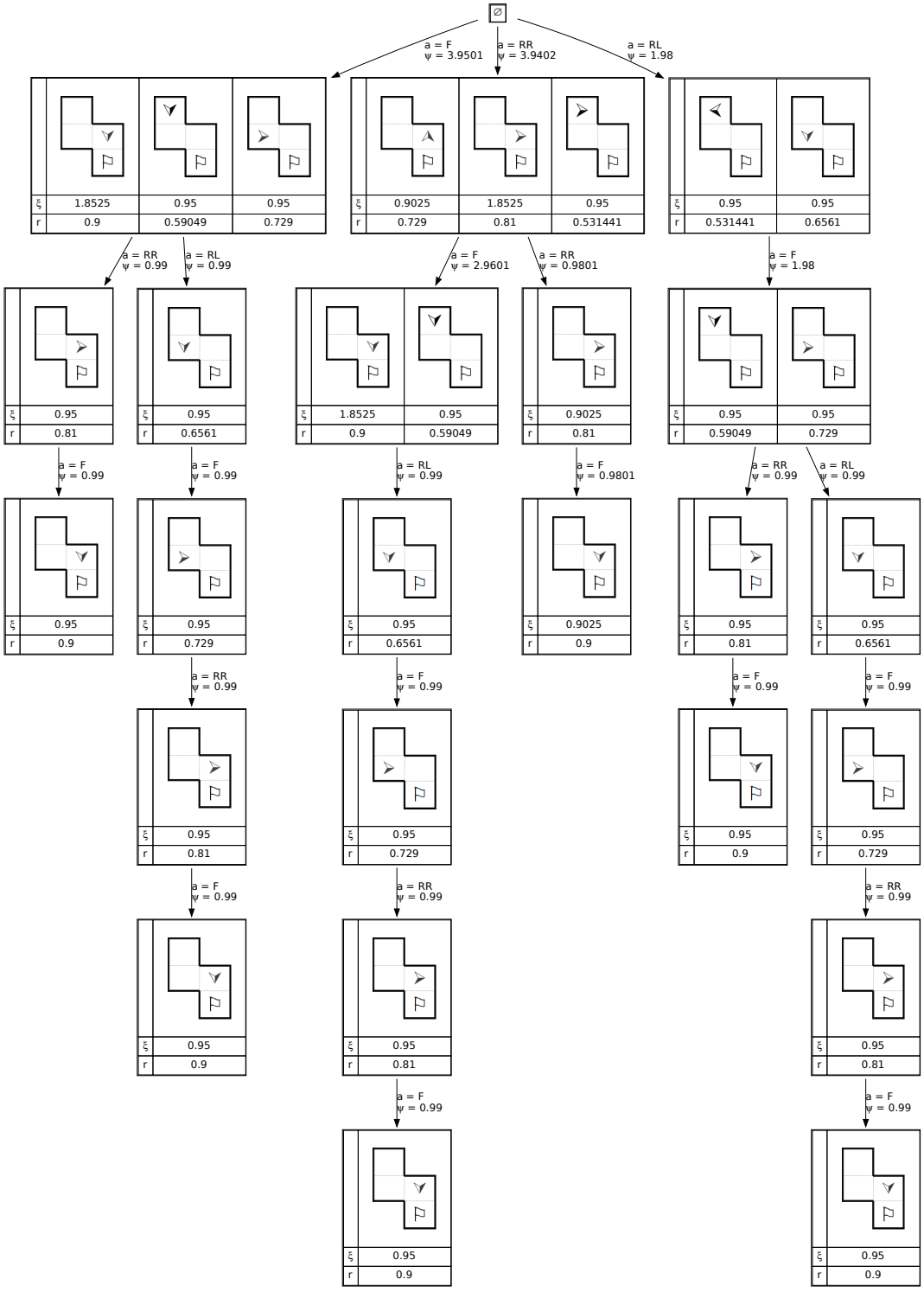


Figure 2.4: An extended sequence tree data structure for the fully observable tiny navigation environment problem. F stands for *forward*, RR for *rotate-right* and RL for *rotate-left* actions.

Algorithm 3 UPDATE-SEQUENCE-TREE(T, e)

Require: T is an EST

Require: e is a history of the form $s_1 a_1 r_2 ... s_{t-1} a_{t-1} r_t s_t$ observed by the agent during a specific period of time.

Ensure: T updated

1: $H \leftarrow$ GENERATE-PROBABLE-HISTORIES(e)
2: **for all** h of H **do**
3: ADD-HISTORY(h, T)
4: **end for**
5: UPDATE-NODE(root node of T) ▷ recursively traverse and update tree for maintenance
6: return T

poral abstractions by generating an EST data structure and using it as a meta-action guide.

Since the underlying reinforcement learning algorithm stays intact, provided that the action selection mechanism allows sufficient exploration (i.e. each state-action pair is visited infinitely often), the extended learning model preserves many of the theoretical properties – such as convergence to an optimal value function or policy– of the underlying reinforcement learning algorithm. The reported test results demonstrate the advantages of EST over other approaches in the literature [21].

It is easy to modify Algorithm 2 for non-episodic problems. For those kind of problems, a history up to a designated "reward-peak" point is collected, and immediately after sensing the reward-peak, the sequence tree is updated, history is cleared, and any other sequence tree related variable is reset.

Recalling the structural pattern of a reinforcement learning algorithm as given in Figure 2.2, the EST mechanism can be viewed as a wrapper around it, and the resulting structural view of Algorithm 2 is given in Figure 2.5. Action selection now passes through a filter of EST, and an episode history is deposited in a history database, to be used to derive EST data structure. The important thing to note here is, the area outside the the dotted rectangle is invoked off-line (i.e. not invoked during reinforcement learning), which gives rise to the opportunity of parallelism or pipelining.

2.6 Reinforcement Learning under Partial Observability

Although reinforcement learning methods depend on the MDP formalism to ensure convergence, there is no technical barrier preventing their invocation to problems with partial observability. An obvious practice would be replacing the states with observations in a reinforcement learning setting. However, most of the time, this approach fails to converge to an optimal policy, mostly because of a problem named as *perceptual aliasing*.

In a fully observable (MDP) domain agent has the full state information at any time. Even if

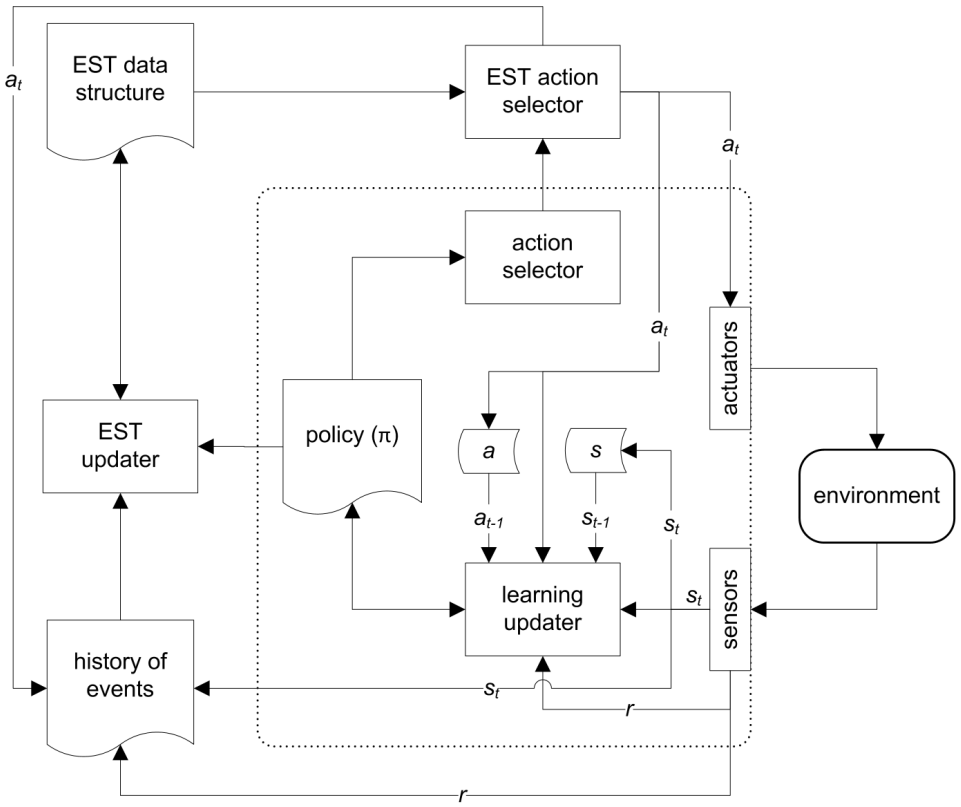


Figure 2.5: A structural view for a reinforcement learning algorithm augmented with EST mechanism.

the actions are non-deterministic, the resulting state is always available to the agent. This setting is based on the assumption that sensors of the agent sense everything in the environment without any restriction. It is almost never the case in real life.

In a more realistic setting, where sensors of an agent are inaccurate and noisy, it is frequently the case that the agent senses two different states to be the same, or similar. More formally, if the same observation is obtained by the agent in two distinct states, where different actions should be performed, the situation is called *perceptual aliasing* [67]. The actual problem is that the resulting observation based process model is not Markovian. When two distinct states require separate actions, the agent has no chance to fulfil this requirement for sure, by using its observation semantics only.

Additionally, belief state formalism (Section 2.3) introduces a continuous problem space issue via belief-MDP construct, which makes the *curse of dimensionality* problem mentioned in Section 2.4 much worse, or even impossible to cope with.

Reinforcement learning studies in POMDP literature are in basically variants of widely known classical algorithms, which focus on diminishing the adverse effects of perceptual aliasing and huge state space.

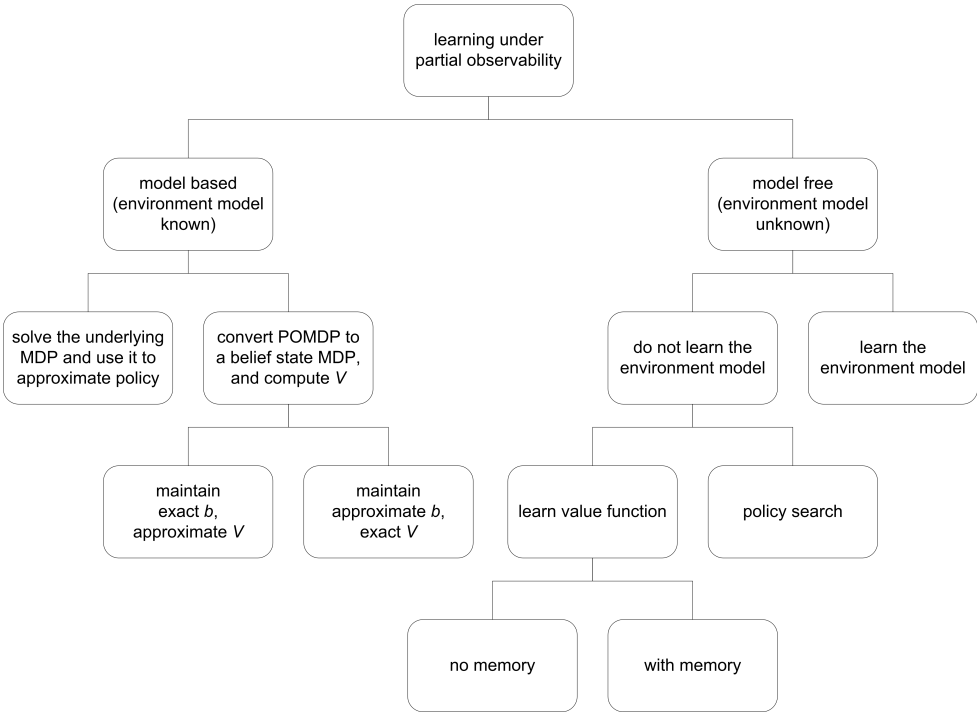


Figure 2.6: Overview of learning approaches for POMDP problems.

An overview of existing learning approaches for POMDP problems is sketched in Figure 2.6. Although many researchers focus on solving POMDP problems via different dynamic programming or planning based methods, this section focuses solely on reinforcement learning point of view and some heuristic methods that might be relevant for our study. Specifically, this thesis highlights

- model based methods (the environment model is known)
- model free methods (the environment model is unknown)
  - memoryless methods: the agent tries to develop a reactive policy
  - memory based methods: the agent makes use of a form of memory

For each category, the most widely used or cited algorithms are emphasized. The reader may refer to [8], [41] or [52] for a broader perspective of POMDP solution techniques, including reinforcement learning.

The following sub-sections summarize the related literature and underlining certain shortcomings and limiting conditions.

2.6.1 Model Based Methods: Known Environment Model

Most of the time, the designer has the opportunity to provide the agent with some additional information about underlying state transitions. Although the agent still can not make a clear discrimination of two perceptually aliased states, a state map, for instance, is more informative than observation space alone. Since finding an exact solution to a POMDP is highly intractable, it is practically useful to solve the known model first, and then merge it with the POMDP solution; or, invoke approximation methods, which can either be over belief states or belief values. Reinforcement learning studies for POMDP which are built on the assumption of a known environment model define an important category of its own, and are summarized below.

2.6.1.1 Methods using the Solution of Underlying MDP

Assume the environment model for a POMDP problem is known by the agent, and assume further that it can be modelled via an MDP. In the literature, the MDP model in this formulation is called *underlying MDP* of the problem. One alternative to solve this POMDP problem is to solve the underlying MDP first, and then combine this solution with the belief state by approximation heuristics.

[8] describes various examples of this approach. One of them is *most likely state* (MLS) approximation. In this intuitive approximation technique, the agent picks the state with the highest probability by using the underlying MDP. Another such heuristic is the Q_{MDP} approximation. Q_{MDP} approximation is based on estimation of the value function of the belief-MDP by using the value function of the underlying MDP weighted according to the distribution encoded in the belief state.

2.6.1.2 Belief Value Approximation Methods

Another approach to approximately solving POMDPs via reinforcement learning is to maintain an exact belief state, but approximate the (PWLC) value function of the belief-state MDP.

As briefly described in Section 2.3, a common way to cope with the limited observation, while maintaining the Markov property, is to relax the problem setting by providing the agent with the underlying MDP model, so that it can maintain an internal *belief state* using T and O. A belief state b is a probability distribution over S. Let $b(s)$ denote the probability assigned to world state s by belief state b ($\forall s \in S, 0 \leq b(s) \leq 1$; $\sum_{s \in S} b(s) = 1$). At each time step, new belief state estimation b' should be computed, given an old belief state b, executed action a, and an observation o. The new belief state in some state s', $b'(s')$, is obtained by using the following equation:

$$b'(s') = \frac{O(s', a, o) \sum_{s \in S} T(s, a, s') b(s)}{Pr(o|a, b)} \tag{2.13}$$

It is possible to convert a POMDP into a *belief-MDP*, so that the problem is transformed into solving a continuous space MDP. Although it is very difficult to solve a continuous space MDPs, there is a practical solution category for belief-MDP based (or model based) reinforcement learning approaches, which attacks the problem by maintaining exact belief state, but approximating the value function on belief-MDP instead of using its exact value.

[42] introduces a reinforcement learning algorithm called SPOVA-RL (Smooth Partially Observable Value Approximation-reinforcement learning) using a new approximation scheme for POMDP problems, based on a continuous, differentiable representation of the value function.

[33] highlights two similar reinforcement learning algorithms of the same category, namely *Replicated Q-Learning* and *Linear Q-Learning*, defined over belief based POMDP model. Both methods generalize Q-Learning to apply to vector valued states and use a single vector, q_a, to approximate the Q function for each action a as $Q_a(b) = q_a \cdot b$, as a linear approximation of the exact Q function. Although a single vector representation per action is not sufficient for most problems, this simple approximation is empirically shown to be effective [12]. The update rule of Replicated Q-Learning is:

$$\Delta q_a(s) = \alpha b(s)(r + \gamma \max_{a'} Q_{a'}(b') - q_a(s)) \tag{2.14}$$

where α is the learning rate, b is a belief state, a is the action executed, r is the immediate reward and b' is the resulting belief state. This update rule is evaluated for every $s \in S$ after each state transition. In Linear Q-Learning, the components of q_a are adjusted to match the coefficients of the linear function that predicts Q values. With a minor change in Equation (2.14), the update rule becomes:

$$\Delta q_a(s) = \alpha b(s)(r + \gamma \max_{a'} Q_{a'}(b') - q_a \cdot b) \tag{2.15}$$

It is worth noting that, when the belief state is deterministic, both Equations (2.14) and (2.15) reduce to ordinary Q-Learning.

