

EPISODIC TASK PLANNING AND LEARNING  
IN PERVASIVE ENVIRONMENTS

by  
YONG LIN

Presented to the Faculty of the Graduate School of  
The University of Texas at Arlington in Partial Fulfillment  
of the Requirements  
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2010

Copyright © by YONG LIN 2010

All Rights Reserved

## ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Fillia Makedon for constantly motivating and encouraging me, and also for her invaluable advice during the course of my doctoral studies. She guides me to the research towards the intelligent robot in the area of pervasive assistive environments. I also wish to thank my academic advisors Dr. Chris Ding, Dr. Heng Huang, Dr. Gutemberg Guerra-Filho and Dr Yonghe Liu for their interest in my research and for taking time to serve on my dissertation committee. My research using Markov decision processes and game theory for the autonomous robot is after the suggestion of Prof. Ding. I also want to thank Prof. Huang for his guidance of machine learning and classification techniques. His knowledge and insights were very helpful to me.

I would also like to extend my appreciation to the Computer Science and Engineering department of the University of Texas at Arlington, for providing financial support for my doctoral studies. I am especially grateful to Dr. Manfred Huber, who opened the gateway and aroused my interest of the area of uncertainty and artificial intelligence. I am grateful to all the teachers who taught me during the years I spent in school, first in China, and finally in the Unites States. I would like to thank Dr. Shuwang Lu for encouraging and inspiring me to pursue graduate studies.

I joined the Heracleia Human-Centered Computing Laboratory in May 2008. From that time on, the working environment of the lab boosts my interest in research for methods of pervasive assistive environments. I thank all the members of the Heracleia lab. Especially, I wish to thank Dr. Zhengyi Le, Dr. Eric Becker for their support and encouragement, so that I can start the research on assistive technologies.

Finally, I would like to express my deep gratitude to my family who has encouraged and inspired me and sponsored my undergraduate and graduate studies. I am extremely fortunate to be so blessed. I am also extremely grateful to my mother, sister and wife for their sacrifice, encouragement and patience. I also thank several of my friends who have helped me throughout my career.

This work is supported in part by the National Science Foundation under award number CT-ISG 0716261, MRI 0923494 and CPS 1035913. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

November 18, 2010

## ABSTRACT

### EPISODIC TASK PLANNING AND LEARNING IN PERVASIVE ENVIRONMENTS

YONG LIN, Ph.D.

The University of Texas at Arlington, 2010

Supervising Professor: Fillia Makedon

During planning and control of autonomous robots in a pervasive environment designed to serve people, we will inevitably face the situations of needing to perform multiple complex tasks. Management and optimization of the execution of complex tasks involve the design of efficient approach and framework based on algorithm, artificial intelligence, machine learning, cognitive science, etc. In this dissertation, we have developed a new method for complex task planning of robots, so that they can improve the service for the elderly and the disabled. The word “episode” comes from the Greek word “*επεισοδισο*”, which means “event”, or “occurrence”. Humans learn and plan from past episodes by connecting them to the current environment and the task at hand. In cognitive science, episodic memory refers to “a human memory subsystem that is concerned with storing and remembering specific sequences and occurrences of events pertaining to a person’s ongoing perceptions, experiences, decisions and actions” [1]. It helps a human plan the next task. In recent years, researchers have begun to realize the importance of episodic memory to artificial intelligence and cognitive robots, and the episodic like approaches to general event processing.

We propose a computational framework that utilizes the idea of episodic memory to cope with robot planning on complex tasks. Our approach is based on the traditional mathematical model of Markov decision processes, combining the episodic memory approach. In this way, it provides a human-like thinking for autonomous robots, so that they can accomplish complex tasks in pervasive assistive environments, and thus achieve the goal of assisting the everyday living of people. In regard to the traditional hierarchical algorithms for Markov decision processes, although they have been proved to be useful for the problem domains with multiple subtasks due to their strength in task decomposition, they are weak in *task abstraction*, something that is more important for task analysis and modeling. Using episodic task planning and learning, we propose a task-oriented design approach, which addresses the functionality of task abstraction. Our approach builds an episodic task model from different problem domains, which the robot uses to plan at every step, with more concise structure and much improved performance than the traditional hierarchical model. According to our analysis and experimental evaluation, our approach has shown to have better performance than the existing hierarchical algorithms, such as MAXQ [2] and HEXQ [3].

We further introduce a hierarchical multimodal framework for robot planning in multiple-sensor pervasive environments, using *multimodal POMDPs*. Considering realistic assistive applications may be time-critical and highly related with the risk of planning, we develop a *risk-aware* approach, allowing robots to possess risk attitudes [4] in their planning. Thus, we have answered the question of how to plan and make sequential decisions efficiently and effectively under complex tasks in pervasive assistive environments, which is very important for the design of applications to assist the living of the elderly and the disabled.

## TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . .	iii
ABSTRACT . . . . .	v
LIST OF FIGURES . . . . .	x
LIST OF TABLES . . . . .	xii
Chapter	Page
1. INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	2
1.2 Significance . . . . .	7
1.3 Outline . . . . .	8
2. PLANNING IN PARTIALLY OBSERVABLE DOMAINS . . . . .	9
2.1 Introduction of MDPs and POMDPs . . . . .	9
2.2 POMDP Solutions . . . . .	11
2.3 Belief Value Iteration with MDP Heuristic . . . . .	13
2.3.1 Belief States Approximation . . . . .	13
2.3.2 The MHVI Algorithm . . . . .	15
2.3.3 Belief Graph and Belief Space Compression . . . . .	17
2.4 Experimental Evaluation . . . . .	21
2.4.1 Performance Comparison . . . . .	21
2.4.2 Stage Convergence and Post-Convergence Iteration . . . . .	22
2.5 Conclusion . . . . .	25
3. EPISODIC TASK LEARNING IN ASSISTIVE ROBOTS . . . . .	27
3.1 Introduction to Episodic Tasks . . . . .	27

3.1.1	Episode and Episodic Memory . . . . .	28
3.1.2	Episodic Task Planning and Markov Decision Processes . . . . .	30
3.2	Hierarchical Approaches for MDPs . . . . .	32
3.3	Task-Oriented Design . . . . .	35
3.3.1	Tasks and MDPs . . . . .	35
3.3.2	Experience and Task Abstraction . . . . .	37
3.4	Episodic Task Learning . . . . .	38
3.4.1	Differentiate the States . . . . .	38
3.4.2	Episodic Task State Abstraction . . . . .	40
3.4.3	Knowledge Acquisition by TMDP . . . . .	43
3.5	Extended to POMDP Task . . . . .	44
3.6	An Alternative Approach for the Modeling . . . . .	48
3.7	Comparison of Episodic Task Learning and Episodic Memory . . . . .	50
3.8	Experimental Evaluation . . . . .	53
3.9	Conclusion . . . . .	57
4.	PLANNING FOR MULTIMODAL PERVASIVE APPLICATIONS . . . . .	58
4.1	Introduction . . . . .	58
4.2	Technical Preliminary . . . . .	60
4.3	Pervasive Human-Computer Interaction . . . . .	62
4.3.1	Sensor Coverage in Functional Areas . . . . .	62
4.4	Dynamic Fusion of Sensor Data . . . . .	63
4.4.1	Event System and Multi-observation . . . . .	65
4.5	Hierarchical Multimodal Markov Stochastic Domains . . . . .	67
4.5.1	Activity Planning by MDPs . . . . .	67
4.5.2	Action Planning by Multimodal POMDPs . . . . .	69
4.5.3	Action Control for Reminding Tasks . . . . .	72