Following a notation similar to the structural view in Figure 2.2, Replicated Q-Learning and Linear Q-Learning algorithms can be sketched as in Figure 2.7. The main difference to note here is that, the agent now perceives an observation, instead of the full state, and is also equipped with the state transition function T, by which it is able to update its belief state before feeding it into the learning update engine.

In this thesis, Replicated Q-Learning and Linear Q-Learning are used as the representative algorithms for the model based partially observable reinforcement learning category.

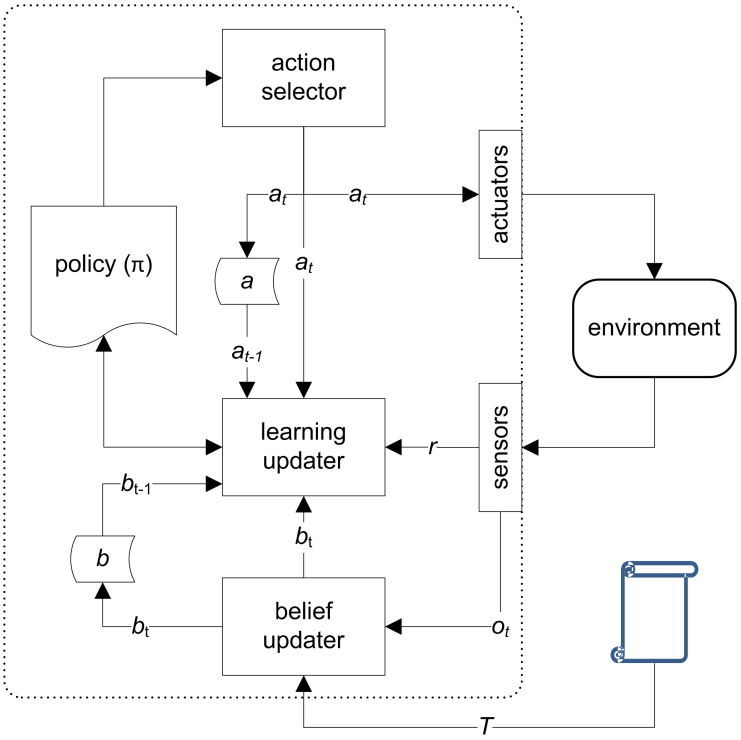


Figure 2.7: A structural view for a belief state based reinforcement learning algorithm for POMDP problems.

2.6.1.3 Belief State Approximation Methods

It is also possible to maintain an exact value function for the belief-state MDP, but approximate the belief state. Boyen-Koller algorithm [6] does this for an environment model specified compactly as a dynamic Bayesian network (DBN), using sampling to perform approximate belief state updating.

2.6.2 Model Free Methods: Unknown Environment Model

This category of POMDP problem setting defines a difficult scenario, where the agent has no prior information about the environment, and tries to explore it by trial-and-error cycles, under limited observability.

2.6.2.1 Memoryless Methods

An obvious way of applying reinforcement learning in POMDP problems is to use "observations" which are available to learner, instead of "states" which are hidden from learner. For instance, the learner can use $Q(o, a)$ values instead of $Q(s, a)$ values. Depending on the strength of the adverse effect of perceptual aliasing (although not descriptive enough,

this strength can intuitively be quantified by $|S|/|\Omega|$ ratio within a problem), obviously this method is destined to succeed arbitrarily.

[32] shows that in non-Markovian environments, finding good deterministic memoryless policies (even for a very weak definition of good) is intractable, even when a complete map of the environment is available to the agent for preprocessing.

Nevertheless, there are a few enhancement efforts on memoryless methods. In their work, [34] make use of a certain family of reinforcement learning algorithms with *eligibility traces* (namely SARSA(λ)) to empirically show that they can work very well on hidden state problems that have memoryless policy solutions. Similarly, [2] propose a memoryless reinforcement learning algorithm named VAPS (value and policy search), making search both in value function space and policy space, and converges for environments with partial observability where a memoryless policy exists.

In this thesis, SARSA(λ) is preferred as the representative memoryless algorithm in the model free reinforcement learning methods for POMDP problems, which is a well known and easy to use alternative.

SARSA(λ) Algorithm An important variant of Q-Learning (Algorithm 1) is SARSA [62], where the update rule of Q-Learning (Equation 2.11) is rewritten as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.16}$$

While Q-Learning algorithm updates Q-values based on the best estimate, SARSA executes these updates in an *on-policy* manner, meaning that, it always updates the Q-value corresponding to the executed action, regardless of whether it is the best possible action or not.

A variant of the classical SARSA algorithm widely accepted for its relatively better performance in generation of memoryless policies under partial observability is SARSA(λ) (Algorithm 4). Due to a mechanism called *eligibility traces*, it is able to derive good policies directly mapping observations to actions, if there exists any [34, 50]. Eligibility trace mechanism invokes a naive form of history tracing into the learning procedure, by keeping track of a trace for each observation-action pair, determining how eligible that pair is. Eligibility trace update procedure is also embedded into the learning algorithm, with a decay parameter $0 \leq \lambda \leq 1$.

Although SARSA(λ) is originally designed assuming the MDP formalism, it is empirically shown that invocation of eligibility traces as a weak form of history performs well for POMDP problems that have memoryless policy solutions [34].

Algorithm 4 SARSA(λ)

repeat
  initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$, $\forall s, a$
  initialize s,a
  repeat
    take action a, observe r, s'
    choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    for all s, a **do**
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
      $e(s, a) \leftarrow \gamma\lambda e(s, a)$
    end for
    $s \leftarrow s'$
    $a \leftarrow a'$
  until s is terminal
until some convergence criterion is met

2.6.2.2 Methods with Internal Memory

If a reinforcement learning agent is augmented with a form of internal memory, it becomes possible to approximate the *complete information state*, based on the current observation and history. This approximation can then be appropriately replaced with the "state" notion of the reinforcement learning method. The learning agent continuously updates the approximation function and value function, resulting in faster convergence and better policies compared to the memoryless policies. Clearly, this enhancement is due to significant disambiguation of perceptually aliased states. Researchers implemented many variants of this idea.

Using a *finite size history* is perhaps the most straightforward way to enhance the internal information state. The main motivation is to use a history of last n steps as the information state. [31] presents the results of experiments that use a finite size history window of past observations and actions. In this method, the designer of the agent shall properly analyse and set the size of the memory needed to model the world, prior to learning.

Memory bits method, introduced by [46], is another memory based method which is inspired by the notion of *stigmergy* (stigmergy is the process by which the results of an insect's activity act as a stimulus to further activity) by means of external "memory bits". Obviously, the impact of this approach to approximate the complete information state is relatively limited, and requires special treatment in action space to manipulate the external memory.

Long short-term memory is a gradient-based approach introduced by [26] as an extension of Recurrent-Q algorithm [31]. Long short-term memory makes use of a special recurrent neural network (RNN) specifically designed to capture relevant features of the problem space.

Recurrent-Q algorithm also uses a RNN to learn to discriminate perceptually aliased states. Recurrent-Q algorithm includes complex neurons that learn to turn on or off their memory inputs and outputs.

The *variable length history* methods attack the problem of maintaining an internal memory with a flexible size that can be adaptively extended whenever required. [38] presents a series of *instance-based* methods that keep all past interactions of the agent with the environment in the form of tuples of action-reward-observation known as "instances". Nearest Sequence Memory (NSM), Utile Suffix Memory (USM), and U-Tree algorithms are variations of the same instance-based reinforcement learning idea, with increasing effectiveness.

Utile Suffix Memory Algorithm Utile Suffix Memory (USM) algorithm [38] is one of the fundamental memory based reinforcement learning algorithms for model free learning in partially observable domains. Using a history database of observations, actions and rewards, the agent eventually learns to discriminate underlying (hidden) states based on statistical differences among the same observations with different history, effectively overcoming the *perceptual aliasing* problem by time.

At the core of the USM algorithm lies a *suffix tree*, used as a repository of short term histories derived from raw experiences, called *instances*. There is no limit on the depth of the tree. The depth is dynamically increased throughout the learning process as necessary to resolve perceptually aliased states via observation-action histories. In-depth expansion of the tree is realized via keeping track of *fringe nodes* up to a certain predefined depth, and *promoting* a fringe node and its necessary relatives whenever a new distinction if found to be valuable. An example USM-style suffix tree structure is given in Figure 2.8. The tree represents a repository of short term instances (back to time $t-2$, inclusive) found to be useful to distinguish the current observation (at time t).

In fact, USM suffix tree is a clustered version of the raw experiences (i.e. action-observation-reward tuples) of the agent, with a clustering schema where the deeper layers of the tree add distinctions based alternately on previous observations and actions. There are three types of nodes in terms of distinctive meaning:

- *Internal nodes* are old leaf nodes and currently have no significance other than identification of a path from root to a leaf.
- *Leaf nodes* constitute the current Q table, each holding a Q value for a pair of distinctive state and action.
- *Fringe nodes* are potential future leaf nodes and are continually applied statistical tests against current leaves for identifying new distinctions.

USM is perhaps one of the most effective memory based reinforcement learning algorithms for problems with hidden state, and is used to represent memory based family of reinforce-

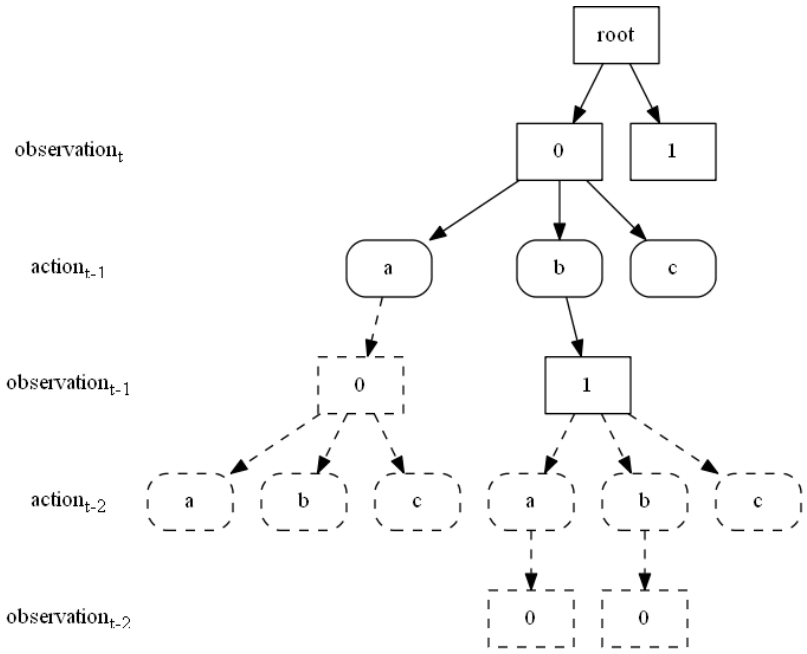


Figure 2.8: A sample USM-style suffix tree data structure. Observations are indicated by integers, actions by letters. The dashed nodes are fringe nodes.

ment learning methods in this thesis. [38] also presents a more popular algorithm, U-Tree, which is essentially a generalized version of USM in which observation and action spaces are *factored* into their dimensions. In other words, USM is the single dimensioned version of U-Tree algorithm. Since EST is not capable of distinguishing multiple dimensions of problem space, U-Tree algorithm is not appropriate for our purposes.

2.7 Temporal Abstractions for Reinforcement Learning under Partial Observability

For POMDP case, there is a limited number of studies on temporal abstractions for reinforcement learning. Most of the relevant studies are from the planning literature [11, 24, 48, 65], which are out of reinforcement learning context.

In the following two sub-sections, existing few studies on temporal abstractions for reinforcement learning in partially observable case are summarized, in both model based and model free perspective.

2.7.1 Model Based Methods

A promising study combining existing framework of MDP and belief state approximation is by [64], which explores invocation of temporally extended actions in POMDPs. A model

based reinforcement learning algorithm is proposed over grid-points in belief space, which uses macro-actions and Monte Carlo updates of the Q-values. The algorithm is then applied to large scale robot navigation and demonstrated the various advantages of macro-actions in POMDPs. Experimental results show that with macro-actions an agent experiences a significantly smaller part of the belief space than with simple primitive actions. In addition, learning is faster because an agent can look further into the future and propagate values of belief points faster. And finally, well designed macros, such as macros that can easily take an agent from a high entropy belief state to a low entropy belief state, enable agents to perform information gathering. The shortcoming of the method is that, incorporation of macro-actions requires a clever design, and they are not derived automatically throughout reinforcement learning process.

[18] presents a way to automatically create and reuse useful sub-goals for partially observable problems in reinforcement learning context. In their work, a state is considered as a sub-goal if it is visited frequently on successful trajectories. Once sub-goals are created, RNNs are used to attain them. Then learned RNNs are integrated into the main RNN as experts. Although the method clearly falls into the family of sub-goal based automated temporal abstractions, the abstraction procedure is defined up to the sub-goal identification phase, lacking the mechanisms to explicitly construct and make use of options.

2.7.2 Model Free Methods

For the memoryless case, a notable algorithm is *HQ-Learning* [68]. HQ-Learning is a sub-goal oriented hierarchical extension of Q-learning in which POMDPs are decomposed into a fixed number of reactive (memoryless) policies, assuming each sub-goal completion is completely observable. Good results are shown on partially observable maze environments with a relatively large number of states. One of the main problems with the method is managing the transfer of control between sub-policies in the presence of noise. Another important problem is that, the proposed architecture is essentially a multi-agent design. Although one can argue that this design can be transformed into a single agent setting, the method relies on the number of agents used to partition the problem, and this number should be provided in advance. Thus, it requires an educated guess on the number of agents in order to to avoid shortcomings in learning, depending on the nature of the problem.

[70] presents another study on memoryless case that attacks the abstraction problem for reinforcement learning with hidden states, by means of derivation of abstractions automatically by their algorithm called Macro/SARSA(λ). Macro/SARSA(λ) is a hybrid algorithm bringing classical SARSA(λ) and a simple routine that acquires macro actions from short term experiences. However, their method suffers from irrelevant macro action generation since it does not have mechanisms to explicitly handle perceptual aliasing.

To our knowledge, there are no temporal abstraction studies that attempt to accelerate a memory based case of reinforcement learning setting for partially observable problems.

CHAPTER 3

ACCELERATING MODEL BASED PARTIALLY OBSERVABLE REINFORCEMENT LEARNING

And there is evidence that believers are happier. So why not believe? It is even possible to believe while knowing that belief is absurd.

– Michael Foley, *The Age of Absurdity*

This chapter proposes a belief based EST method as a direct automatic abstraction mechanism to enhance an underlying model based reinforcement learning algorithm.

Our choice of underlying model based reinforcement learning algorithms imposes a learning setting with exact maintenance of belief states, and approximated use of value function, as summarized in Section 2.6.1.2. In this setting, a trivial modification in EST would be to replace every occurrence of state s with belief b in the EST method, and update all related supporting procedures accordingly. However, since the belief state space is infinite, this solution is practically impossible to work.

Thanks to discretization methods, it is possible to devise a finite approximation of the infinite belief state space. By using belief state discretizations instead of actual belief states, it becomes practically possible to invoke EST method over a model based reinforcement learning algorithm.

3.1 Belief Discretization Methods

Before presenting the belief based EST method, different belief state discretization methods that can be used by our new EST approach are presented in the following three sub-sections. In section 3.2, belief based EST method will be described followed by related experimentations.

Algorithm 5 $D_{GRID}(b, M)$

Require: belief state b over S

Require: resolution of grid $M \in \mathbf{N}^+$

Ensure: a grid coordinate representing discretization of b

1: Create an $|S|$-vector x such that $x(s) = M \sum_{i=s}^{|S|} b(i)$ for $1 \leq s \leq |S|$

2: Let v be the largest integer $|S|$-vector such that $v(s) < x(s)$ for all $s \in S$.

3: Let d be an $|S|$-vector such that $d(s) = x(s) - v(s)$ for all $s \in S$.

4: Let p be an $|S|$-vector that contains a permutation of the integers $1, 2, ..., |S|$ that orders the components of d in descending fashion, so that $d(p(1)) \geq d(p(2)) \geq ... \geq d(p(|S|))$.

5: Find the vertices $\{v^i : 1 \leq s \leq |S|\}$ of the sub-simplex in G' that contains x as follows:

$$
\begin{aligned}
v^1(s) &= v(s), \text{ for } 1 \leq s \leq |S| \\
v^{i+1}(s) &= \begin{cases} v^i(s) + 1, & \text{if } s = p(i) \\ v^i(s), & \text{otherwise} \end{cases}
\end{aligned}
$$

6: return v.

3.1.1 Fixed-Resolution Regular Grid Discretization

Grid based approximation is one of the effective approaches to overcome the dimensionality problem of POMDPs. This family of methods defines a transformation from belief space into a fully observable MDP with a state space that consists of all probability distributions over the core states of the POMDP. For a POMDP with n core states, the transformed state space is called the n-dimensional simplex, or *belief simplex* [71]. *Fixed resolution regular grid* [35] is one of the most popular methods, in which the points of the grid are spaced in a regular pattern and divide the belief simplex into equal-sized sub-simplices.

$D_{GRID}(b, M)$ function in Algorithm 5 defines a slightly modified version of the original method, where final two steps of Lovejoy's original algorithm are truncated. Algorithm 5 finds and returns an integer vector of size $|S|$ representing the grid coordinate of the corresponding discretization of b within the regular grid generated by resolution M. The truncated part, calculating the barycentric coordinates to be used by any interpolation schema that may be required, is not necessary for our purpose.

The major drawback of the algorithm is the exponential growth of the size of the grid as we scale M.

3.1.2 State Rank Discretization

Most Likely State (MLS) approximation [9] is probably the most intuitive and straightforward belief discretization scheme in the literature. [13] proposes a discretization method which generalizes MLS approach using a rank based mechanism, inspired by statistical ranking.

Algorithm 6 $D_{RANK}(b, p_{threshold})$

Require: belief state b over S

Require: a threshold value for cumulative probability $p_{threshold} \in [0.0, 1.0]$

Ensure: a vector of state identifiers representing discretization of b

1: let b_{hash} be a hash table such that $b_{hash}[s_i] = b(s_i)$ for $s_i \in S, 1 < i < |S|$
2: let v be a vector of state identifiers $s_i \in S$
3: $v \leftarrow \langle\rangle$
4: $p_{sum} \leftarrow 0.0$
5: **repeat**
6: $s_{max} \leftarrow \arg\max(b_{hash})$ ▷ resolve ties by a deterministic method, such as lexical comparison of state identifiers
7: append s_{max} to v
8: $p_{sum} \leftarrow p_{sum} + b_{hash}[s_{max}]$
9: delete entry indexed by s_{max} from b_{hash}
10: **until** $p_{sum} \geq p_{threshold}$
11: return v

By $D_{RANK}(b, p_{threshold})$ function given in Algorithm 6, we propose a further generalized version of our original algorithm, which constructs an ordinal state rank vector that is truncated at a given cumulative probability threshold, and can be used as a discrete approximation of a belief state.

For example, consider a belief state b that is represented by the list $0.3, 0.15, 0.0, 0.15, 0.4$, defining probabilities for being in states s_1, s_2, s_3, s_4 and s_5, respectively. Assuming the conflict between ranks of s_2 and s_4 are resolved by comparison of their subscripted indices (i.e. $2 < 4$), D_{RANK} produces the approximate state vectors $\langle s_5\rangle$, $\langle s_5, s_1\rangle$ and $\langle s_5, s_1, s_2, s_4\rangle$ for $p_{threshold}$ values $0.0, 0.5$ and 1.0 respectively.

Note that, for $p_{threshold} = 0.0$, D_{RANK} algorithm is equivalent to the MLS approximation. Given a belief state b, it gives the world state s with the highest probability:

$$MLS(b) = \arg\max_{s \in S} b(s) \tag{3.1}$$

$$= D_{RANK}(b, 0.0) \tag{3.2}$$

Another feature of the D_{RANK} algorithm is that it crops all impossible states (i.e. $\{s \mid s \in S \wedge b(s) = 0.0\}$) implicitly. Although an impossible state has meaning in the belief state formation, it is obviously redundant for a rank based discretization method.

Algorithm 7 $D_{AUG}(b, p_{threshold}, n)$

Require: belief state b over S

Require: a threshold value for cumulative probability $p_{threshold} \in [0.0, 1.0]$

Require: number of partitions $n \in \mathbf{N}^+$ for normalized entropy discretization

Ensure: a tuple of state identifier vector and a positive integer representing discretization of b

1: let v be a vector of state identifiers $s_i \in S$

2: $v \leftarrow D_{RANK}(b, p_{threshold})$ ▷ by Algorithm 6

3: $H_d(b) \leftarrow \lfloor n\overline{H}(b) \rfloor$ ▷ by equations 3.3 and 3.4

4: return $\langle v, H_d(b) \rangle$

3.1.3 Augmented State Rank Discretization

For discretization of belief state, [49] makes use of Augmented MDP (AMDP) representation. Augmented MDP approach is based on the fact that, for most POMDP problems, uncertainty of the system is domain specific and localized. Under this assumption, it is usually possible to summarize a belief vector by a tuple of MLS and the entropy of the belief state. Whenever MLS of a belief state suffers from perceptual aliasing, the level of uncertainty represented by the entropy distinguishes it from other similar belief states.

Entropy of a belief state b is given by

$$H(b) \quad = \quad -\sum_{i=1}^{|S|} b(s_i) log_2 b(s_i) \tag{3.3}$$

$$\overline{H}(b) \quad = \quad \frac{H(b)}{H(b_u)} \tag{3.4}$$

where b_u is the uniform distribution over all states, and $\overline{H}(b)$ is the entropy normalized to lie in the interval $[0, 1]$. Combining MLS and normalized entropy, the low dimensional representation $\tilde{b}$ of belief b is the tuple

$$\tilde{b} \quad = \quad \left\langle MLS(b); \overline{H}(b) \right\rangle \tag{3.5}$$

Obviously, the entropy value makes $\tilde{b}$ a continuous variable. In order to make AMDP space finite sized, [49] discretizes the entropy value into a fixed number of cells.

The fact that $\tilde{b} = \left\langle D_{RANK}(b, 0.0); \overline{H}(b) \right\rangle$ suggests an extension to the original AMDP to cover state rank discretization mechanism defined in Section 3.1.2. More generalized version of AMDP approximation method is proposed in Algorithm 7, which unites D_{RANK} with entropy discretization. AMDP is just a special case of D_{AUG}, where $p_{threshold} = 0$.

D_{AUG} method provides a means to increase the granularity of D_{RANK} for problem domains that might make use of distinguishing effect of entropy augmentation.

3.2 Belief Based Extended Sequence Tree Algorithm

Given a number of belief discretization methods, it is now possible to redesign EST method to cover model based reinforcement learning algorithms Linear Q-Learning and Replicated Q-Learning for partial observability.

Let D be a discretization function over belief state space. We redefine building blocks of EST method using belief state discretization, starting with the redefinition of history (Definition 2.6).

Definition 3.1. *A D-history is defined as*

$$h_{D_{t\tau}} = D(b_t), a_t, r_{t+1}, D(b_{t+1}), a_{t+1}, ..., r_\tau, D(b_\tau)$$

where actions and rewards observed by the agent are listed starting from time t until time τ. ■

Next, the original extended sequence tree (Definition 2.7) is modified to handle discretized belief states.

Definition 3.2. *A D-extended sequence tree (D-EST) is a tuple $\langle N, E \rangle$, where N is the set of nodes and E is the set of edges. Each node represents a unique action sequence that is used to reach that node; the root node, denoted by $\emptyset$, represents the empty action set. If the action sequence of node q can be obtained by appending action a to the action sequence represented by node p, then p is connected to q by an edge with label $\langle a, \psi \rangle$; it is denoted by the tuple $\langle p, q, \langle a, \psi \rangle \rangle$. ψ is the eligibility value of the edge to indicate how frequently the action sequence of q is executed. Furthermore, let d_i denote $D(b_i)$ (D being the discretization function over belief state space); q holds a list of tuples $\langle d_1, \xi_{d_1}, R_{d_1} \rangle$, ... , $\langle d_k, \xi_{d_k}, R_{d_k} \rangle$ stating that action a can be chosen at node p if current approximation of belief state observed by the agent is in $\{d_1, ..., d_k\}$, which is called the continuation set of node q, denoted $cont_q$. R_{d_i} is the expected total cumulative reward that the agent can collect by selecting action a upon gathering a belief state with approximation d_i after having executed the sequence of actions represented by node p. ξ_{d_i} is the eligibility value of approximation d_i at node q and indicates how frequently action a is actually selected at some state yielding a belief state with approximation d_i.* ■

Finally, we enhance the model based reinforcement learning algorithm with the new EST mechanism, which we call *Belief based EST* or *BEST*, as in Algorithm 8. This modifies the original EST method such that, the underlying reinforcement learning method is now one of the model based algorithms (Linear Q-Learning or Replicated Q-Learning), and EST actors are replaced with the new ones as defined in Definitions 3.1 and 3.2.

Algorithm 8 BEST(D)

Require: D is a discretization function over belief space

1: let T_D be a D-EST
2: initialize $T_D \leftarrow$ tree with empty root node only
3: **repeat**
4: let $current$ denote the active node of T_D
5: initialize $current \leftarrow$ root node of T_D
6: let b be the current belief state
7: initialize episode D-history $h_D \leftarrow D(b)$
8: $active \leftarrow false$
9: **repeat**
10: **if** $active = true$ **then**
11: $a \leftarrow$ select action using $D(b)$ and T_D, updating $active$ as side effect
12: **else**
13: $a \leftarrow$ select action using current value function, updating $active$ as side effect
14: **end if**
15: take action a, observe r and construct next belief state b'
16: execute underlying reinforcement learning update rule (Equation 2.14 or 2.15)
17: append r, a, $D(b')$ to h_D
18: $b \leftarrow b'$
19: **until** environment signals a terminal state
20: update T_D by episode history h_D
21: **until** a termination condition holds

Function D can be replaced by one of the discretization methods, as long as it provides a finite set of approximate states for D-EST data structure. The invoked discretization method is not intended to replace belief states in the underlying model based learning procedure, but is used to build up a finite set of representative state approximations to be used in BEST.

Action selection mechanism (lines 10-14) shall invoke $Q_a(b) = q_a.b$ approximation in line 13, instead of direct derivation of the state-action value. Additionally, if the action selection strategy directs the flow of control to T_D (line 12), the mechanism shall need to compare existing belief approximations in the continuation sets of nodes.

In a certain perspective, BEST is a generalization attempt to EST in order to cover a more general reinforcement learning family, just like model based reinforcement learning is a generalization of classical reinforcement learning. To be more specific; similar to Q-learning, which is a special case of algorithms presented in sub-section 2.3, EST is a special case of BEST, for which $D = MLS$.

Following the convention of Figure 2.7, Figure 3.1 summarizes how BEST structurally appends the base reinforcement learning algorithms.

In the following section, we provide empirical evidence presenting that, with an appropriate

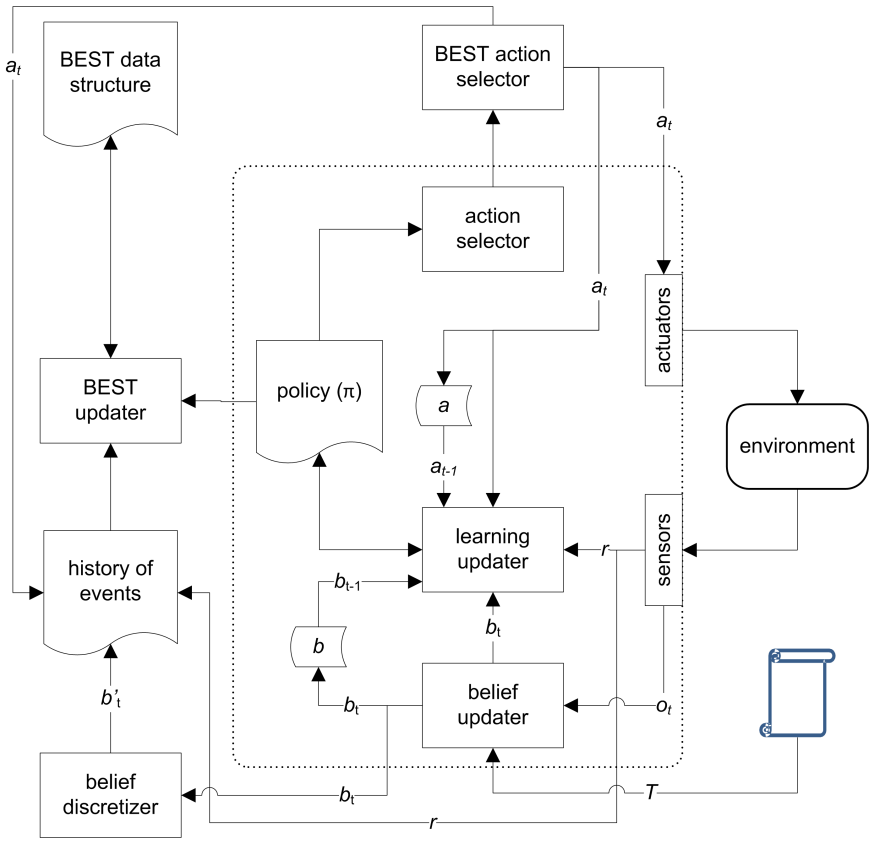


Figure 3.1: A structural view for a belief state based reinforcement learning algorithm for POMDP problems, together with BEST extension.

belief discretization method, BEST effectively improves learning performance of underlying model based reinforcement learning algorithm.

3.3 Experiments

A number of experiments are done to provide empirical evidence that BEST actually enhances performance of selected partially observable reinforcement learning methods. The domains are of different sizes and difficulties, which are selected among widely used benchmark problems in POMDP literature, which are described in the following paragraphs:

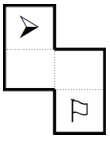


Figure 3.2: Cassandra's tiny navigation problem (*Mini-hall*).

- Tiny navigation environment (*Mini-hall*) [33] is a hallway navigation problem with deterministic actions and limited deterministic sensors (Figure 3.2).

The problem is intended to model a robot navigating in a simple office environment. There are four cells, one of which is a distinguished goal cell, and initially the agent may be in any cell except the goal cell (indicated by a flag), facing one of the four compass directions. The agent can observe relative location of the surrounding walls, and whether it is in goal cell or not. Possible actions are *forward*, *rotate left* and *rotate right*.

The purpose of the learning agent is to devise the policy to reach a designated goal cell of the simulated grid world, in minimum number of actions. This is a simple problem with full determinism, representing the most trivial situation in our problem set.

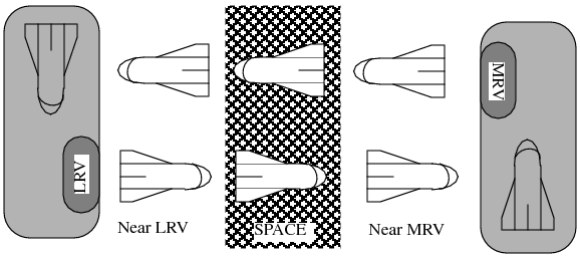


Figure 3.3: Chrisman's space shuttle docking problem (*Shuttle*) [12].

- Space shuttle docking problem (*Shuttle*) [12] is a simplified discrete simulation of a space shuttle, alternating between two space stations to deliver supplies (Figure 3.3). In this problem, agent's focus is on maximization of total reward throughout this delivery process, by trying to learn how to go to the least recently visited station.

  Three possible movements are deterministic *go forward*, *turn around* actions, and the noisy *backup* action. With a certain noise, the shuttle agent can see the state station in front of it, or it can sense that it is docked in one of the stations, or it can see nothing.

  What distinguishes this problem from others is that it lacks a designated goal state, so the task is not inherently episodic. Although the problem size is small, unreliability of sensors and actions places the problem to a more difficult category than *Mini-hall* in our set.

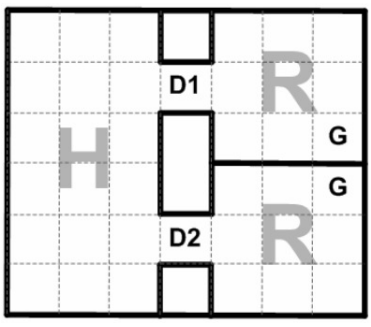


Figure 3.4: Dung's *Virtual Office* problem [18].

- *Virtual Office* [18] problem is a navigation problem that has two bottleneck states (D1 and D2), and two goal states (G) that can be reached through each bottleneck state

separately (Figure 3.4).

The agent is expected to learn to move from any random starting position in the hall H (the left room) to one of the goals in the right rooms. The agent can observe with certainty whether there is wall at each of the four compass directions or not. Due to this observation semantics, the observations in the upper right room are same as observations in the lower right room except the goal positions. Action space is composed of movement actions to the four compass directions. If the agent reaches the goal, it receives a reward of $+10$. For any other movement, the agent gets a reward of -1.0. The door states D1 and D2 clearly impose a hierarchical structure in the solution policy for the domain. Fortunately, there T and O functions are deterministic in this problem.