4.6	Experimental Evaluation . . . . .	73
4.7	Related Work . . . . .	76
4.8	Conclusion . . . . .	77
5.	PLANNING FOR RISK-SENSITIVE TASKS . . . . .	78
5.1	Introduction . . . . .	78
5.2	Decision Theories and Utility Functions . . . . .	83
5.3	Risk-Aware MDPs . . . . .	85
5.4	Decision-Making with Cost and Wealth . . . . .	88
5.4.1	Decision Processes and Decision Graph . . . . .	88
5.4.2	Reward, Cost and Wealth . . . . .	88
5.5	Bridging Cost and Wealth . . . . .	91
5.5.1	Problem Formulation . . . . .	91
5.5.2	Equipotential Operations . . . . .	93
5.5.3	Bi-Directional Value Iteration . . . . .	96
5.6	Example . . . . .	99
5.7	Conclusion . . . . .	104
6.	CONCLUSION AND FUTURE WORK . . . . .	105
6.1	Summary . . . . .	105
6.2	Future Work . . . . .	106
6.3	Closing Remarks . . . . .	106
	REFERENCES . . . . .	108
	BIOGRAPHICAL STATEMENT . . . . .	117

## LIST OF FIGURES

Figure	Page
1.1 The Task for An Robot Serves Medicine to A Patient . . . . .	3
2.1 Example of A State Graph and A Belief Graph (a) State Graph; (b) Belief Graph . . . . .	18
2.2 Convergence of Reward in MHVI Algorithm: (a) Hallway2; (b) Tag . . . . .	22
2.3 MHVI Performance of Average Steps to Absorbing States (y-axis is the number of steps, i.e. the stochastic distance to absorbing states): (a) Hallway2; (b) Tag . . . . .	23
2.4 Effect of Belief Points Reassignment Level on the Performance: (a)Size of Belief Space $ B $ decreases with greater $\ell$ ; (b) The use of $\ell$ does not change the Reward . . . . .	24
3.1 An Example of Episodic Memory [5] . . . . .	30
3.2 MAXQ Framework for Taxi . . . . .	33
3.3 Taxi Domain . . . . .	39
3.4 Episodic Task Abstraction for Taxi . . . . .	41
3.5 Episodic Task Learning Approach Diagram . . . . .	42
3.6 Task Abstractions for HEXQ and MAXQ: (a) Taxi HEXQ; (b) Taxi MAXQ . . . . .	43
3.7 RockSample[4,4] Domain . . . . .	46
3.8 Task Abstraction for RockSample[4,4] (POMDP with abstract actions) . . . . .	47
3.9 TSN Graph for Taxi Task (MDP) . . . . .	49
3.10 TSN Graph for Coffee Task (POMDP without Abstract Actions) . . . . .	51
3.11 Episodic Memory Formation . . . . .	52

3.12	Comparison for Taxi Domain (Single Trial): (a) Reward of by Iteration Steps; (b) Reward of by Time . . . . .	54
3.13	Comparison for Taxi Domain (Different Trials): (a) Convergence Steps; (b) Convergence Time . . . . .	56
4.1	Sensor Deployment in A Pervasive Environment . . . . .	63
4.2	Sensor Events and Dialogue Events Become the Source of Multi-observation for Reminder Planning System . . . . .	65
4.3	A Scenario for Loose Coupling Pervasive Interaction (Remind the User to Take Medicine): dashed-lines indicate optional speech response . . . . .	66
4.4	Hierarchical Multimodal Framework for Reminder Planning System . . . . .	68
4.5	Activities for Reminding Tasks . . . . .	68
4.6	The Action Domain TSN Graph for Every Activity . . . . .	70
4.7	Learning Curve for the Multimodal POMDP . . . . .	74
4.8	Analysis of the Actions in Multimodal POMDP Subtasks . . . . .	74
4.9	Statistic for the Prompt Action in the Reminder System . . . . .	75
5.1	Relationship of Wealth, Risk Attitude with Utility . . . . .	79
5.2	Decision Tree for the Investment of a Venture . . . . .	81
5.3	Bi-directional Value Iteration for Bridging Cost and Wealth (equipotential positions of decision nodes are visualized as bold lines) . . . . .	97
5.4	Decision Graph for the Block-World Problem . . . . .	102
5.5	Comparing the Initial Plan ( $W_0$ ) and the Optimal Plan ( $W_0 = 20$ ): x-axis be $\tau$ , y-axis be the index of plan . . . . .	104

## LIST OF TABLES

Table		Page
2.1	Comparison of Different Algorithms (Items with * are tested in our platform) . . . . .	26
3.1	Model Parameters of RockSample[ $n, k$ ] ( $n^2$ map, $k$ rocks) . . . . .	47
3.2	Performance Comparison of POMDP Tasks . . . . .	57
4.1	Correlations for States and the Second Class Atomic Actions . . . . .	73
5.1	Comparison of Reward, Cost and Wealth in Planning . . . . .	91
5.2	Four Optional Plans for the Block-World Problem . . . . .	102
5.3	Example Data from the Bi-directional Value Iteration Process . . . . .	103

## CHAPTER 1

### INTRODUCTION

A pervasive assistive environment is a smart home space that provides healthcare monitoring and assists the elderly and the disabled people. The domestic equipment automation system, the pervasive computing and data mining based healthcare monitoring system, and the robot service system are three branches for the platform of assistive environments. Wireless sensors, cameras, PDAs and healthcare monitoring devices are used to detect user activity and health [6, 7]. With accurate information of the human, such as the living habits and daily activities, it is possible to analyze the physical and mental state of the human.

The elderly and the disabled are a special group of users that deserve special care. A simple and friendly human-machine interface is necessary for the assistive environment instruments. For the increasing number of elderly and disabled that live alone, it becomes a social problem to introduce a good solution to provide service and support for their activities of daily living (ADL) [8]. Sensor networks provide a mechanism to detect a person's activity. Using data-mining techniques, we are able to analyze the intention and health of a person. However, only the robot as an autonomous instrument is able to sense the world and change the world.

The concept of service robot has emerged over the past decades. The service robot has successfully found its way in society as factory robot, cleaning robot, coffee robot, rescue robot, etc. Planning, navigation, decision making and learning is the basic abilities for a robot to make good service for the user.

## 1.1 Motivation

How could a robot serve humans effectively and efficiently. This is a problem that researchers want to solve for many years, looking at different aspects of the problem. Generally speaking, a robot needs to organize its service to humans as tasks. A task is often referred to a specific process with goals or termination conditions. Tasks are highly related to situation assessment, decision making, planning and execution. For each task, we achieve its goals by a series of actions. A complex task contains not only different kinds of actions, but also various internal relationships, such as causality, hierarchy, etc.

An assistive robot in pervasive environments faces many types of tasks. Take a healthcare robot that serves medicine to a patient as an example (Figure 1.1). It has to get a cup of water and a bottle of medicine first, then it delivers the water and medicine to the patient. This is a rather simple task for a human. When it comes to a robot, we have to consider how to optimize its planning and execution. A more complex situation of the task is, if only one of the three cups has water, and only one of the two bottles has medicine, while the robot does not know exactly which cup has water and which bottle has medicine, then this task has to involve a complex process of searching. This may greatly increase the complexity of the problem domain. A similar domain with the medicine domain is RockSample [9], in which a robot explores in square region to determine the usability of several rocks located throughout the area. In these cases, the tasks have partially observable factors, that is, the medicine robot cannot determine whether the cup is full of water until it comes close to the cup, and the exploring robot has to come close to the rock until it has a probability 1 to make a successful check of the usability of the rock.

Markov decision processes (MDPs) serve as useful mathematical tools to model the assistive tasks in a pervasive environment. An MDP is a model that an agent interacts synchronously with an environment that is fully observable. It assumes the sensor system of the agent is precise enough such that it can observe every state with probability 1. Partially Ob-



Figure 1.1. The Task for An Robot Serves Medicine to A Patient.

servable Markov Decision Processes (POMDPs) are generalized forms of MDPs, in which the observations are considered to have some certain probabilities [10]. POMDPs are more flexible to deal with uncertainty in different tasks. In [11], POMDPs are proposed for the planning system to remind people with dementia or cognitive disabilities to wash hands. Some research works incorporate dynamic Bayesian networks to strengthen POMDP [12], which is modeled as factored POMDPs. It is proposed for the spoken dialogue management system served to order the ticket. However, early problems of (PO)MDPs have often been constrained in small state spaces and simple tasks. For example, Hallway is a task in which a robot tries to reach a target in a 15-grids apartment [10]. Considering the task structure, this process has only a single goal. The difficulties come from noisy observations by imprecise sensors equipped on the robot, instead of the task. Although (PO)MDPs have been accepted as successful mathematical approaches to model planning and controlling processes, without an efficient solution for big state spaces and complex tasks, we cannot apply these models on more general problems in the real world. In a simple task of grasping an object, the number of states reaches  $|S| = 1253$  [13]. If the task domain becomes

complex, it will be even harder to utilize these models. Suppose an agent aims to build a house, there will be thousands of tasks, with different configurations of states, actions and observations.

Compared to other task planning approaches, such as STRIPS or Hierarchical Task Network [14], (PO)MDPs consider the optimization for every step of the planning. Therefore, (PO)MDPs are more suitable for planning problems of intelligent agents. However, for task management, the (PO)MDP framework is not as powerful as the Hierarchical Task Network (HTN) planning [15]. HTN is designed to handle problems with many tasks. Primitive tasks can be executed directly, and non-primitive tasks will be decomposed into subtasks, until every subtask becomes a primitive task. This idea is adopted in the hierarchical partially observable Markov decision processes (HPOMDPs) [16]. Actions in HPOMDPs are arranged in a tree. A task will be decomposed into subtasks. Each subtask has an action set containing primitive actions and/or abstract actions. In fact, a hierarchical framework for (PO)MDPs is an approach that builds up a hierarchical structure to invoke the abstract action subfunctions. Although inherited the merits of task management from HTN, the hierarchical (PO)MDPs do not specially address the optimization of the big state space problem. Another solution considers multiple tasks as a merging problem using multiple simultaneous MDPs [17]. This solution does not specially consider the characteristic of different tasks, and it limits the problem domains to be MDPs.

Existing hierarchical approaches such as MAXQ [2] emphasize the decomposition of abstract actions, rather than discovering the internal relationship of subtasks. Compared to MAXQ, the VISA algorithm [18] is more close to an episodic relationship because it can depict the causal relationships between different state variables. Unfortunately, the causal relationships of VISA are correlated with state variables, rather than states, not to mention task-related states. While in the episodic task learning, we apply the relationships between abstract states, which are task-related states. Like the states defined in state space,



an abstract state is a reasonable combination of state variables. The difference between states and abstract states is, in our task-oriented approach, we extract the most useful task-related states as the abstract states. The author of HEXQ [3] clearly proposes a concept of skill-reuse. If a reinforcement learning agent can find and learn reusable subtasks and in turn employ them to learn higher level skills, then it should be more efficient than an agent without the skill-reuse. This idea is close to our proposition of task learning. Unfortunately, as examined in our experiments, the performance of HEXQ is not as good as other hierarchical approaches.

A cognitive robot is a robot that is human-aware [19], in order to improve the possibility to cohabitate with a human, and strengthen the human-robot interaction. Research into the development of a human-like thinking [20, 21] of a robot aims at enabling the robot to remember scenarios encountered, and use memory as experiences to improve the future planning.

When tasks to assist people are too complex for a robot to decide what to do next, the task planner wants to utilize a smarter framework for the sequential decision-making. An effective approach can be considered is the human-like thinking approach. In fact, artificial intelligence is motivated just using some mathematical, algorithm or other theoretical technologies to simulate the smart humankind. In this dissertation, we propose a new approach for the task planning, which we call *episodic task planning*, we also describe how to learn such a episodic task framework. The word of *episodic* comes from episodic memory from cognitive science [22]. In recent years, *episodic memory* for autonomous robots has become a hot research topic [5, 23]. There is also work on using intelligent agents that incorporate human-like thinking, so as to become cognitive robots [24, 21].

Our approach of episodic task planning and learning provides a computational framework and a solid mathematical foundation for the robot planning with human-like thinking. It is known as episodic memory, referring a functionally subsystem of human memory that

is concerned with storing and remembering specific sequences of events pertaining to a person’s ongoing perceptions, experiences, decisions and actions [1]. The human memory can be divided into short term memory (working memory) and long term memory. The short term memory stores information recently received or needed for further processing. The long term memory consists of three parts [25]:

- *Semantic Memory*: It is a network of associations and concepts that underlies our basic knowledge of the word meanings, categories, facts and propositions.
- *Procedural Memory*: It is memory for behavior or motor skills, or more generally, perceptuomotor or ideomotor skill. The skill such as riding a bicycle, swimming is a procedural memory.
- *Episodic Memory*: It involves the literal re-experiencing of past events-the bringing back to awareness of previous experiential episodes. A human accesses this memory if he can say “I remember”.

Episodic memory is concerned with unique, concrete, personal experience dated in the person’s past (e.g. a trip to a park). On the other hand, *semantic memory* [26] refers to a person’s abstract, timeless knowledge of the world (e.g. the color of the sky).

We propose a task-oriented design approach. Specifically, we use the concept of *task abstraction*, which is a snapshot of a scenario. When a planner comes to such a snapshot, it will arouse the planner to recall previous memory about a similar episode. Then the planner will be able to follow its stored experiences to execute the current situation. Our computation framework is based on Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs). We improve the original design of (PO)MDPs based approaches and incorporate the advantages of episodic memory, as a result, we call this approach *episodic task planning and learning*.

We further introduce a hierarchical multimodal framework for robot planning in a pervasive environment, using multimodal POMDPs, which models our approach to solve

real-world problems in pervasive environments. Considering that realistic applications may be risk-sensitive, we use a risk-aware approach for the task planning. As a result, we answer the question of planning and decision-making for complex tasks in pervasive assistive environments, which is very important for the design of assistive applications. The episodic task planning and learning forms the core part of our approach.

## 1.2 Significance

In this work, we answer a fundamental question about how to effectively plan the actions of a robot in complex pervasive environments. Our computational framework makes sequential decisions for the execution of tasks. This is significant in the following aspects:

Compared to the episodic memory approach, our approach provides an ability to optimize the system using global data, rather than simply to correlate an episode with an event. We use the classical models of MDPs and POMDPs that have been proved to be a rather flexible and powerful mathematical tools to optimize the action control of robots. It provides a strong computational framework for the execution of tasks and the completion of objectives.

On the other hand, considering the traditional hierarchical algorithms for MDPs, although they are strong in task decomposition, they are weak in task abstraction, which is more important for task analysis and modeling. In this work, we propose a task-oriented design to strengthen the task abstraction. Our approach learns an episodic task model from the problem domain, with which the planner obtains the same control effect, with concise structure and much improved performance than the original model. According to our analysis and experimental results, our approach has better performance than the existing hierarchical algorithms, such as MAXQ and HEXQ. More importantly, our approach pro-

vides a way to learn and store experiences, and make use of these experiences for future planning and execution.

We also provide solutions for multimodal and risk attitudes factors in planning systems. We give examples to explain how to solve these problems. As a result, the question of how to make planning in pervasive environments that have complex tasks is answered in this dissertation.

### 1.3 Outline

The remainder of this dissertation is organized as follows. Chapter 2 introduces the technical preliminaries of MDPs and POMDPs, and gives an algorithm for POMDPs. Chapter 3 shows how episodic task learning can be used by intelligent agents to effectively and efficiently make sequential decisions to serve people in assistive pervasive environments. Chapter 4 further introduces a hierarchical framework for multimodal POMDPs. The application background is a reminder system to serve people with cognitive impairments. Chapter 5 explains how to deal with risk-aware planning in pervasive environments. Chapter 6 discusses the results and gives ideas for future work.