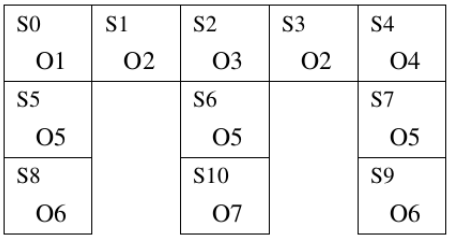


Figure 3.5: State/observation space of Pineau's *Cheese-Taxi* problem [47].

- *Cheese-Taxi* [47] is a hybrid of two well known problems, namely *Cheese Maze* [38] and *Taxi Domain* [16]. *Cheese-Taxi* combines them to join the state uncertainty aspects proper to the *Cheese Maze* and the hierarchical structure proper to the *Taxi Domain*, which are the motives that make it a member of our set (Figure 3.5).

  The problem simulates a taxi agent, where the agent must pickup a passenger located at state S10 and then deliver the passenger to a destination cell, either S0 or S4, selected randomly. The agent can execute seven actions: *North*, *South*, *East*, *West*, *Query*, *Pickup*, *Putdown*, and can perceive ten distinct observations: O1, O2, O3, O4, O5, O6, O7, *destination*S0, *destination*S4, and *null*. Observations O1 through O7 indicate wall placement in all four directions immediately adjacent to the location. Note that, states S5, S6 and S7 look identical, as do respectively S1, S3 and S8, S9. Other states are uniquely identified by corresponding observations. Destination observations are provided without noise to the agent in response to the $Query$ action, only when the passenger is onboard. The agent perceives a $Null$ observation after the $Pickup$ and $Putdown$ actions.

  The goal of the agent is to deliver the passenger at the correct location and receive the $+20$ reward. Incorrect use of $Pickup$ and $Putdown$ are punished with a reward of -10. There are two sources of uncertainty in this problem. The initial location of the taxi is randomly distributed over maze cells and can only be disambiguated by taking a sequence of motion actions.

- *Hallway* is a middle sized domain (Figure 3.6) in a series of hallway navigation problems proposed by [8]. Observation and action semantics is the same as in *Mini-hall*

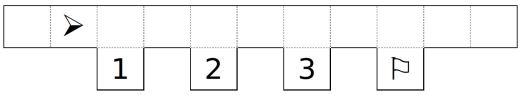


Figure 3.6: Cassandra's *Hallway* navigation problem [8].

domain. Additionally, it has three distinguishing states where the agent can perfectly sense its absolute location, so that agent can gain an advantage if it devises a policy making use of these three states.

The domain is relatively large and extremely noisy, which makes it harder than the previous problems.

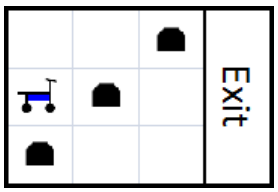


Figure 3.7: Rock sampling problem with a 3x3 grid and 3 rocks [56].

- Rock sampling problem (*RockSample[x,y]*) [56] is a scalable problem that models rover science exploration, where x is the dimension of square-shaped grid world and y is number of rocks in the environment (Figure 3.7).

  In addition to the four compass directions, the rower can execute $Sample$, $Check_1$, $Check_2$, ... $Check_y$ actions. Each rock in the world is randomly set to either "valuable" or "valueless" for sampling. The aim of the rover agent is to $Sample$ all "valuable" rocks, and then go to the designated exit state. It always knows the coordinates of itself and the rocks, but it does not know in advance, which rocks are "valuable" (i.e. worth sampling). By performing the dedicated sensory action $Check_i$ for each rock, it can imperfectly perceive whether the rock is valuable or not, via a noisy sensor depending on distance. When the rover samples a rock, the rock becomes "valueless".

  $RockSample$ is the most challenging problem of our set because of its following properties: (1) The environment is non-stationary, since after sampling a rock it becomes valueless. Moreover, each randomly generated initial state configuration defines a different maximum total reward that can be expected. (2) Observation space is not only very restricted, but also unreliable.

3.3.1 Setup

Table 3.1 summarizes the experimented problem domains, providing the sizes of problems in terms of state, action and observation spaces, noise in the problem's transition function (T) and observation function (O) and the reference publication for the domain.

Each problem domain is experienced with different settings of $p_{threshold}$, M and n as given

Table 3.1: Problem Domains

Problem	Sizes			Noise	Ref.
	$\|\mathbf{S}\|$	$\|\mathbf{A}\|$	$\|\mathbf{\Omega}\|$		
Mini-hall	13	3	9	-	[33]
Shuttle	8	3	5	T/O	[12]
Virtual Office	38	4	12	-	[18]
Cheese-Taxi	35	7	10	T	[47]
Hallway	60	5	21	T/O	[8]
RockSample[3,3]	257	9	2	O	[56]

Table 3.2: Experiment Settings

Problem	D_{GRID}	D_{RANK}	D_{AUG}	
	M	$p_{threshold}$	n	$p_{threshold}$
Mini-hall	7	0.2	10	0.0
Shuttle	3	0.1	5	0.0
Virtual Office	3	0.6	5	0.0
Cheese-Taxi	15	1.0	15	1.0
Hallway	10	1.0	20	1.0
RockSample[3,3]	5	1.0	10	1.0

in Table 3.2, which are identified via a number of trial-and-error experimentations. These parameters were observed to provide performance gain compared to learning without abstraction for the given problem. Additionally, they perform relatively better in terms of performance parameters like CPU time and memory footprint.

Default learning settings for problems are given in Table 3.3. The only exception is the *RockSample* domain, for which ϵ is taken as 0.3, in order to promote exploration and achieve faster convergence, since learning failed to converge in a reasonable time with the default settings.

For all experiments, Modified-ϵ-Greedy method in [21] is updated by the required changes in the action selection mechanism, as discussed in Sections 2.5.3.2 and 3.2.

Each problem is experimented with a bare model based reinforcement learning algorithm (i.e. Linear Q-Learning or Replicated Q-Learning; without BEST) against the same algorithm en-

Table 3.3: Learning Settings

Parameter	Value
α	0.125
γ	0.9
ϵ	0.1
ψ_{decay}	0.95
ξ_{decay}	0.99
$\psi_{threshold}$	0.01
$\xi_{threshold}$	0.01

hanced with BEST (by Algorithm 8) for every discretization function using the settings for the problem as given in Table 3.2. The experiments were run for varying number of episodes for each problem, depending on the convergence behavior. All test runs were executed 250 times and the results were averaged over episodes of each run. Primary performance criterion is the average reward gained by the agent per time-step (*reward-per-step*); except for *Rock-Sample* domain, due to its non-stationary nature, average accumulated reward (*total reward*) is measured instead.

For *Shuttle* problem, due to its non-episodic nature, an episode is assumed to last 250 steps, after which a new episode begins. For the same reason, in this problem, Algorithm 8 is modified with a "reward-peak" strategy, where maximum possible reward (instead of episode end) is used to trigger a sequence-tree update.

3.3.2 Results and Discussion

Before presenting the results of the experimentations, it will be convenient to have a measure of the scales of selected problems in mind. Although defining uncertainty throughout an agent's experience within an environment is not trivial, normalized entropy of belief states (Equation 3.4) may give a rough idea about how blur the agent senses the environment it is trying to explore. Figure 3.8 shows normalized average belief entropy values measured over plain Replicated Q-Learning (without BEST) experiments for each domain. Remember that a lower value means less uncertainty. It can clearly be seen from the plot that *RockSample[3,3]* domain is the most misty environment of the problem set, while *Shuttle* domain provides the most complete information state to the agent on the average.

Figures 3.9 to 3.14 show the average performances of each setting given in Table 3.2 throughout the episodes of the experiment. Plots are smoothed for visual clarity.

BEST has some extra CPU time cost, which is given in Figure 3.15. For every domain and for each discretization method, this figure shows the CPU time overhead of BEST over the

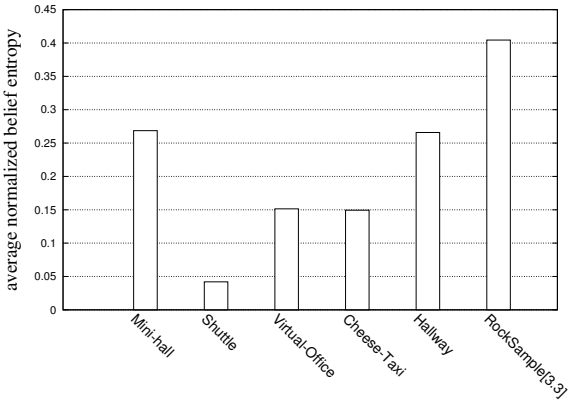


Figure 3.8: Average normalized belief entropy levels of problems.

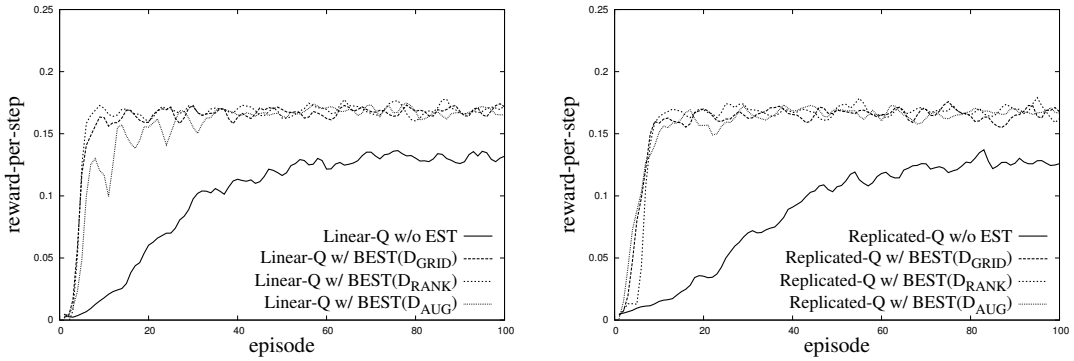


Figure 3.9: Experiment results for *Mini-hall* domain

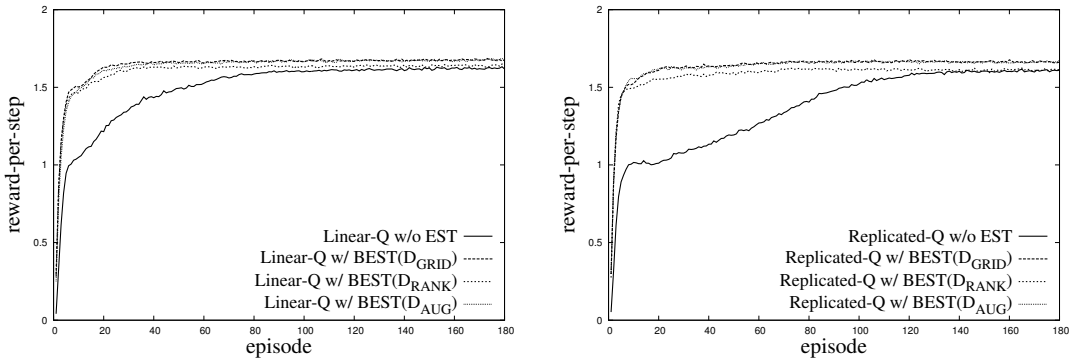


Figure 3.10: Experiment results for *Shuttle* domain

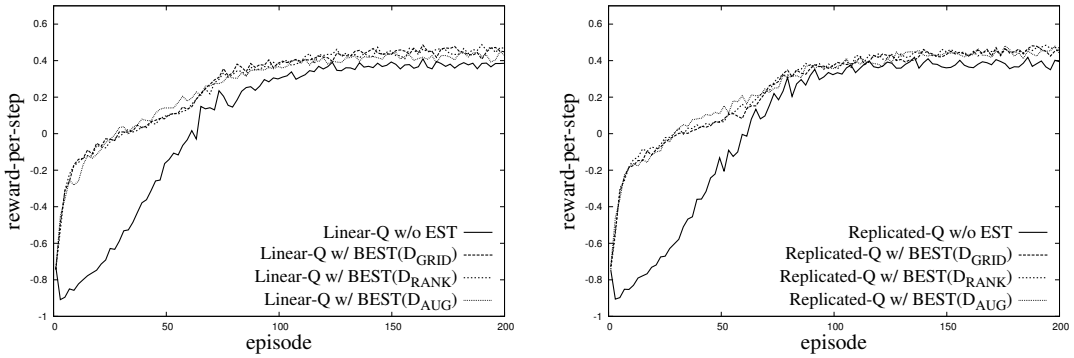


Figure 3.11: Experiment results for *Virtual Office* domain

underlying learning algorithm, as a percentage value.

As another important performance parameter is the memory footprint, Table 3.4 shows the average number of nodes generated by BEST for each domain, given the underlying reinforcement learning algorithm and the discretization method.

Figure 3.16 provides the ratio of abstract actions in all actions taken during learning. Namely, it is the average usage rate of "options" compared to all primitive "actions" taken, as a per-

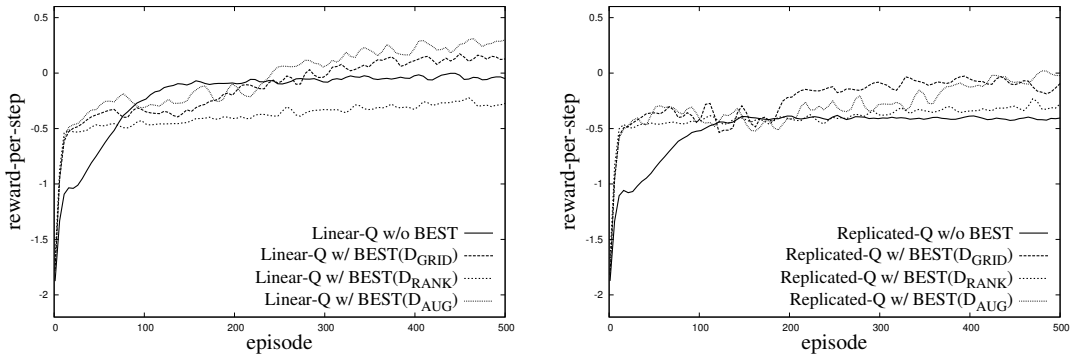


Figure 3.12: Experiment results for *Cheese-Taxi* domain

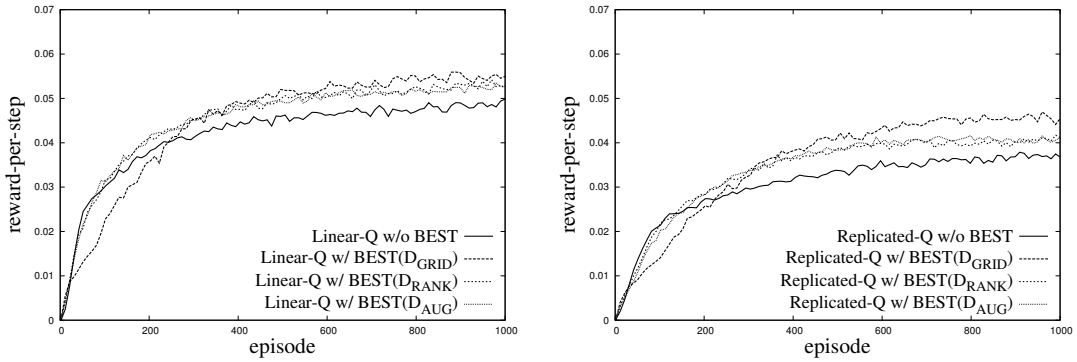


Figure 3.13: Experiment results for *Hallway* domain

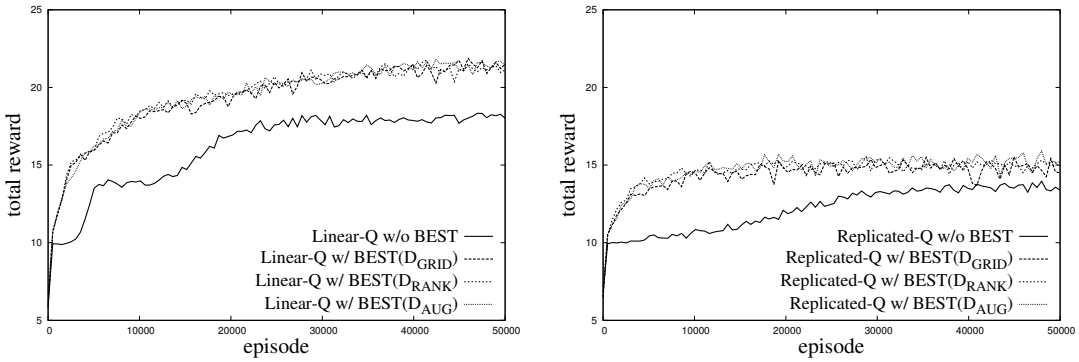


Figure 3.14: Experiment results for *RockSample[3,3]* domain

centage value.