## CHAPTER 2

### PLANNING IN PARTIALLY OBSERVABLE DOMAINS

#### 2.1 Introduction of MDPs and POMDPs

A Markov Decision Process (MDP) [27, 28] is a model to solve sequential decision making under uncertainty. An MDP defines the model an agent interacts synchronously in fully observable environments. The fully observable environment is an ideal assumption. It assumes the sensor system of the agent is precise enough such that it can observe every state with probability 1. The model is described as a tuple  $M_{mdp} = \langle S, A, T, R, \gamma \rangle$ , where  $S$  is a set of states,  $A$  is a set of actions,  $T(s, a, s')$  is the transition probability from state  $s$  to  $s'$  using action  $a$ ,  $R(s, a)$  defines the reward when executing action  $a$  in state  $s$ , and  $\gamma$  is the discount factor. The optimal state-action mapping for the  $t^{th}$  step, denoted as  $\pi_t$ , can be calculated by the optimal  $(t - 1)$ -step value function  $V_{t-1}$ :

$$\begin{aligned}\pi_t(s) &= \arg_a V_t(s) \\ &= \arg \max_a Q_t(s, a) \\ &= \arg \max_a \left[ R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{t-1}(s') \right].\end{aligned}\tag{2.1}$$

where  $V_t(s)$  is the value of a state  $s$  in the time step  $t$ , and  $Q_t(s, a)$  is the  $Q$ -value of a state  $s$ , given an action  $a$  in the time step  $t$ .

This is originally known as Bellman equation [29]. It is solved using dynamic programming, with convergence condition as  $\max_s |V_t(s) - V_{t-1}(s)| < \epsilon$ .

A POMDP models an agent's action in an uncertainty world. At each time step, the agent needs to make a decision based on collected historical information. A policy is defined as a function of action selection under stochastic state transitions and noisy obser-

vations. A POMDP is represented as  $M_{pomdp-S} = \langle S, A, O, T, \Omega, R, \gamma \rangle$ , where  $S$  is a finite set of states,  $A$  is a set of actions,  $O$  is a set of observations. At each time step, the agent lies in some state  $s \in S$ . After taking an action  $a \in A$ , the agent goes into a new state  $s'$ . The transition is a conditional probability function  $T(s, a, s') = p(s'|s, a)$ , which is the probability the agent lies in  $s'$ , after taking action  $a$  in state  $s$ . The agent makes observations to gather information, with observation result  $o \in O$ . This is modeled as a conditional probability,  $\Omega(s, a, o) = p(o|s, a)$ .

Now that the POMDP has to model the planning with partially observable information, it introduces a concept of belief state, representing how much belief an agent assigns to the possible states in which it resides. After belief state is taken into consideration, the original POMDP model changes to a seemingly fully observable MDP model, denoted as  $M_{pomdp-B} = \langle B, A, O, \tau, R, b_0, b_g, \gamma \rangle$ , where  $b_0$  is the initial belief state, and  $b_g$  is the absorbing states, representing the goals and/or termination states (RockSample [9] is a typical example using termination states. POMDP processes without any goal or termination state are theoretically feasible. If this condition is considered, we can simply set  $b_g = null$ ). The  $B$  is a set of belief states, i.e. belief space. The  $\tau(b, a, b') = p(b'|b, a)$  is the probability the agent changes from  $b$  to  $b'$  after taking action  $a$ . The  $R(b, a) = \sum_s R(s, a)b(s)$  is the reward for belief state  $b$ .

The POMDP framework is used as a control model for an agent. In a control problem, a utility is defined as a real-valued parameter to determine the action of an agent at every time step. This is represented as a real-valued reward  $R(s, a)$ , which is a function of state  $s$  and action  $a$ . The optimal action selection becomes a problem to find a sequence of actions  $a_{1..t}$  to maximize the expected sum of rewards  $E(\sum_t \gamma^t R(s_t, a_t))$ . When we use a discount factor  $\gamma$ , the values can converge to a fixed number. When states are not fully observable, the goal becomes to maximize the expected reward for each belief. The  $n^{th}$  horizon value

function can be built from the previous value  $V_{n-1}$  using a *backup* operator  $H$ , i.e.  $V = HV'$ .

The value function is formulated as the following Bellman equation

$$V(b) = \max_{a \in A} [R(b, a) + \gamma \sum_{b' \in B} \tau(b, a, b') V(b')]. \quad (2.2)$$

Here,  $b'$  is the next step belief state,

$$b'(s) = b_t(s') = \eta \Omega(s', a, o) \sum_{s \in S} T(s, a, s') b_{t-1}(s), \quad (2.3)$$

where  $\eta$  is a normalizing constant.

When optimized exactly, this value function is always piece-wise linear and convex in the belief space [30].

## 2.2 POMDP Solutions

The uncertainty property and the scale of problems affect the solution of POMDP. Various approaches have been proposed in prior works.

Value iteration is an approach originated from utility theory [31]. Some early algorithms, such as Witness [32], consider the use of exact value function for belief states. Unfortunately, the time and space complexity of an exact value iteration makes it only suitable to solve POMDPs with dozens of states. More researchers try to find approximate methods for value iteration. A smooth and differentiable approximation method called SPOVA is proposed in [33]. A Boyen-Koller approximation algorithm for belief states is proposed in [34]. It requires the model to be specified as a dynamic Bayesian network. In [35], belief states are approximated by sets of (weighted) samples drawn from particle filter. When a new belief state is encountered, its Q-value is obtained by finding  $k$  nearest neighbors from the historical records, with result being the linearly averaging of the Q-values.

The MDP heuristic is considered in  $Q_{MDP}$  [10] to solve POMDPs. The Q-value for belief state  $b$  is estimated at  $Q_a(b) = \sum_s b(s) Q_{MDP}(s, a)$ . Policies within  $Q_{MDP}$  do not take

actions to gain information, and decisions are made under current uncertain information. Therefore,  $Q_{MDP}$  can lead to loops of belief states [10]. Several methods are considered to improve the MDP heuristic approach. The most-likely-state approximation computes the most likely state  $x^* = \arg \max_x b(x)$ , and defines  $\pi(b) = \pi^{MDP}(x^*)$ . A transition entropy Q-MDP algorithm is proposed in [36]. From the experimental results of Hallway2 problem, the goal achievement percentage is 63.7%, compared to 3.9% for Q-MDP.

Reachable belief state based algorithms (such as PBVI [37], Randomized PBVI [38]) are considered as a substitute for each belief update. PBVI produces new beliefs in a single step forward trajectory for every action. It only keeps one belief state that is farthest away from any points already in the belief set  $B$ . The randomized PBVI initiates reachable belief states by randomly exploring the environment. Instead of backing up every  $b \in B$ , it backs up random belief states, instead of every  $b \in B$ . Classification and pattern adaptation are used in [39] to find a decomposition of belief space into a small number of belief features. The planning is taken over a low-dimensional space by discerning features and using standard value iteration to find policies over discrete beliefs. However, utilizing the classification technique on the belief spaces, the efficiency issue needs to be taken into consideration. Another is the cluster approach to aggregate states [40], according to the optimal MDP values. The soft clustered belief space is then projected onto the POMDP. A framework extended from Bayesian reinforcement learning on the POMDP problem domains is proposed in [41]. It relies on the Dirichlet distribution to approximate the infinite belief space to a finite space.

Several approaches based on AND-OR trees and AND-OR graphs are proposed in [42] to eliminate repeated states. In such a tree or a graph, OR-nodes represent action choices by the planning agent, and AND-nodes represent the arrival of percepts. The purpose is to provide a redundant structure to solve domains with huge belief states. In this paper, we propose a least visited belief point reassignment (LVBPR) technique, which is



proved to be useful to limit the increase the belief space, and without a complex and redundant structure.

### 2.3 Belief Value Iteration with MDP Heuristic

In this section, we introduce the MDP heuristic based value iteration algorithm for POMDPs.

#### 2.3.1 Belief States Approximation

If the environment is not fully observable to an agent, the first work is to find out a quantification representation of the environment. This is why we consider belief states. Belief states are continuous probabilities representing the current realization of an uncertain environment. A belief state is the realized state of the world from sufficient statistic of the history information. A POMDP model can be solved using  $M_{pomdp-B}$ . The belief states approximation becomes an estimation from an infinite continuous belief space to a finite discrete belief space, denoted as  $\tilde{B} = SE(B)$ , where  $SE$  is the belief state estimation function. The solution becomes  $M_{pomdp-\tilde{B}} = \langle \tilde{B}, A, O, V, \Pi, \tau, R, b_0, b_g, \gamma \rangle$ , where  $V(s)$  is the optimal value derived from  $M_{MDP}$ . Correspondingly, the value of belief state  $b$  is denoted as  $V(b)$ . The  $\Pi(b)$  is the optional set of optimal actions for belief state  $b$ , and  $\tilde{B}$  is a finite set of belief states, compared to the infinite set of belief states  $B$ .

In a point-based approach, a belief space is built using the reachable points of belief states [37]. A belief state  $b \in B$  is a probability distribution over discrete state space  $s_{1..n}$ ,  $\sum_{s=1}^n b(s) = 1$ . A belief point is a label to identify the belief state  $b$ , denoted as  $\tilde{b}$ . The identification of a belief state  $b \in B$  is a classification of belief points in the belief space. Our method has two stages. In the first stage, we use *binary discrimination* as a rough classification of the probability distributions. Let  $b_1, b_2$  be two belief states with the same state space. We can easily get two binary similarities:

1.  $b_1$  and  $b_2$  are  $\varepsilon$ -binary similar iff  $\forall s, b_1(s) > \varepsilon \wedge b_2(s) > \varepsilon$ , or  $b_1(s) \leq \varepsilon \wedge b_2(s) \leq \varepsilon$ ;
2.  $b_1(s)$  and  $b_2(s)$  are  $(1 - \varepsilon)$ -binary similar iff  $\forall s, b_1(s) > 1 - \varepsilon \wedge b_2(s) > 1 - \varepsilon$ , or  $b_1(s) \leq 1 - \varepsilon \wedge b_2(s) \leq 1 - \varepsilon$ .

If belief states  $b_1$  and  $b_2$  are both  $\varepsilon$ -binary similar and  $(1 - \varepsilon)$ -binary similar, then  $b_1$  and  $b_2$  are *binary similar*, denoted as  $b_1 \approx b_2$ . Using binary belief discrimination, the continuous infinite belief space  $B$  is approximated to a finite belief space  $\tilde{B}$ , with size  $|\tilde{B}| = 2^{|S|}$ . The binary belief discrimination is a rough but efficient classification for discrete states distribution. Its time complexity is  $O(|\tilde{B}||S|)$ .

Although binary discrimination is efficient when approximating the infinite belief space onto finite horizon (finite number of states), we have to consider a more precise method to match belief states with labeled belief points, which becomes the second stage. There are several mathematical models to represent the difference of distribution, such as K-L divergence. Considering the specific belief states distribution, we adopt the distribution *distance* and *affinity* theory [43]. Let  $b_1$  and  $b_2$  be two belief states defined on the same belief space  $B$ , the distance between  $b_1$  and  $b_2$  is

$$d(b_1, b_2) = \sqrt{2(1 - \rho(b_1, b_2))},$$

where  $\rho(b_1, b_2)$  is the affinity of  $b_1$  and  $b_2$ , representing the likeness of the distributions,

$$\rho(b_1, b_2) = \sqrt{\sum_{s \in S} b_1(s)b_2(s)}. \quad (2.4)$$

Matusita [43] proved  $\rho(b_1, b_2)$  has the following properties:

1.  $0 \leq \rho(b_1, b_2) \leq 1$ ;
2.  $\rho(b_1, b_2) = 1$  iff  $b_1 = b_2$ .

In Algorithm 1, we list the process for the estimation of belief state  $b'$ , which is the next belief state computed from  $b$ . If  $b'$  is an existing belief state in  $\tilde{B}$ , the algorithm returns its belief point label. For a new belief point, we will insert  $b'$  in  $\tilde{B}$ .

```

input:  $b, a, b'$ 
output:  $\tilde{b}'$ 
 $\tilde{B}' = \{x | \tau(b, a, x) > 0 \wedge b' \approx x\}$ 
if  $\tilde{B}' \neq \emptyset$  then
     $\tilde{b}' = \arg \max_{\tilde{x}} \rho(b, x \in \tilde{B}')$ 
else
     $\tilde{B}'' = \{x | x \in \tilde{B} - \tilde{B}' \wedge b \approx x\}$ 
     $\tilde{b}' = \arg \min_{\tilde{x}} d(b, x \in \tilde{B}'')$ 
end if
if  $\tilde{b}' = \emptyset$  then
     $\tilde{B} \leftarrow \tilde{B} + b', \tilde{b}' \leftarrow \text{belief\_point}(b')$ 
end if

```

**Algorithm 1:** Belief State Estimation

### 2.3.2 The MHVI Algorithm

Now we introduce an MDP heuristic based belief value iteration (MHVI) algorithm to model the POMDP problems. It is based on an alternative tuple  $M_{POMDP-\tilde{B}}$ . Solution for the POMDP problem becomes a process to solve the tuple. Besides the elements provided in Section 2.3.1, we introduce others in the following. The  $\Pi$  can be derived from the result of  $M_{MDP}$ . This serves as the heuristic to compute the optimal policies of POMDP. The value for belief point can be computed by iterating Equation (4.2). The approximate belief space  $\tilde{B}$  is initialized with  $|O| + 2$  belief points (one belief point for each observation, plus  $b_0$  and  $b_g$ ), denoted as  $\tilde{B}_{0..|O|+1}$ . Let the belief point created by observation  $o$  be  $\tilde{B}(o, s)$ ,  $\tilde{B}(o, s) = \eta \Omega(s, \forall a \in A, o)$ . Unlike other belief points, during the entire algorithm, the initial set of belief points  $\tilde{B}_{0..|O|+1}$  keep fixed without change.

We use a learning process to build  $\tau$ . On initialization,  $\tau$  has a single entry,  $\tau(b_g, \forall a \in A, b_0) = 1$ . More transitional relationships can be learned during the explore-exploit process of value iteration.

```

repeat
   $b_{t+1} \leftarrow b'_t, t \leftarrow t + 1$ 
  for  $\forall a \in A$  do
     $\tilde{b}' = \text{BeliefStateEstimation}(b, a, b')$ 
    update  $\tau(\tilde{b}, a, \tilde{b}')$ 
     $V(\tilde{b}') = b' \times V(S)$ 
    if  $a \in \Pi(b)$  then
       $V_a(\tilde{b}) = \max_{a \in A} [R(b, a) + \gamma \sum_{b' \in B} \tau(\tilde{b}, a, \tilde{b}') V(\tilde{b}')]$ 
    end if
  end for
   $\pi = \arg \max_a V_a(\tilde{b})$ 
   $b'_t \leftarrow \text{BeliefStateEstimation}(b, \pi, b')$ 
until converge

```

**Algorithm 2:** MHVI Explore-Exploit Iteration

As is shown in Algorithm 2, MHVI assumes the optimal policy  $\pi$  to be an element of the optimal policy set for belief point  $\Pi(\tilde{b})$ , which is obtained from the tuple  $M_{mdp}$ . We do not constrain the exploit process using the policy set. Therefore, unknown belief points can still be found, which will benefit the final solution. Thus, the solution can be more failure-proof. Since  $|\Pi(b)| \leq |A|$ , this will also be helpful to decrease the exploration and search time.

A transition probability of two belief points indicates the probability when we take an action  $a$  on belief point  $b$ , and the system transit to another belief point  $b'$ . We have

$$\begin{aligned}\tau(\tilde{b}, a, \tilde{b}') &= \sum_{o \in O} [\Omega(b, a, o) T(b, a, o, b')] \\ &= \Omega(b, a, o),\end{aligned}$$

where  $T(b, a, o, b') = 1$  if  $SE(b, a, o) = \tilde{b}'$ ,  $T(b, a, o, b') = 0$  otherwise.