Finally, Table 3.5 is a summary of discretized state space size generated by the BEST for a for each discretization method, given each problem domain.

For almost all cases, significant performance gain is achieved by the use of BEST on top of model based reinforcement learning. However, interpretation of parameters that play role on the "significance" level of performance gain is not straightforward.

Mini-hall, as one of the simplest problems of our set, seemed to take more advantage of BEST than other problems. With appropriate parameter values, all discretization methods make BEST boost learning, beginning from the very early stages of learning (Figure 3.9). Even for small values of M, n and $p_{threshold}$, like the ones in Table 3.2, BEST succeeds to improve underlying learning algorithm, mostly because, on the average, almost 50% of actions are abstracted and used as "options" (Figure 3.16), which is the property that leads BEST to success, even for such a small sized domain.

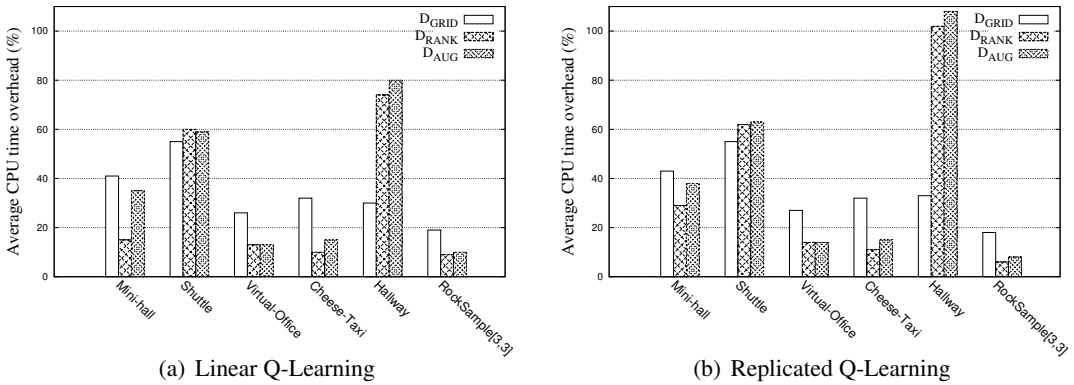
Shuttle, another small domain of the problem set, also made significant performance gain by using BEST. Although abstraction potential is much less than other problems (as indicated by very low option usage percentages in Figure 3.16), its deterministic nature (as pointed by low uncertainty in Figure 3.8) gives rise to performance increase, even for very small values of discretization parameters (Table 3.2).

As seen in Figure 3.11, BEST also performed well on *Virtual Office*, one of the problems specially designed suitable to temporal abstraction. Note the high values of option usage percentages for this problem in Figure 3.16.

The other problem that is designed for temporal abstraction is *Cheese-Taxi* domain that also benefits from BEST (Figure 3.12). Especially at the initial stages of learning, BEST supports underlying learning algorithm very well. For this problem domain, although D_{RANK} boosts performance at the earlier episodes, it fails to successfully represent the noisy nature of the domain later on. D_{AUG} and D_{GRID} (with appropriate parameters) perform much better with their ability to define a more granule discretized state space for BEST (Table 3.5).

For extremely noisy domains like *Hallway*, it is highly probable that BEST discovers some options that are in fact not correct, but accidentally succeeded (due to noise) and recorded as a successful history, thus added to the D-EST data structure. Although these paths are pruned later from the D-EST, the pruning mechanism has some latency in our settings, due to conservative decay and threshold parameter values given in Table 3.3, which explains the late recovery of the D_{GRID} curves. Additionally, as the number of problem states increases and average belief entropy gets high, the number of discretization required increases. As a consequence, in accordance with the relatively large discretized belief state space generated (Table 3.5), high CPU time overhead can be observed, as is the case for the *Hallway* problem (Figure 3.15).

When we compare the discretization methods, the most remarkable gaps in the average number of nodes are observed in the *Hallway* domain (Table 3.4). D_{RANK} and D_{AUG} require significantly more nodes than in the D_{GRID} case. This is because D_{GRID} discretization seems to be more suitable in the long term for generation of more frequently invoked (or more useful) options for the *Hallway* domain (Figure 3.16) by using less number of discretized states (Table 3.5). In general, it is more likely to have such gaps for extremely noisy and relatively large domains like *Hallway*. For this kind of situations, D_{RANK} and D_{AUG} have a tendency to represent the continuous belief state space with more number of discretized states. This



(a) Linear Q-Learning (b) Replicated Q-Learning

Figure 3.15: Average CPU time overhead percentages.

may easily trigger the production of different discretizations for similar belief states, leading to the generation of more options as the solution of the same sub-task.

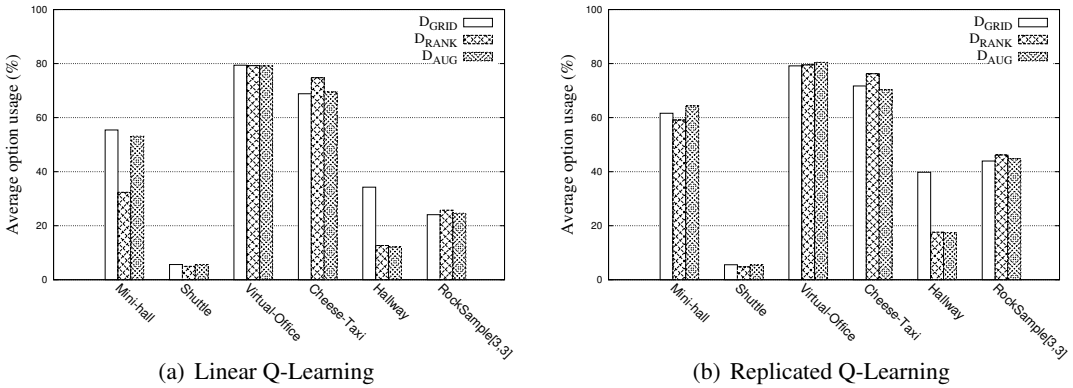
Finally, for the *RockSample[3,3]* problem, BEST with any discretization method clearly outperforms a model based reinforcement learning setting without BEST. Although this domain exhibits the highest uncertainty (Figure 3.8), BEST successfully supports the underlying learning algorithm with a relatively moderate percentage of option usage throughout an episode (Figure 3.16).

Complexity of all three discretization algorithms presented in this paper are $O(|S|log|S|)$. Thus, it is hard to identify a direct effect of a selected discretization method to the overall computational cost. However, together with the nature of the problem domain, the discretization method used seems to determine the size of the discretized belief state space, thus indirectly effecting the number of D-EST nodes and CPU time overhead.

Model based reinforcement learning with BEST can reach an average reward peak at a much earlier episode than learning alone (e.g. for *Shuttle* domain, model based reinforcement learning catches its peak near 100^{th} episode, while the same algorithm with BEST reaches similar values before 20^{th} episode), and succeeds to increase total reward due to option invocations.

Table 3.4: Average number of nodes in D-EST

	Linear Q-Learning			Replicated Q-Learning		
Problem	D_{GRID}	D_{RANK}	D_{AUG}	D_{GRID}	D_{RANK}	D_{AUG}
Mini-hall	103.47	62.17	104.83	105.37	62.69	105.36
Shuttle	51.19	43.43	50.65	55.45	45.83	53.71
Virtual Office	419.80	427.03	381.76	431.14	424.25	374.68
Cheese-Taxi	886.78	525.93	794.50	828.06	505.79	788.09
Hallway	9959.80	48588.29	54999.75	12470.42	79783.73	88499.95
RockSample[3.3]	1518.97	1289.17	1382.53	691.71	555.63	612.44



(a) Linear Q-Learning (b) Replicated Q-Learning

Figure 3.16: Average usage of options among all action steps.

For this reason, CPU time overhead is generally fairly acceptable due to early convergence and apparent increase in average rewards. Moreover, it is always possible to run BEST in parallel with the underlying learning algorithm with a certain number of episodes latency, due to its off-line nature.

Of course, there are issues where BEST needs improvement. Perhaps the most important one is that, the success of BEST requires correct selection or adjustment of abstraction and discretization parameters. Unfortunately, there is no common practice on how to estimate the correct values of these parameters for a specific domain, except some sort of an educated guess.

Table 3.5: Average discretized belief state space size generated

	Linear Q-Learning			Replicated Q-Learning		
Problem	D_{GRID}	D_{RANK}	D_{AUG}	D_{GRID}	D_{RANK}	D_{AUG}
Mini-hall	19.13	13.04	18.18	19.19	12.95	18.01
Shuttle	16.71	7.95	13.11	16.78	7.96	13.01
Virtual Office	49.52	46.88	38.32	49.91	46.99	38.55
Cheese-Taxi	68.08	49.58	68.30	65.40	48.84	67.72
Hallway	2774.11	14049.06	14442.85	3064.65	16999.05	17743.92
RockSample[3,3]	1156.36	889.17	1430.64	1046.57	751.03	1124.92

CHAPTER 4

LEARNING REACTIVE POLICIES FASTER FOR REINFORCEMENT LEARNING WITH HIDDEN STATE

As you adequately put, the problem is choice. But we already know what you're going to do, don't we?

– The Architect (portrayed by Helmut Bakaitis), *The Matrix: Revolutions*

Some of the early studies in reinforcement learning for problems with hidden state focused on generation of memoryless policies that can produce acceptable solutions throughout the observation alone, for a given POMDP problem. In this chapter, we propose a method for enhancing one of the representative reinforcement learning algorithms of this category, namely SARSA(λ), using extended sequence tree abstraction method [14].

As described in Section 2.6, the main problem for reinforcement learning with the hidden state constraint is *perceptual aliasing*, especially when the underlying MDP model is not available to the agent in advance.

Since EST method assumes MDP setting, construction of EST data structure, which is nothing but a tree representing useful loop-free histories in a compact manner, makes the assumption that there is no aliasing on states. Thus, EST method is extremely fragile under partial observability assumption, since a misleading unification of two perceptually aliased states easily directs the agent to a completely wrong direction in solution space.

In order to overcome this problem, a way to identify and get rid of perceptually aliased cases on EST is needed. At this point, it is worth reminding that EST data structure is mainly used for "exploitation" of successful sub-policies under the assumption that EST always keeps only successful histories. Using this fact, it is easy to see that a *misleading exploitation* (i.e. an EST path leading to nowhere valuable) shall help to identify "false positive" paths on the tree.

In this chapter, a careful pruning strategy on the EST data structure is proposed to make this identification. This novel pruning strategy makes it possible to accumulate useful sub-policies from episode histories, with SARSA(λ) as the underlying reinforcement learning algorithm for problems with hidden state.

4.1 History Aware Extended Sequence Tree Data Structure

Before necessary modifications on EST method procedures, the EST data structure is redefined as follows:

Definition 4.1. *A history aware extended sequence tree data structure (HA-EST) is a tuple* $\langle N, E \rangle$*, where* N *is the set of nodes and* E *is the set of edges. Each node represents a unique action sequence that is used to reach that node; the root node, denoted by* $\emptyset$*, represents the empty action set. If the action sequence of node* q *can be obtained by appending action* a *to the action sequence represented by node* p*, then* p *is connected to* q *by an edge with label* $\langle a, \psi \rangle$*; it is denoted by the tuple* $\langle p, q, \langle a, \psi \rangle \rangle$*.* ψ *is the eligibility value of the edge to indicate how frequently the action sequence of* q *is executed. Furthermore,* q *holds a list of tuples* $\langle \langle o_1, \Pi_1^p \rangle, \xi_{\langle o_1, \Pi_1^p \rangle}, R_{\langle o_1, \Pi_1^p \rangle} \rangle$*, ... ,* $\langle \langle o_k, \Pi_k^p \rangle, \xi_{\langle o_k, \Pi_k^p \rangle}, R_{\langle o_k, \Pi_k^p \rangle} \rangle$ *stating that action* a *can be chosen at node* p *if current observation and previous continuation set element makes a pair that is in* $\{\langle o_1, \Pi_1^p \rangle, ..., \langle o_k, \Pi_k^p \rangle\}$ *which is called the continuation set of node* q*, denoted* $cont_q$*.* Π_i^p *denotes the element of* $cont_p$ *that is the immediate ancestor of current continuation set element, meaning that* Π_i^p *was the previous continuation set element chosen in the previous option exploitation step. A continuation set element is indexed by the pair* $\langle o_i, \Pi_i^p \rangle$*.* $R_{\langle o_i, \Pi_i^p \rangle}$ *is the expected total cumulative reward that the agent can collect by selecting action* a *upon gathering a pair* $\langle o_i, \Pi_i^p \rangle$ *after having executed the sequence of actions represented by node* p*.* $\xi_{\langle o_i, \Pi_i^p \rangle}$ *is the eligibility value of pair* $\langle o_i, \Pi_i^p \rangle$ *at node* q *and indicates how frequently action* a *is actually selected at some state yielding the pair* $\langle o_i, \Pi_i^p \rangle$*.* ■

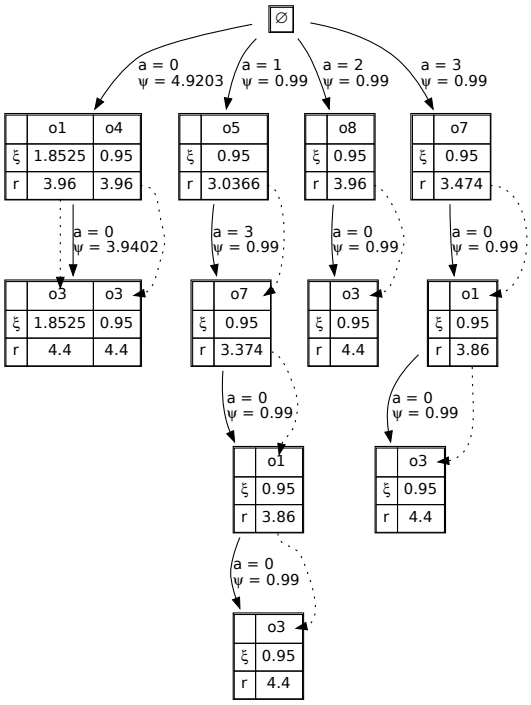


Figure 4.1: An example HA-EST data structure.

Figure 4.1 is an example tree following the Definition 4.1. Continuation set elements in a node are now pairs consisting of an observation and a parent continuation set element (represented by dotted arrows). For this reason, a node of HA-EST may now contain the same observation more than once. Additionally, by this way, every continuation set element in a node is a step on a unique path from root node to a leaf node, unlike the case in the original EST data structure. Putting it the other way, from any continuation set element, there is only one path through continuation set elements up to the root node. This property will be useful for our pruning mechanism.

Construction of HA-EST requires minimal change in the original EST procedures (Algorithms 2 and 3). In the function generating potentially useful sub-histories (called at line 1 of Algorithm 3), the only change is that "states" are replaced with "observations". In the tree update phase (i.e. Algorithm 3), "state" information is replaced with the information pair consisting of "observation" and "previous step mark before that observation", represented by $\langle o_i, \Pi_i^p \rangle$ in Definition 4.1. Mechanisms for history addition and action selection are explained in the following section.

4.2 Extended Sequence Tree Abstraction with Misleading Sub-policy Removal

Every history represented by HA-EST is potentially ambiguous. In other words, any path from the root node to a leaf node through continuation set elements may involve observations that are aliases of some distinct states. These paths should be removed from the tree, so that eventually only unambiguous sub-policies remain. Even when the number of perceptually aliased states are high for a problem, just a few "discriminating" observations (i.e. observations corresponding to states that do not suffer from perceptual aliasing) may lead as the initiation points for unambiguous successful histories.

For this purpose, a pruning mechanism is developed and integrated to the "history addition" (called at line 3 of Algorithm 3) and "action selection" (called at line 6 of Algorithm 2) mechanisms of the EST method.

We first define a *Forbidden Sub-policy Repository* (FSR) that stores histories in the form of observation-action sequences. There are alternative ways to implement such a repository, with varying effectiveness in terms of time and space. Our implementation of FSR is in the form of a tree data structure, which we found to be fairly acceptable. The aim of FSR is keeping track of sub-sequences that have previously been proven to be misleading on the way to goal.

Algorithm 9 is the modified history addition algorithm that prevents insertion of sub-sequences that were added into FSR before, and constructs the continuation set element links as defined in Definition 4.1.