We find out  $\Omega(b, a, o)$  by the following process. First we compute  $b'$ ,  $b'(s) = b(s) = \eta \sum_{s \in S} b(s) T(s, a, s')$ , where  $\eta$  is a normalizing factor. By  $\Omega(b, a, o, s) = \eta b'(s) B(o, s)$ , we can finally obtain the transition probability for  $\tau(\tilde{b}, a, \tilde{b}')$ .

### 2.3.3 Belief Graph and Belief Space Compression

The transitional relationship for fully observable problems is often represented as a *state graph*. In a state graph, a *node* denotes the state of a system. States in a state space are bijective to nodes in its state graph. The initial state  $s_0$  is the beginning node of a trajectory. It can be revisited during the traversal. A state graph may have multiple absorbing states. An *edge* in a state graph represents an action  $a \in A$ . The transitional relationship is described as a probability  $T(s, a, s')$ . A state graph may have *loop*, which starts from state  $s$  and ends also in  $s$ . A state graph may contain *cycle*, which passes through state  $s$ , goes into another state, and returns to  $s$ . The absorbing states cannot be in a cycle. In a state graph, reward from action  $a$  is represented as the weight of an edge, i.e.  $weight(s, s') = R(s, a, s')$ .

The optimal policies of a state graph can form a path  $P_s^*$ , starting from  $s_0$  and going into  $s_g$ . *There is no loop or cycle in the optimal path  $P_s^*$  of a state graph.* In a specific application, the action may be uncertain, for example, a traveler has 0.8 probability to reach a new state and 0.2 probability to stay in the current position at a given step. This may result in a loop in the real trajectory, but not a loop in the optimal path  $P_s^*$ .

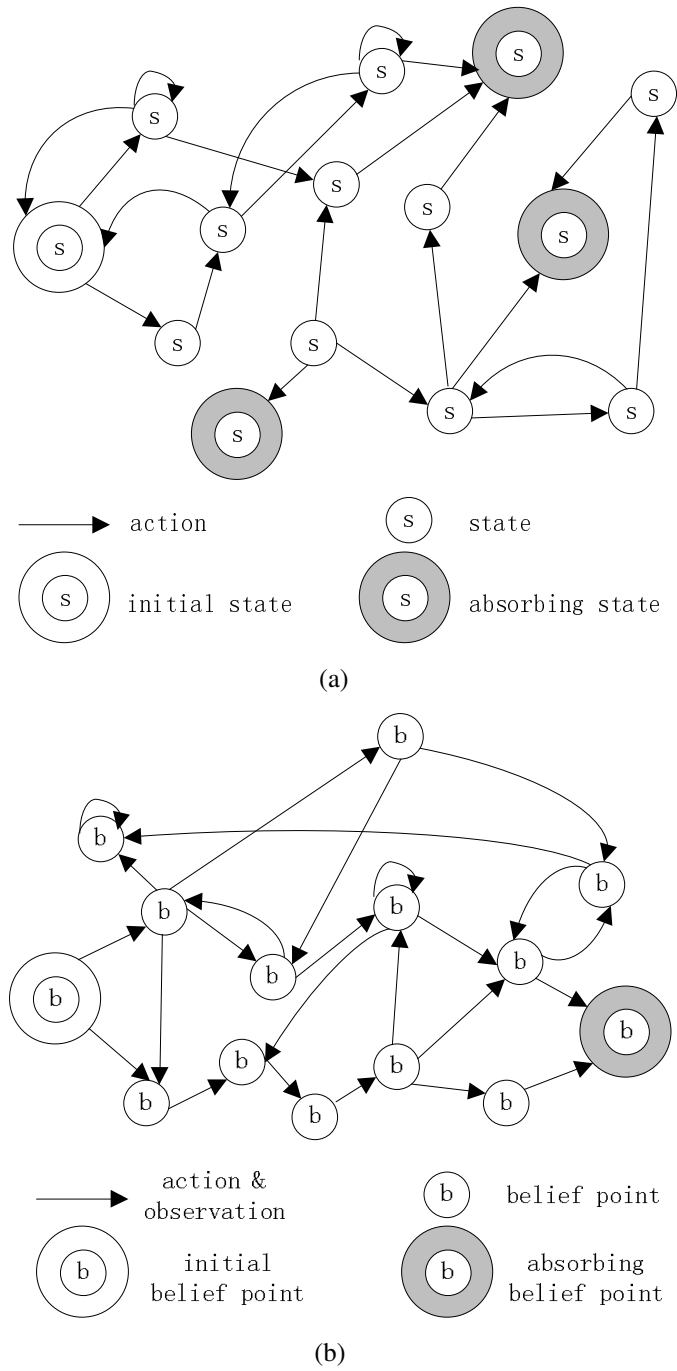


Figure 2.1. Example of A State Graph and A Belief Graph: (a) State Graph; (b) Belief Graph.

In the case of partially observable problems, we use a belief graph to depict a transitional relationship. In a belief graph, a node is a belief point  $\tilde{b}$ , which labels a belief state  $b$ . By approximating the belief space, the cardinality  $|B|$  changes from infinite to a finite value  $|\tilde{B}|$ . Therefore, belief states from the real belief space are surjective to nodes in the belief graph. A belief graph has an *initial belief point*  $b_0$  and an *absorbing belief point*  $b_g$ . An edge in a belief graph represents the action and observation pair  $(a, o)$ . A belief graph can contain a loop, which starts from a belief point  $b$  and ends also at  $b$ . A belief graph may contain a *cycle*, which passes through belief point  $b$ , goes in some other belief points, and returns to  $b$ . In a belief graph, the weight for an edge is the reward from the action and observation pair  $(a, o)$ , i.e.  $weight(\tilde{b}, \tilde{b}') = R(b, a, o, b')$ .

In this paper, every benchmark POMDP problem that has been researched by previous researchers makes the assumption that, states in POMDP are only partially observable, except for the absorbing states. This makes the absorbing belief states identifiable.

The optimal policies of a belief graph form a path  $P_b^*$ , starting from  $b_0$  and going into  $b_g$ . Suppose a belief graph is built from the exact belief space, i.e. every node represents a different belief state, there will be no cycle or loop in  $P_b^*$ . When we use the approximate belief space  $\tilde{B}$ , multiple belief states may be projected onto one belief point by surjection, therefore  $P_b^*$  may contain loop or even cycle. This is different from the optimal path of a state graph. We have two *constraint conditions* in the belief graph:

1. (Loop)  $\forall \tilde{b}, \tilde{b}' \in P_b^*, \tilde{b} = \tilde{b}' \Rightarrow b \neq b'$ ;
2. (Cycle)  $\tilde{b}_1, \tilde{b}_2, \dots, \tilde{b}_k, \tilde{b}'_1 \in P_b^*, \tilde{b}_1 = \tilde{b}'_1 \Rightarrow b \neq b'_1$ .

These constraint conditions will eliminate loops and cycles in a belief graph, so that we obtain a more reliable graph to establish the relationships between belief points.

Belief space is an important factor influencing the performance of the POMDP algorithm. The computing cost for one iteration can be  $O(|A||S|^2 + |A||\tilde{B}||S|)$ . We cannot change  $|S|$  and  $|A|$ . For an POMDP problem with big state space, the compression of belief

space is an effective strategy to maintain good performance. Other algorithms also have different types of constraints in order to restrict the size of belief space. For example, the Point Based Value Iteration (PBVI) algorithm [37] appoints only one belief point in each time step. Next, we introduce a *least visited belief point reassignment* (LVBPR) approach, by which we manage and compress the belief space in a natural way.

Every time a belief point is visited, it will be recorded into a counter. Let  $\ell$  be the *belief point reassignment level*. The system uses a buffer to store the belief points. At every time step, the system has a probability to release the belief points whose counter is lower than  $\ell$ . The released belief points can be assigned to the new belief state. Details of the LVBPR approach are as follows:

Let  $\varphi(\tilde{B})$  be the buffer of the belief space, with a lower-bound  $\lfloor \varphi(\tilde{B}) \rfloor$ , size  $|\varphi(\tilde{B})|$ , and an upper-bound  $\lceil \varphi(\tilde{B}) \rceil$ . In the beginning, let  $\lfloor \varphi(\tilde{B}) \rfloor = \tilde{B}_{|O|+2}$ , and the upper-bound of the belief space be  $\lceil \varphi(\tilde{B}) \rceil = \tilde{B}_{|O|+2+|\varphi(\tilde{B})|}$ .