Action selection is the final mechanism to be altered. The original action selection method of EST runs through the nodes beginning from the root until either a leaf node is reached

Algorithm 9 ADD-HISTORY(h, T)

Require: h is a history of the form $o_1 a_1 r_2 ... o_{t-1} a_{t-1} r_t o_t$

Require: T is a HA-EST

$h' \leftarrow o_1 a_1 ... o_{t-1} a_{t-1} o_t$ ▷ rewards removed

if any path in FSR is a prefix of h' **then** ▷ history has ambiguities

    exit ▷ do not add history to HA-EST

end if

$\Pi \leftarrow NULL$ ▷ parent continuation set element link

$R[t] \leftarrow r_t$ ▷ time indexed array of discounted cumulative rewards

for $i \leftarrow t-1$ to 1 **do**

    $R[i] \leftarrow r_i + \gamma R[i+1]$

end for

$n_{current} \leftarrow$ root node of T

for $i \leftarrow 1..t-1$ **do**

    if $\exists$ a node n such that $n_{current}$ is connected to n by an edge with label $\langle a_i, \psi \rangle$ **then**

        Increment ψ.

        if n contains a continuation set element indexed by $\langle o_i, \Pi \rangle$ **then** ▷ update values in continuation set element found

            Increment $\xi_{\langle o_i, \Pi \rangle}$

            $R_{\langle o_i, \Pi \rangle} \leftarrow R_{\langle o_i, \Pi \rangle} + \alpha(R[i] - R_{\langle o_i, \Pi \rangle})$

        else ▷ create new continuation set element in n

            Add a new tuple $\langle \langle o_i, \Pi \rangle, 1, R[i] \rangle$ to node n.

        end if

    else

        Create a new node n containing the tuple $\langle \langle o_i, \Pi \rangle, 1, R[i] \rangle$.

        Connect $n_{current}$ node to n by an edge with label $\langle a_i, 1 \rangle$.

    end if

    $\Pi \leftarrow$ link to continuation set element that contains $\langle o_i, \Pi \rangle$ in n ▷ prepare parent link for next iteration

    $n_{current} \leftarrow n$

end for

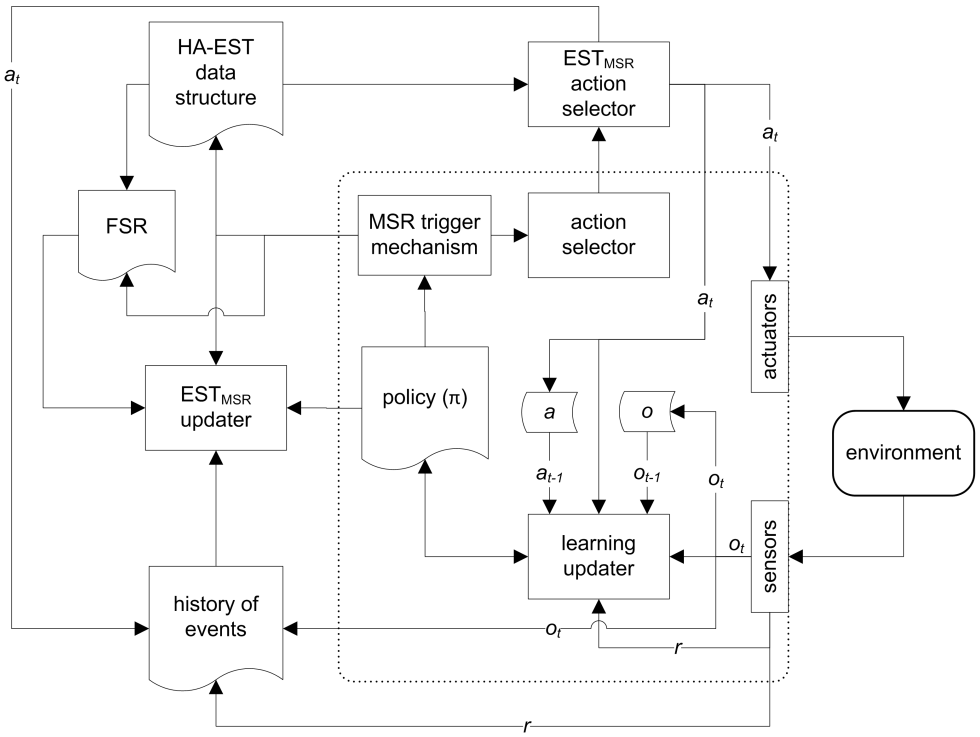


Figure 4.2: A structural view for a reinforcement learning algorithm with hidden states together with EST_{MSR} extension.

or a node does not represent the current situation. In either case, end of the current option is triggered and control is given back to the underlying reinforcement learning algorithm. Under the assumption that domain is stationary and completely deterministic, HA-EST has a useful property to detect ambiguity (Determinism requires that, for any given state, an action results in the same state transition all the time. An environment is stationary if it does not change throughout the learning process). If one of the following events happens, it means the tree path that has been followed was misleading the agent:

- given that the flow of control is on a certain HA-EST node, if the current observation does not exist in any child node (which means the control will exit the option exploitation in the next time step)
- the corresponding action is executed for a continuation set element which has no children, but the goal state has not been reached

Each of the above conditions ensures that the path exploited up to that point contains at least one hidden state suffering from perceptual aliasing. In other words, under the determinism assumption, any option defined by HA-EST should execute until goal state, or must be deleted otherwise.

When the above situation occurs, our modified action selection mechanism recalls the last processed continuation set element, finds all leaf continuation set elements that are reachable

from that element, and finally prunes every continuation set element path in the tree that can be traversed from each leaf continuation set element found, up to the root, by following parent continuation set element links. After deletion, all deleted continuation set element paths, which are nothing but observation-action sequences, are added to FSR, to prevent them to be added into HA-EST again.

We name our HA-EST based method that attacks perceptual aliasing problem using this pruning mechanism as *EST with Misleading Sub-policy Removal* (EST_{MSR}).

Following the convention of Figure 2.2, Figure 4.2 summarizes how EST_{MSR} structurally appends reinforcement learning algorithms which produce reactive policies for problems with hidden state. Note that, all states (s) in Figure 2.2 are replaced with observations (o).

It is also worth noting that if the problem is fully observable and deterministic, MSR mechanism is never triggered and reduces to classical EST.

4.3 Experiments

EST_{MSR} method is effective on problem domains that fulfil two assumptions. The first assumption is explained in the previous section as determinism and an unchanging environment. For this reason, all selected problem domains have deterministic and unchanging nature. The other assumption is that, the selected problem domain shall have at least one memoryless policy solution, which is implicitly imposed by SARSA(λ), meaning that there must be something *learnable* through observations only, for the learning algorithm to be successful.

- One of our benchmark problems is tiny navigation environment (*Mini-hall*) [33], as described in Section 3.3 (Figure 3.2).

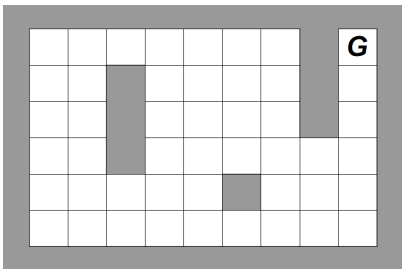


Figure 4.3: *Sutton's grid world* domain. G represents the goal state [32].

- A larger problem is *Sutton's grid world* [61], which is a 9×6 grid world with surrounding and some interior walls (Figure 4.3). The agent starts with any of the cells, and it can move in any of the four compass directions at a time, trying to reach the goal state at the north-east corner of the world. The environment responds with a 0 reinforcement signal for each movement, except upon reaching the goal cell, the agent receives a reward of $+1$.

The original problem is fully observable, and the partially observable variant is introduced in [32]. Partial observability is defined as observing the neighbouring 8 grid cells.

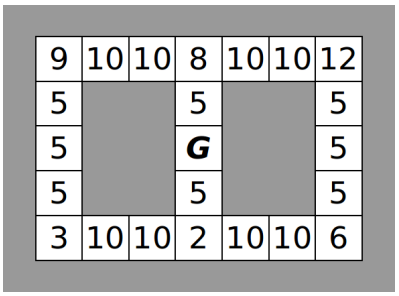


Figure 4.4: *McCallum's maze* domain. G is the goal state, and each number indicate the observation sensed by the agent at the given state [38].

- Final problem domain is *McCallum's maze* [38]. It is a 7×5 maze with narrow hallways, in which agent is only able to sense its surrounding walls. Agent starts an episode from one of the randomly assigned corners. Actions are movement steps to the four compass directions, as in Sutton's grid world. The environment responses to the agent with a reinforcement of -1.0, if an action attempts to move into the wall, +5.0 upon reaching the goal state, and -0.1 for any other action. Figure 4.4 is an illustration of the domain together with observation identifiers corresponding to states.

  McCallum's maze is a widely used benchmark problem due to its highly ambiguous nature. Note that, the states on north-to-south and west-to-east corridors (observed as 5 and 10, respectively) are observed to be the same.

  Although this problem domanin has no optimal memoryless policy solution, a near optimal policy might perform fairly well under a stochastic action selection mechanism, like ϵ-Greedy.

4.3.1 Setup

Table 4.1 summarizes the experimented problem domains, providing the sizes of problems in terms of state, action and observation spaces, and the reference publication for the domain.

Learning settings used for experimentation are given in Table 4.2. For every problem domain,

Table 4.1: Problem Domains

Problem	**Sizes**			**Ref.**
	$\|\mathbf{S}\|$	$\|\mathbf{A}\|$	$\|\mathbf{\Omega}\|$	
Mini-hall	13	3	9	[33]
Sutton's maze	47	4	31	[32]
McCallum's maze	23	4	9	[38]

Table 4.2: Learning Settings

Parameter	Value
α	0.01
γ	0.9
λ	0.9
ϵ	0.1
ψ_{decay}	0.95
ξ_{decay}	0.99
$\psi_{treshold}$	0.01
$\xi_{treshold}$	0.01

250 experiments are executed and the results are averaged over episodes of each run. ϵ-Greedy exploration strategy is invoked in action selection for all problems, with a constant ϵ value.

"Reward-per-step" is the performance criterion for *Mini-hall* and *McCallum's maze* domain, while "number of steps to goal" is the correct measure for evaluation in *Sutton's grid world* domain. For visual clarity, the resulting plots are smoothed.

4.3.2 Results and Discussion

In general, results show that EST_{MSR} performs well and increases learning performance for the selected problem domains.

In the tiny navigation environment, EST_{MSR} boosts learning starting with the very early stages of the experiment. Moreover, in the long term, learned abstractions assist the agent to increase average reward obtained, since the refined abstract actions contain no observation ambiguity, and can by-pass the stochastic action decision semantics (Figure 4.5).

Also in *Sutton's grid world* domain, EST_{MSR} supports the underlying SARSA(λ) algorithm by increasing learning performance at the very beginning of the episode (Figure 4.6). The hallway near the goal state (see Figure 4.3) constitutes a useful sub-policy pattern that should exist in every possible solution, since the agent should pass through these states –each of which possess a unique discrimination observation– in order to achieve the goal state. Thus, the agent is able to make use of abstractions starting immediately after the initial construction of HA-EST. Pruning does its job very well, trying to keep the HA-EST data structure full of useful and unambiguous sub-policies.

EST_{MSR} performs surprisingly well for *McCallum's maze*, despite it has no memoryless solution (Figure 4.7). Of course, the success in each episode comes from the stochastic nature of the applied action selection mechanism, namely ϵ-Greedy. In order to better understand how EST_{MSR} provides the performance improvement, consider the HA-EST data structure given in Figure 4.8, referring Figure 4.4 as a guide. HA-EST has saturated after many episodes, and tells us that the only useful and unambiguous memoryless sub-policies for this domain are

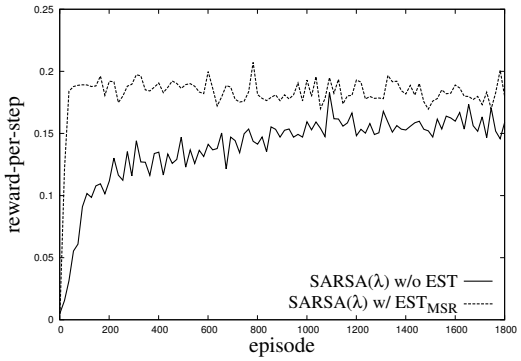


Figure 4.5: Experiment results for *Mini-hall* domain.

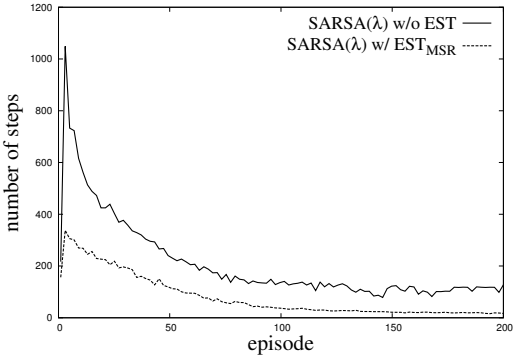


Figure 4.6: Experiment results for *Sutton's grid world* domain.

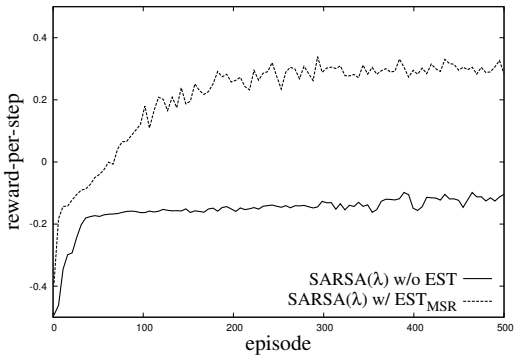


Figure 4.7: Experiment results for *McCallum's maze* domain.

o2-N-o5-N-G and o8-S-o5-S-G ("oi" stands for observation with id i, "N" for north, "S" for south). If this guidance of HA-EST were absent, since o5 is an ambiguous observation, no single action would lead the agent to the goal state either from o2 or from o8. By EST_{MSR}, the agent is now aware that o2 and o8 are discriminating observations that lead to useful sub-policies.

Table 4.3 is a comparison of several metrics collected and averaged over episodes for each domain.

- Average use of options shows the average percentage of actions that are invoked within an option (i.e. abstract action) compared to the total number of actions executed in the episode.

  Mini-hall seem to make most use of temporal abstraction compared to other domains, since the useful abstraction near the goal state is relatively long. Once the EST_{MSR} extracts that abstraction, it makes extensive advantage from it.

- Average number of HA-EST nodes is an indicator of the average memory footprint the EST_{MSR} mechanism uses, for every episode.

  McCallum's maze has the smallest footprint, which is consistent with the relatively less abstraction potential, as discussed before. *Sutton's grid world*, on the other hand,

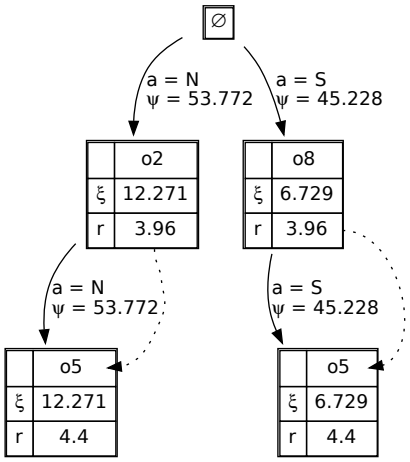


Figure 4.8: HA-EST data structure for McCallum's Maze domain at the end of an experiment.

triggers a deeper and wider branching of HA-EST in accordance since it has the most number of distinct observations in our problem set.

- Average CPU usage metric is the CPU time in milliseconds, that is used by the algorithms on the average for an experiment to finish.

  Generally speaking, EST_{MSR} may reduce total CPU time required for learning a reactive policy, if it can significantly reduce the number of learning steps at all. The CPU time saving is in fact a consequence of a trade-off between EST_{MSR} overhead and the gain obtained in terms of number of learning steps. In *Mini-hall* problem, for example, obviously the cost of EST_{MSR} mechanism exceeds the gain.

With the MSR mechanism, EST_{MSR} rapidly prunes sub-policies that would cause wrong abstractions, by making use of agent's immediate experiences. Meanwhile, EST_{MSR} successfully supports the agent by providing and testing its potentially useful abstractions throughout the learning process. Moreover, although SARSA(λ) is selected here as a powerful representative of its category, by making minor modifications, it is always possible to replace SARSA(λ) with any underlying observation based memoryless reinforcement learning algorithm (like Q-Learning, Q(λ), SARSA, TD(λ) etc.).

EST_{MSR} also has some shortcomings that need improvement. First of all, as mentioned before, the method is effective on deterministic and stationary problems only. A non-determinism

Table 4.3: Comparison of some experiment metrics

Problem	avg. use of options (%)	avg. num. of HA-EST nodes	avg. CPU usage (msec)	
			without EST_{MSR}	with EST_{MSR}
Mini-hall	59	28.83	351	471
Sutton's grid world	18	407.65	13065	5005
McCallum's maze	11	5.18	23329	6038

factor in the environment can easily cause complete pruning of the HA-EST, resulting in an ineffective abstraction mechanism.

Another problem is that, EST_{MSR} fails to extract intermediate abstractions (i.e. sub-policies that are free of perceptual aliasing problem, but do not lead to a goal state) in the environment. In other words, due to its operational nature, it focuses on abstractions near the goal state, tending to eliminate any macro that is far earlier than the goal region.

CHAPTER 5

ENHANCING UTILE SUFFIX MEMORY ALGORITHM

We demand rigidly defined areas of doubt and uncertainty!

– Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

In this chapter, EST_{MSR} is used to speed up Utile Suffix Memory (USM) algorithm by means of automatic temporal abstractions through observations. Although EST_{MSR} is designed to improve memoryless reinforcement learning algorithms, it is quite possible to invoke EST_{MSR} on top of a memory based algorithm like USM. Moreover, we present a way to extend EST_{MSR} to integrate the USM tree data structure, so that the resulting method can remember and make use of some intermediate abstractions that EST_{MSR} would typically prune immediately.

We will call the new method $EST_{MSR/USM}$ to emphasize the mutual dependency of MSR pruning mechanism and USM tree data structure on the way to success.

5.1 Modifying EST_{MSR} for Utile Suffix Memory Algorithm

EST_{MSR} has some limitations. As described in Section 4.3.2, possibly the most important one is that, although it generates some intermediate abstractions at the early steps of learning, it prunes them later due to its all-or-nothing nature. Thus, EST_{MSR} can not generate intermediate temporal abstractions at all.

Before the $EST_{MSR/USM}$ method is described, it can be useful to more formally define the USM's suffix memory mechanism.

A history instance represented by USM at time step t is a transition $I_t = \langle I_{t-1}, a_{t-1}, o_t, r_t \rangle$, and is deposited in the leaf node whose suffix, σ, matches some suffix of the actions and observations of the transition instances that precede I_t in time. In other words, a transition I_t belongs to the leaf with a label that is some suffix of $[...o_{t-3}a_{t-3}o_{t-2}a_{t-2}o_{t-1}]$. The set of instances associated with the leaf labelled σ is written $I_t(\sigma)$. The suffix tree leaf which instance I_t belongs to is written $L(I_t)$.

Algorithm 10 GENERATE-PROBABLE-HISTORIES(h)

Require: h is a history of the form $o_1 a_1 r_2 ... o_{t-1} a_{t-1} r_t o_t$

1: $best[L_{t-1}] \leftarrow L_{t-1} a_{t-1} r_t L_t$ ▷ $best$ holds current most promising history candidates
2: $R[L_{t-1}] \leftarrow r_t$ ▷ $R[L_t]$ holds the total cumulative reward for $best[L_t]$
3: **for** $i \leftarrow t-2$ down to 1 **do**
4: ▷ from rear to front
5: **if** $R[L_i]$ is not set or $r_{i_1} + \gamma R[L_{i+1}] > R[L_i]$ **then**
6: ▷ if L_i is either not encountered before or has a lower return estimate
7: $best[L_i] \leftarrow L_i a_i r_{i+1} \circ best[L_{i+1}]$ ▷ create or update the candidate history corresponding to state L_i.
8: $R[L_i] \leftarrow r_{i+1} + \gamma R[L_{i+1}]$ ▷ update maximum reward.
9: **end if**
10: **end for**
11: let $best_o$ be a multimap ▷ reduce all history entries to single observations
12: **for** every element of $best$ indexed by l **do**
13: $h_o \leftarrow$ apply ω to state entries of all transition in $best[l]$
14: $best_o[\omega(l)] \leftarrow h_o$
15: **end for**
16: **return** $best_o$

As an example, recall the USM tree given in Figure 2.8. It is nothing but a Q table for state estimations represented by suffixes $a0, 1b0, c0, 1$. For instance, suppose the current history of the agent is defined by the transition $I_{current} = [...0a1b0]$; then the internal information state of the agent is the leaf node representing the suffix $L(I_{current}) \equiv 1b0$.

The original EST mechanism makes use of state equivalences during the construction of useful history portions, based on experiences. For this equivalence test, EST_{MSR} directly uses observations instead of states, and prunes the paths that are experienced to be misleading later on. USM data structure, on the other hand, is capable of discriminating observation instances, dynamically updated throughout learning. Most of the time, USM method can discriminate some observations at the early stages of learning. With the USM data structure, the state equivalence test of EST can be carried on using USM states for the given history instances. By this way, beginning at the very early stages, probable history generation can be done more effectively using *suffixes*, or equivalently by using the leaf nodes of USM.

For this purpose, the procedure responsible for probable history generation of EST_{MSR} is modified to make the state equivalence test through USM leaf nodes instead of sole observations, as seen in Algorithm 10. L_i stands for $L(I_i^h)$ where I_i^h is the instance of the history h at time i. $\omega(L)$ gives the observation of the instance represented by L. Note that, the resulting continuation set elements of HA-EST data structure still involve single observations, not suffixes or observation-action sequences. This will give $EST_{MSR/USM}$ mechanism the opportunity to be invoked more frequently, especially at the beginning of learning, compared to a hypothetical USM instance based design. All of the other mechanisms of EST_{MSR} remain

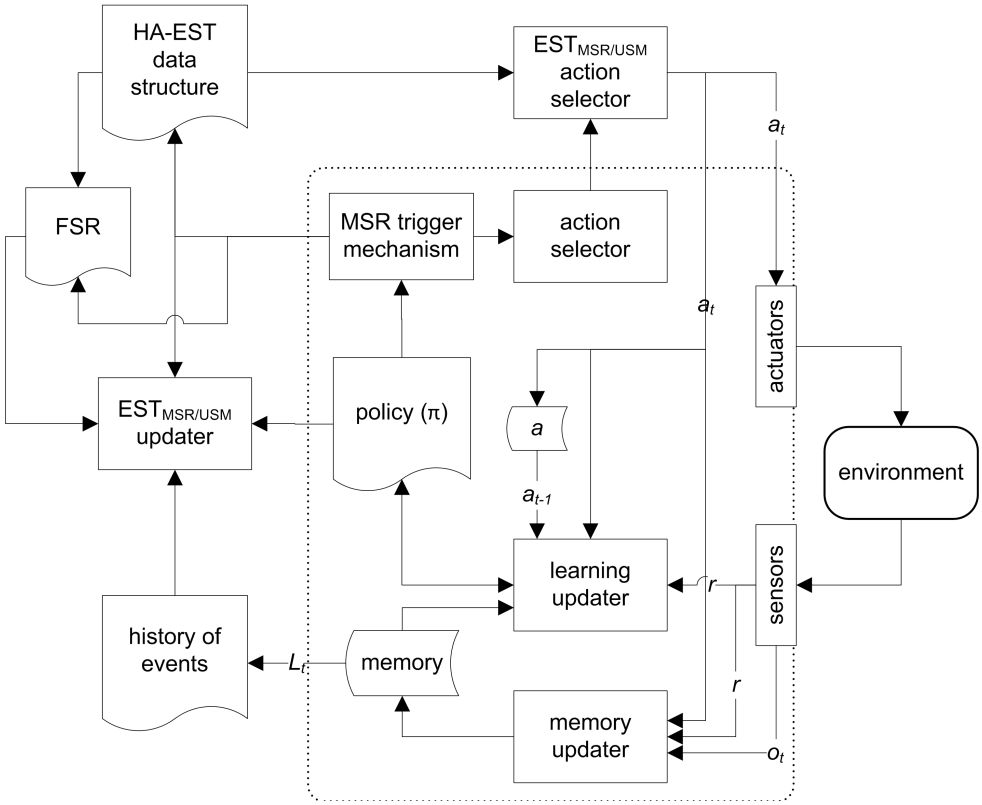


Figure 5.1: A structural view for the USM algorithm together with $EST_{MSR/USM}$ extension.

unchanged.

With $EST_{MSR/USM}$, much deeper HA-EST trees are generated and used for option exploitation. Probability of catching and making use of repeating observation sequences in the domain that are useful on the path to goal increases, which is not possible for EST or EST_{MSR} for partially observable problems. On the average, since the number of option paths will increase in the long term (i.e. pruning mechanism will be less eager, since more HA-EST paths will succeed to reach the goal), one can expect $EST_{MSR/USM}$ to give better results than EST_{MSR} for USM as the underlying reinforcement learning algorithm.

An important property of $EST_{MSR/USM}$ is that it reduces to ordinary EST if the problem is fully observable and deterministic, and USM fringe depth parameter is set to zero.

A structural view of $EST_{MSR/USM}$ extension to USM algorithm is given in Figure 5.1. Note that some mechanisms and data structures like HA-EST, FSR and MSR trigger are inherited from EST_{MSR}. Observation flows, on the other hand, are enhanced by USM memory extensions this time, enriching the history generation procedure.

5.2 Experiments

Like EST_{MSR}, $EST_{MSR/USM}$ method is effective on problem domains that have deterministic nature. Performance of USM both with EST_{MSR} and $EST_{MSR/USM}$ are experimented against USM alone, using the following deterministic benchmark problems:

- First one of the benchmark problems is, again, the *tiny navigation environment* (*Mini-hall*) [33], as described in Section 3.3 (Figure 3.2). This problem is not only the simplest of our set, it also possesses a structure that helps EST_{MSR} make use of full advantage of its nature: the single useful temporal abstraction package is adjacent to the goal state.

- *Virtual Office* [18] problem is the next problem that is used for experimentation. The problem characteristics are briefly described in Section 3.3 (Figure 3.4). The importance of this problem in our set is that it is relatively large, has multiple goals and bottleneck states, imposing a hierarchical solution.

- The next problem domain is *McCallum's maze*, which is described in in Section 3.3 [38] (Figure 4.4). As in the *Mini-hall* problem, there are useful temporal actions near the goal state, which are known to be successfully identifiable by EST_{MSR} (see Figure 4.8). However, there are other useful intermediate abstractions in the domain that are expected to be captured and used by the $EST_{MSR/USM}$ method.

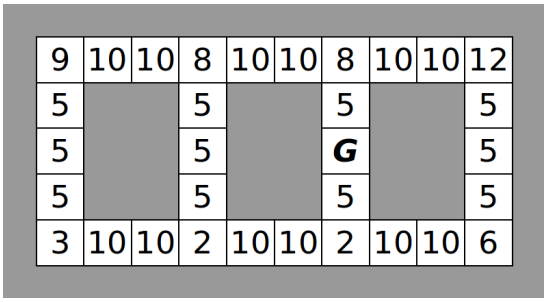


Figure 5.2: An extended version of McCallum's maze, specially designed for testing $EST_{MSR/USM}$.

- Finally, an extension of *McCallum's maze*, specially designed for testing the effectiveness of $EST_{MSR/USM}$ is proposed, as illustrated in Figure 5.2. The transition and observation semantics is identical with the original problem. The main difference here is that, there is no discriminating observation near the goal state, as opposed to the original *McCallum's maze*. While observations identified by 2 and 8 are uniquely disambiguates their corresponding states in the original maze, there are two of them both in this domain. So, ambiguity of the observations are spread to all of the states except the starting states (i.e. the four corners). In other words, there are no distinguishable abstraction packages near the goal state, thus an abstraction procedure should make use of intermediate abstractions to achieve a speed-up.

Table 5.1: Problem Domains

Problem	Sizes			Ref.
	$\|\mathbf{S}\|$	$\|\mathbf{A}\|$	$\|\mathbf{\Omega}\|$	
Mini-hall	13	3	9	[33]
Virtual Office	38	4	12	[18]
McCallum's maze	23	4	9	[38]
McCallum's maze extended	32	4	9	-

5.2.1 Setup

Table 5.1 summarizes the experimented problem domains, providing the sizes of problems in terms of state, action and observation spaces, and the reference publication for the domain.

Learning settings used for experimentation are given in Table 5.2. ϵ-Greedy is used as the action selection strategy for all problems, with a constant ϵ value. Kolmogorov-Smirnov (K-S) test for USM algorithm (as defined in [38]) is executed after each time step.

For every problem domain, 100 experiments are executed and the results are averaged over the episodes of each run. "Reward-per-step" is the performance criterion for success. For visual clarity, result plots are smoothed.

5.2.2 Results and Discussion

In general, results show that USM can benefit from both EST_{MSR} and $EST_{MSR/USM}$ to increase the learning performance for the selected problem domains.

In the *Mini-hall* domain, both abstraction mechanisms boost learning, beginning with the very early steps of the experiments (Figure 5.3). $EST_{MSR/USM}$ performs better at the beginning of the experiment, but falls behind EST_{MSR} later on. This is due to a property of $EST_{MSR/USM}$ falling short sometimes: At the early stages of learning, $EST_{MSR/USM}$ uses a relatively immature USM construct to derive state estimations, which may not be de-

Table 5.2: Learning Settings

Parameter	Value
α	0.125
γ	0.9
ϵ	0.1
K-S test treshold	0.01
max. fringe depth for USM	4
ψ_{decay}	0.95
ξ_{decay}	0.99
$\psi_{treshold}$	0.01
$\xi_{treshold}$	0.01

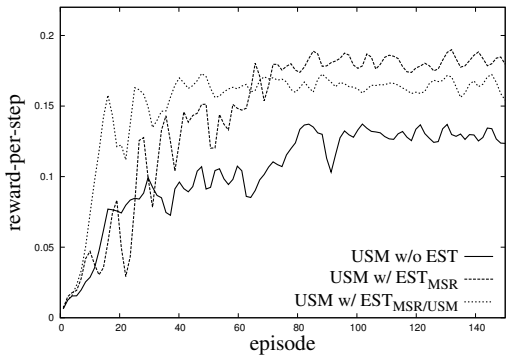


Figure 5.3: Experiment results for *Mini-hall* domain.

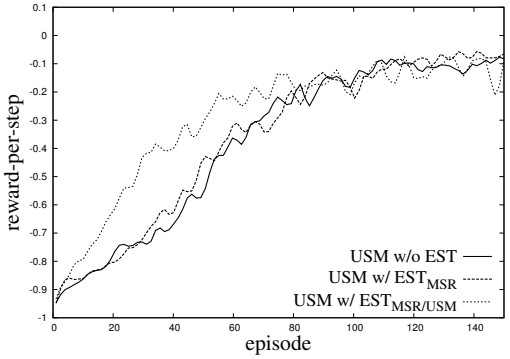


Figure 5.4: Experiment results for *Virtual Office* domain.

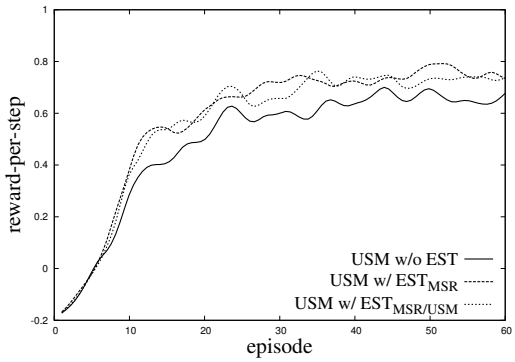


Figure 5.5: Experiment results for *McCallum's maze* domain.

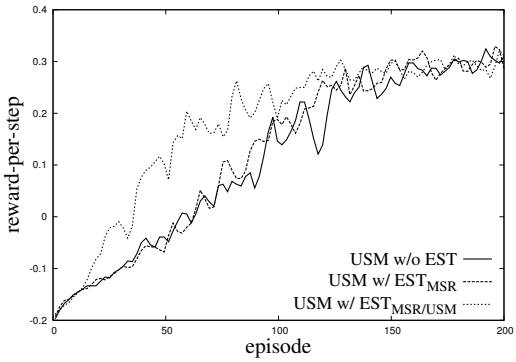


Figure 5.6: Experiment results for *McCallum's maze extended* domain.

tailed enough to give a meaningful discrimination over observations. At the same time, it can sometimes be the case that, using immature state estimations, $EST_{MSR/USM}$ derives observation-action sequences that lead to a goal state, but is longer than it should be, due to some redundant loops ending with the same observations. Although longer paths assist well at the beginning of the experiment, since $EST_{MSR/USM}$ fails to detect the loops, it may spend extra time to exploit these redundant option steps, which is the case in the tiny navigation problem. In the long term, learned abstractions assist the agent to increase average reward obtained for both abstraction methods, since the refined abstract actions contain no observation ambiguity, and can by-pass the stochastic action decision semantics (i.e. ϵ-Greedy) of USM.