In every time step, we verify the visited count of  $\lfloor \varphi(\tilde{B}) \rfloor$  with  $\ell$ , if it is higher than  $\ell$ ,  $\lfloor \varphi(\tilde{B}) \rfloor \leftarrow \lfloor \varphi(\tilde{B}) \rfloor + 1$ . We get the available belief points from buffer  $\varphi(\tilde{B})$ . The current location is set to an entry between  $\lfloor \varphi(\tilde{B}) \rfloor$  and  $\lceil \varphi(\tilde{B}) \rceil$ , denoted as  $\varphi(\tilde{B}_x)$ . If the visited count of a current entry  $\phi = \varphi(\tilde{B}_x)$  is lower than  $\ell$ , we release  $\tau(\phi, \forall a \in A, \forall \tilde{B}_i \in \tilde{B})$  and  $\tau(\forall \tilde{B}_i \in \tilde{B}, \forall a \in A, \phi)$ , with  $\phi$  assigned to the new belief state. Otherwise, we search the available belief point in next buffer entry. If  $\varphi(\tilde{B}_x) \leq \lceil \varphi(\tilde{B}) \rceil$ , then

$$\varphi(\tilde{B}_x) = \begin{cases} \varphi(\tilde{B}_x) + 1 \sim 1 - p \\ \lfloor \varphi(\tilde{B}) \rfloor \sim p \end{cases}$$

where  $p$  is the probability to return to  $\lfloor \varphi(\tilde{B}) \rfloor$ . This means that, in each step, the current pointer has a  $p$  probability to change to  $\lfloor \varphi(\tilde{B}) \rfloor$ .

If  $\varphi(\tilde{B}_x) > \lceil \varphi(\tilde{B}) \rceil$ , then  $\lceil \varphi(\tilde{B}) \rceil \leftarrow \lceil \varphi(\tilde{B}) \rceil + 1$ , the entry of  $\varphi(\tilde{B}_x)$  is assigned to the current belief state, and  $\varphi(\tilde{B}_x) \leftarrow \lfloor \varphi(\tilde{B}) \rfloor$ . In this case, we reassign the belief point



when the buffer is overflow. We make use of the parameters  $\ell$ ,  $\varphi(\tilde{\mathcal{B}})$  and  $p$  to adjust the performance. Therefore, LVBPR is a dynamic and configurable algorithm.

By single step simulation of next belief point from  $b$ , we get at most  $|A|$  belief points. By belief states estimation process, new belief points will be inserted into  $\tilde{\mathcal{B}}$ .

## 2.4 Experimental Evaluation

To evaluate the performance of MHVI, the benchmark problems of Tiger-grid, Hallway [10], Tag [37] and RockSample [9] are chosen in the simulation experiments. The experiments are implemented on Intel 2.4GHz CPU 2GB memory by Matlab.

### 2.4.1 Performance Comparison

The first experiment aims to get comparable results with existing algorithms. Replicate earlier experimental settings, each problem is executed 100 times, terminates after convergence within 251 steps. The results are averaged over 100 runs. Table 3.2 provides the comparison results based on goal completion rate, sum of rewards, policy computation time and number of belief points. We test Persus[44] and MHVI in the same environment. The comparison results indicate MHVI achieves competitive performance.

Most of the rewards in MHVI are higher than the algorithms seen in prior works, except for the Tag. However, even for the Tag domain, we found that if we use a different configuration in our algorithm, the reward of MHVI can even reach  $-5.83$ . For almost every problem domain, the belief space of MHVI is lower than that in other algorithms. This puts the MHVI algorithm to a light-weight search space, so as to achieve better performance. This is because we utilize a *least visited belief point reassignment* (LVBPR) mechanism in the MHVI, so that it can manage the belief space, and speed up the search process.

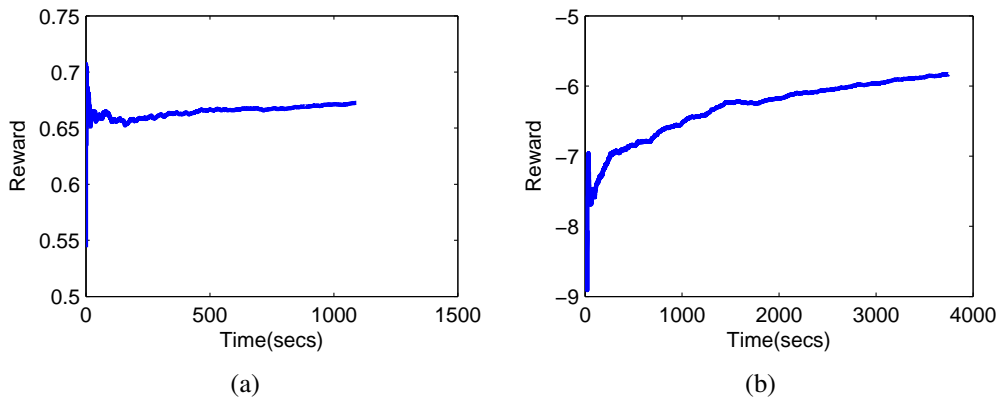


Figure 2.2. Convergence of Reward in MHVI Algorithm: (a) Hallway2; (b) Tag.

We keep same parameter configuration for different problem domains in experiment 1 (By the adjustable configuration parameters, in fact, we can optimize the results of reward and  $|\tilde{B}|$ ). However, rewards and  $|\tilde{B}|$  can be affected by the configuration. For example, in Tag-domain problem (Table 3.2), the reward of MHVI is  $-7.37$  and Persus  $-6.17$ . In the following experiments, by properly tuning the configuration of  $\ell$ , the reward of MHVI can even reach  $-5.83$  (Figure 2.2(b)). This kind of tuning will also affect the size of  $\tilde{B}$ . More details and explanations about the approach are presented in the next section.

#### 2.4.2 Stage Convergence and Post-Convergence Iteration

Since we use a discount factor  $\gamma$  in Bellman equation, and the state space of MDP is fixed, the value for each state is deterministic to converge after a sufficient number of iteration steps. The iteration process for MHVI serves as a learning process to build the POMDP model. The LVBPR approach guarantees the increase of  $\tilde{B}$  in a slow process.

Using the discount factor  $\gamma$ , the value function of a POMDP problem will always converge [32]. Given an approximation belief point set  $\tilde{B}$ , since the belief point of absorbing states  $b_g \subset \tilde{B}$ , if the values  $V(\tilde{B}_{0..|\tilde{B}|})$  converge, then the absorbing states are sure to be visited. Hence the goal can be reached when values converge.