In *Virtual Office* domain, $EST_{MSR/USM}$ performs significantly better than EST_{MSR} (Figure 5.4). When the transition dynamics of the problem is examined, it can be identified that each of the two goal states are reached through long observation-action sequences with ambiguous observations, which are better maintained by $EST_{MSR/USM}$ method in the long term.

Both abstraction mechanisms perform well for *McCallum's maze*, which is specially designed by its creator to demonstrate the power of USM algorithm (Figure 5.5). In other words, this problem is where USM shines by itself. Nevertheless, both EST_{MSR} and $EST_{MSR/USM}$ improves learning performance of USM successfully. However, there is no significant differ-

Table 5.3: Comparison of some experiment metrics

Problem	avg. use of options (%)		avg. num. of HA-EST nodes	
	EST_{MSR}	$EST_{MSR/USM}$	EST_{MSR}	$EST_{MSR/USM}$
Mini-hall	43	68	26.46	65.80
Virtual Office	6	38	3.09	515.49
McCallum's maze	18	51	5.97	33.18
McCallum's maze extended	1	43	1.43	207.01

ence between performance increase provided by two abstraction methods.

In *McCallum's maze extended*, on the other hand, EST_{MSR} loses its power due to the design property of the domain as described previously. Moreover, its performance is almost the same as the USM algorithm alone. $EST_{MSR/USM}$ significantly outperforms USM and EST_{MSR}, especially at the early stages of learning.

Metrics given in Table 5.3 supports the argument that $EST_{MSR/USM}$ extracts and makes use of more abstractions (in fact, intermediate abstractions) than EST_{MSR}. $EST_{MSR/USR}$ not only generates more HA-EST nodes on the average (up to 150 times), but also takes advantage of those abstractions by invoking them properly. An interesting result to note is EST_{MSR} values of *McCallum's maze extended*, which did not make use of any abstractions at all, as expected.

Average total CPU times elapsed for all experiments are given in Table 5.4. In almost all cases, USM with no EST takes more time than the cases with abstraction. In fact, these results mostly depend on the USM learning parameters, and the nature of the selected problem domain. Cost of USM tree maintenance (statistical tests, fringe promotions, and history links etc.) seems to dominate the cost of maintaining EST data structure, in such a way that gain achieved by EST_{MSR} or $EST_{MSR/USM}$ (in terms of number of steps to goal) can drastically reduce total time spent. Obviously, these results could have been completely different if some other USM learning setting were used. Nevertheless, these results show that, EST_{MSR} and $EST_{MSR/USM}$ can provide some gain also in terms of CPU time.

An example HA-EST data structure for *McCallum's maze* is given in Figure 5.7 (refer to Figure 4.4 as a guide to observation identifiers). Dotted arrows represent links from the parent continuation set elements. This is a tree (much larger than a HA-EST tree derived by EST_{MSR}, as in Figure 4.8) that can incorporate intermediate abstractions even if the corre-

Table 5.4: CPU usage in milliseconds

Problem	USM	USM w/ EST_{MSR}	USM w/ $EST_{MSR/USM}$
Mini-hall	2498	2792	1917
Virtual Office	46621	43241	34146
McCallum's maze	997	823	755
McCallum's maze extended	234434	238939	106960

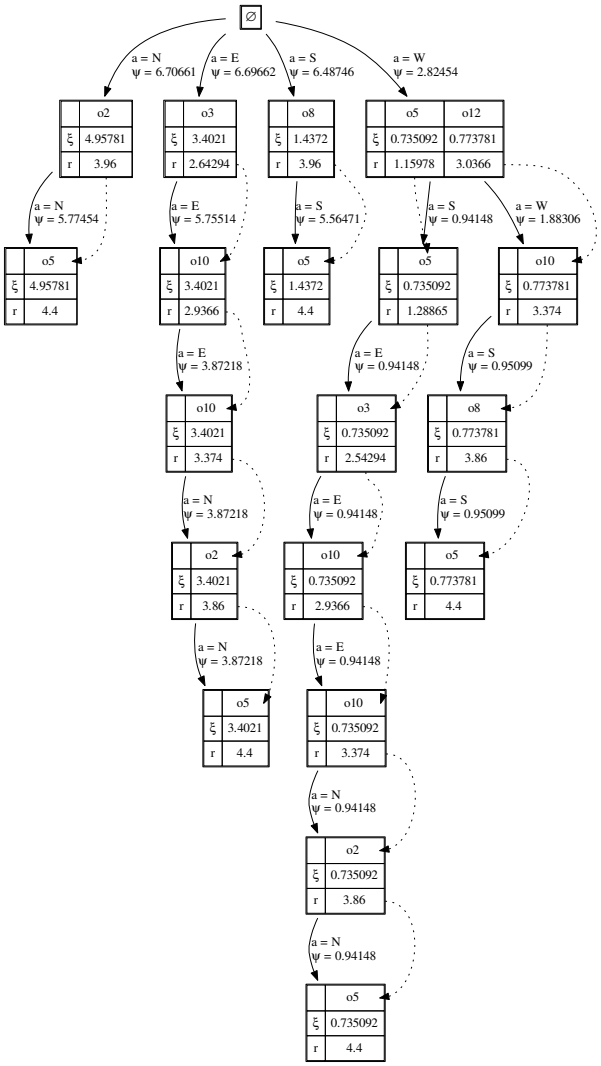


Figure 5.7: HA-EST data structure for *McCallum's maze* domain at an intermediate step of experiment with $EST_{MSR/USM}$. The rightmost branch (o12-W-o10-W-o8-S-o5-S) is misleading (since o10 is not observed twice) and will be pruned as soon as it is executed. All other branches will remain.

sponding observations included are ambiguous. However, a side effect of this property is its disability to detect redundant loops within the extracted options.

MSR is a mechanism for EST to rapidly prune sub-policies that would cause wrong abstractions, by making use of agent's immediate experiences through observations. While EST_{MSR} successfully supports the agent by providing and testing its potentially useful abstractions, $EST_{MSR/USM}$ can make an improvement on the quality of the derived option, especially by covering intermediate abstractions missed by EST_{MSR}.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This thesis proposes methods to accelerate well known reinforcement learning algorithms that are designed for partial observable problems, by modifying the original EST abstraction method.

BEST expands the existing EST abstraction method to cover model based reinforcement learning algorithms designed for POMDPs, via discretization of belief space. Some belief discretization methods suitable for use in BEST are also proposed. Effectiveness of BEST together with the discretization methods are shown via experimentation over six partially observable problem domains, and the results are discussed extensively.

An obvious future research direction for BEST is the automatic extraction or incremental adjustment of discretization parameters. Additionally, application of other belief discretization schema would be valuable. Expanding this study to cover more model based partially observable reinforcement learning algorithms, like SPOVA is another challenging research direction.

Another reinforcement learning paradigm focused in this thesis is model free partially observable setting. As the representative algorithm under memoryless agent assumption, SARSA(λ) is enhanced by temporal abstractions. The proposed method, namely EST_{MSR}, performs well and improves learning performance significantly, under certain assumptions. The method is experimented on three problems with hidden state, and the results are discussed.

In the last part of the thesis, another EST variant is proposed to accelerate reinforcement learning for partial observability, assuming memory based model free setting. Since EST_{MSR} fails to effectively handle intermediate abstractions, a modification on EST_{MSR} is proposed to enhance a well known algorithm of this category, namely USM. The proposed abstraction method, $EST_{MSR/USM}$, makes use of USM state information, and thus performs very well to leverage USM learning. Experimentation comparing the performances of USM, USM with EST_{MSR} and USM with $EST_{MSR/USM}$ is carried out with four problem domains and the results are discussed.

One of the major drawbacks of $EST_{MSR/USM}$ is that it fails to catch redundant repetitions of observation-action sequences. An alternative solution can be using fringe leaf nodes in-

stead of official leaf nodes in history generation, however, it may cause over-discrimination of estimated states, harming the quality of options in the long term.

Unfortunately, both EST_{MSR} and $EST_{MSR/USM}$ fail to handle non-determinism and changes in the environment. One of the promising research directions is to relax this requirement, potentially by incorporation of some statistical methods for the pruning mechanism.

All of the EST extensions proposed in this thesis are generalized versions of the original EST, just like POMDP is a generalization of MDP. In other words, under certain conditions, the proposed methods reduce to EST. In that sense, although the picture is still not complete, this study can be seen as a comprehensive attempt to generalize automatic temporal abstraction to reinforcement learning in partially observable domains.

REFERENCES

[1] E. Alpaydın. *Introduction to Machine Learning*. The MIT Press, 2004.

[2] L. Baird and A. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 11, pages 968–974. MIT Press, 1999.

[3] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[4] J. Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125, 1994.

[5] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[6] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Uncertainty in Artificial Intelligence*, pages 33–42. Morgan Kaufmann, 1998.

[7] S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances In Neural Information Processing Systems*, volume 7, pages 393–400. MIT Press, 1994.

[8] A. R. Cassandra. *Exact and approximate algorithms for partially observable markov decision processes*. Ph.D. thesis, Brown University, 1998.

[9] A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 963–972, 1996.

[10] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 2, pages 1023–1028. AAAI Press, 1994.

[11] L. Charlin, P. Poupart, and R. Shioda. Automated hierarchy discovery for planning in partially observable environments. In *Advances in Neural Information Processing Systems*, volume 19, pages 225–232. MIT Press, 2007.

[12] L. Chrisman. Reinforcement learning with perceptual aliasing: the perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI Press, 1992.

[13] E. Çilden and F. Polat. Abstraction in model based partially observable reinforcement learning using extended sequence trees. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 348–355, 2012.

[14] E. Çilden and F. Polat. Generating memoryless policies faster using automatic temporal abstractions for reinforcement learning with hidden state. In *IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 719–726, 2013.

[15] P. A. Crook. *Learning in a State of Confusion: Employing Active Perception and Reinforcement Learning in Partially Observable Worlds*. Ph.D. thesis, University of Edinburgh, 2006.

[16] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[17] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[18] L. T. Dung, T. Komeda, and M. Takagi. Reinforcement learning in non-markovian environments using automatic discovery of subgoals. In *SICE, 2007 Annual Conference*, pages 2601–2605, 2007.

[19] J. P. Forestier and P. Varaiya. Multilayer control of large markov chains. *IEEE Transactions on Automatic Control*, 23(2):298 – 305, 1978.

[20] S. Girgin. *Abstraction in reinforcement learning*. Ph.D. thesis, Middle East Technical University, 2007.

[21] S. Girgin, F. Polat, and R. Alhajj. Improving reinforcement learning by using sequence trees. *Machine Learning*, 81(3):283–331, 2010.

[22] A. Gosavi. Reinforcement learning: a tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

[23] M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[24] R. He, E. Brunskill, and N. Roy. PUMA: planning under uncertainty with macro-actions. AAAI Press, 2010.

[25] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann Publishers Inc., 2002.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[27] T. Jaakkola, S. P. Singh, and M. I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems*, volume 7, pages 345–352. MIT Press, 1995.

[28] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[29] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[30] L.-J. Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 2*, pages 781–786. AAAI Press, 1991.

[31] L.-J. Lin and T. M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical report, Carnegie Mellon University, 1992.

[32] M. L. Littman. Memoryless policies: theoretical limitations and practical results. In *From Animals to Animats 3: Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pages 238–245. MIT Press, 1994.

[33] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 495–503. Morgan Kaufmann Publishers Inc., 1998.

[34] J. Loch and S. P. Singh. Using eligibility traces to find the best memoryless policy in partially observable markov decision processes. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 323–331. Morgan Kaufmann Publishers Inc., 1998.

[35] W. S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operational Research*, 39(1):162–175, 1991.

[36] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning*, pages 71–78. ACM, 2004.

[37] O. Maron. *Learning from Ambiguity*. Ph.D. thesis, Massachusetts Institute of Technology, 1998.

[38] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, University of Rochester, 1996.

[39] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368. Morgan Kaufmann Publishers Inc., 2001.

[40] E. A. McGovern. *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. Ph.D. thesis, University of Massachusetts Amherst, 2002.

[41] K. P. Murphy. A survey of POMDP solution techniques. Technical report, University of California, Berkeley, 2000.

[42] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1088–1094. Morgan Kaufmann Publishers Inc., 1995.

[43] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances In Neural Information Processing Systems*, volume 10, pages 1043–1049. MIT Press, 1998.

[44] R. E. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. thesis, University of California, 1998.

[45] M. D. Pendrith and M. McGarity. An analysis of direct reinforcement learning in non-markovian domains. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 421–429. Morgan Kaufmann Publishers Inc., 1998.

[46] L. Peshkin, N. Meuleau, and L. P. Kaelbling. Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 307–314. Morgan Kaufmann Publishers Inc., 1999.

[47] J. Pineau. *Tractable planning under uncertainty: exploiting structure*. Ph.D. thesis, Carnegie Mellon University, 2004.

[48] J. Pineau and S. Thrun. An integrated approach to hierarchy and abstraction for POMDPs. Technical Report CMU-RI-TR-02-21, The Robotics Institute at Carnegie Mellon University, 2002.

[49] N. Roy. *Finding Approximate POMDP Solutions through Belief Compression*. Ph.D. thesis, Carnegie Mellon University, 2003.

[50] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[51] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, second edition, 2003.

[52] G. Shani. *Learning and Solving Partially Observable Markov Decision Processes*. Ph.D. thesis, Ben-Gurion University of the Negev, 2007.

[53] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[54] O. Sigaud and O. Buffet. *Markov Decision Processes in Artificial Intelligence*. ISTE - Wiley, 2010.

[55] Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, pages 95–102. ACM, 2004.

[56] T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 520–527. AUAI Press, 2004.

[57] E. J. Sondik. *The optimal control of partially observable Markov decision processes*. Ph.D. thesis, Stanford University, 1971.

[58] M. Stolle and D. Precup. Learning options in reinforcement learning. In S. Koenig and R. C. Holte, editors, *Abstraction, Reformulation, and Approximation*, volume 2371, pages 212–223. Springer Berlin / Heidelberg, 2002.

[59] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[60] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[61] R. S. Sutton. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann Publishers Inc., 1990.

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

[63] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[64] G. Theocharous and L. P. Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances In Neural Information Processing Systems*, volume 16, pages 775–782. MIT Press, 2004.

[65] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable markov decision process models for robot navigation. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 511–516, 2001.

[66] C. Watkins. *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University, 1989.

[67] S. Whitehead and D. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[68] M. Wiering and J. Schmidhuber. HQ-Learning. *Adaptive Behavior*, 6:219–246, 1997.

[69] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[70] T. Yoshikawa and M. Kurihara. An acquiring method of macro-actions in reinforcement learning. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 4813–4817, 2006.

[71] R. Zhou and E. A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, volume 1, pages 707–716. Morgan Kaufmann Publishers Inc., 2001.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Çilden, Erkin
Nationality: Turkish (TC)
Date and Place of Birth: 3 October 1976, Ankara
Marital Status: Married
e-mail: ecilden@ceng.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU Department of Computer Engineering	2001
B.S.	METU Department of Computer Engineering	1998
High School	Ankara Atatürk Anatolian High School	1994

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
Sep 2008 - Feb 2011	MilSOFT-ICT A.Ş.	Senior Software Engineer
Jan 2006 - Aug 2008	MilSOFT A.Ş.	Lead Software Engineer
Mar 2006 - Nov 2006	TSK Harp Akademileri Komutanlığı Atatürk Harp Oyunu ve Simülasyon Merkezi	IT Officer
Feb 2002 - Apr 2004	METU-TSK MODSIMMER	Part-time Researcher
Aug 1998 - Sep 2004	METU Department of Computer Engineering	Research Assistant
1997	ERE A.Ş.	Summer Practice
1996	ASELSAN A.Ş.	Summer Practice

PUBLICATIONS

International Conference Publications

1. E. Çilden and F. Polat. *Generating memoryless policies faster using automatic temporal abstractions for reinforcement learning with hidden state.* In IEEE 25th International Conference on Tools with Artificial Intelligence, ICTAI '13, pages 719-726, Washington, DC, USA, Nov. 2013.

2. E. Çilden and F. Polat. *Abstraction in model based partially observable reinforcement learning using extended sequence trees.* In 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT), volume 2, pages 348-355, Macau, China, Dec. 2012.

National Conference Publications

1. F. Polat, A. Coşar, S. Girgin, M. Tan., E. Çilden, M. Balcı, E. Göktürk, E. Kapusuz, V. Koç, A. Yavaş, Ç. Ündeğer, "Küçük Ölçekli Harekatın Modellenmesi ve Simülasyonu," USMOS 2005, Turkey, June 2005.