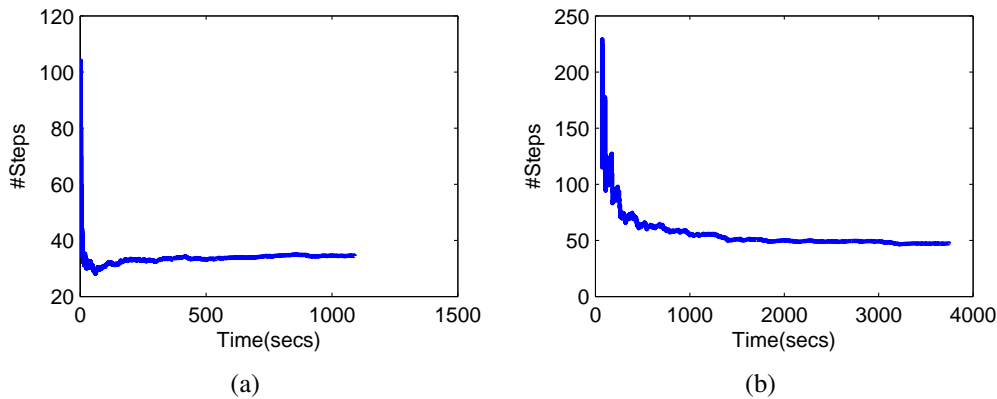
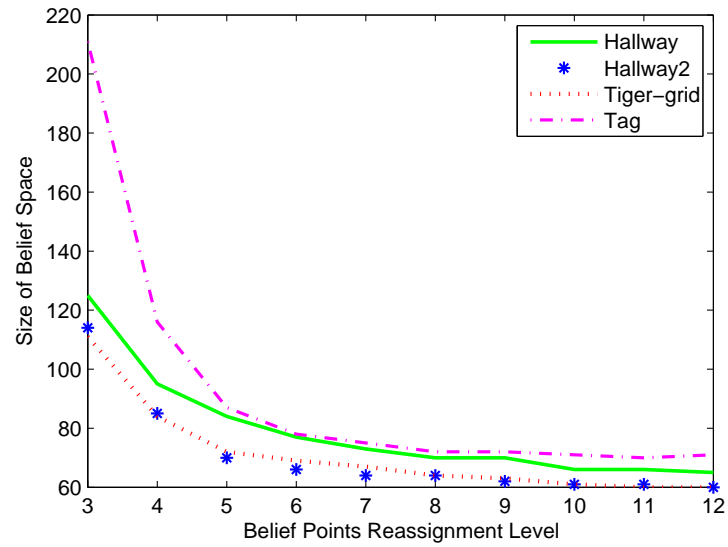


Figure 2.3. MHVI Performance of Average Steps to Absorbing States (y-axis is the number of steps, i.e. the stochastic distance to absorbing states):(a) Hallway2; (b) Tag.

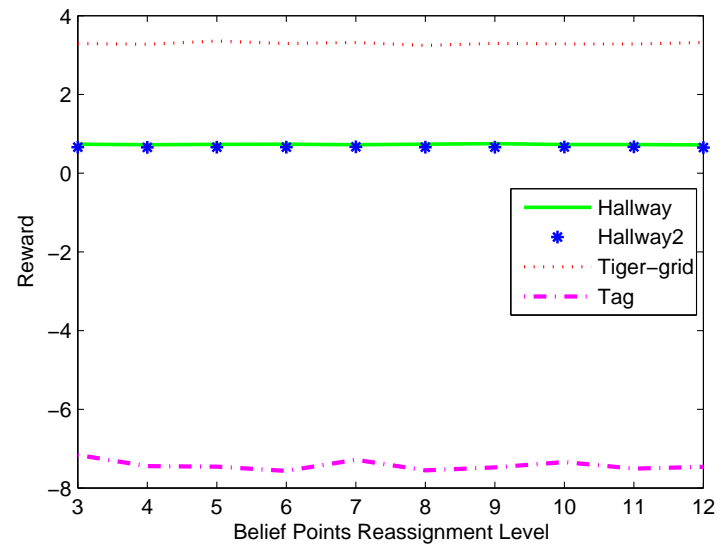
A belief space is a continuous space containing infinite belief points. For a finite approximate belief space  $\tilde{B}$  with converging values  $V(\tilde{B}_{1..|\tilde{B}|})$ , when a new belief point is discovered,  $\tilde{B}$  changes to  $\tilde{B}'$ , and values converge to  $V(\tilde{B}_{1..|\tilde{B}'|})$ . This indicates, unlike MDP, a POMDP model can converge in multiple stages, which we call *stage convergence*. We refer *post-convergence iteration* to the iteration after a convergence stage. The result will lead to a new stage of convergence.

In MHVI, there are multiple stages of convergence. The second experiment clearly demonstrates this effect. From Table 3.2, MHVI converges within no more than 251 steps for every benchmark problem. In experiment 2, each POMDP problem is executed 10000 time steps and then stops after the values converge. The results indicate they often stop within no more than 50 extra steps after the 10000 steps.

Figure 2.2 depicts the average of the discounted reward  $E(\sum_t \gamma^t R(s_t, a_t))$ . The average discount reward for Tag problem increases to  $-5.83$  at the 10000 time step. From the Tag problem (Figure 2.3(b)), the average number of steps to absorbing states decrease steadily with the increase of iteration steps. This indicates, during the post-convergence iteration, the POMDP finds better paths to the goal. The value iteration serves as a learning



(a)



(b)

Figure 2.4. Effect of Belief Points Reassignment Level  $\ell$  on the Performance: (a) Size of Belief Space  $|B|$  decreases with greater  $\ell$ ; (b) The use of  $\ell$  does not change the Reward.

process for the Tag POMDP model. However, the increase of reward for Hallway2 (Figure 2.3(a)) is not obvious.

In experiment 3, we want to make clear the effect of belief points reassignment level  $\ell$  on the performance. The system takes 500 time steps and stops after it reaches convergence. For each  $\ell$ , we execute the system 10 times and average the results. Results of the test problems Hallway, Hallway2, Tiger-grid and Tag are considered for the comparison. When  $\ell$  takes 2, there is no belief points reassignment, the belief points can be over 2000. In Figure 2.4(a),  $|\tilde{B}|$  is high for each problem, and it is lower than 100 for all the problems. Comparing with the results in Table 3.2, the time step is lower than 251,  $|\tilde{B}| = 107$ , and  $\ell = 3$ . In Figure 2.4(b), the reward decreases only a little when  $\ell$  increases. Thus, using  $\ell$ ,  $|\tilde{B}|$  becomes an adjustable parameter.

## 2.5 Conclusion

We present an algorithm of MHVI to solve POMDP problems. MHVI adopts an MDP heuristic approach to build a weighted graph, in order to model the approximate belief space, as well as a dynamic, configurable mechanism to manage graph nodes. The weighted graph is a unified model for POMDP problems, with or without goals and/or termination states. Theoretical analysis and experimental results indicate the weighted graph based heuristic algorithm is fast, flexible and robust in solving POMDPs.

Table 2.1. Comparison of Different Algorithms (Items with \* are tested in our platform)

<b>Method</b>	<b>Goal%</b>	<b>Reward</b>	<b>Time(s)</b>	<b> B </b>
<b>Tiger-grid</b> (36s 5a 17o)				
QMDP[37]	n.a.	0.198	0.19	n.a.
PBVI[37]	n.a.	2.25	3448	470
Persus*[44]	n.a.	2.34	61.6	93
HSVI2[45]	n.a.	2.30	52	1003
MHVI*	n.a.	3.21	1.67	61
<b>Hallway</b> (60s 5a 21o)				
QMDP[10]	47.4	0.261	0.51	n.a.
PBVI[37]	96	0.53	288	86
Persus*[44]	n.v.	0.51	61.4	105
HSVI2[45]	n.v.	0.52	2.4	147
MHVI*	100	0.71	2.80	72
<b>Hallway2</b> (92s 5a 17o)				
QMDP[10]	25.9	0.109	1.44	n.a.
PBVI[37]	98	0.34	360	95
Persus*[44]	n.v.	0.35	64.7	124
HSVI2[45]	n.v.	0.35	1.5	114
MHVI*	100	0.65	3.96	66
<b>Tag</b> (870s 5a 30o)				
PBVI[37]	59	-9.18	180880	1334
Persus*[44]	n.v.	-6.17	1542	418
HSVI2[45]	n.v.	-6.36	24	415
MHVI*	100	-7.37	9.24	104
<b>RockSample</b> [4, 4] (257s 9a 2o)				
HSVI1[9]	n.a.	18.0	577	458
HSVI2[45]	n.a.	18.0	0.75	177
MHVI*	n.a.	18.4	6.64	74
<b>RockSample</b> [5, 5] (801s 10a 2o)				
HSVI[9]	n.a.	19.0	10208	699
MHVI*	n.a.	20.4	13.56	83
<b>RockSample</b> [5, 7] (3201s 12a 2o)				
HSVI[9]	n.a.	23.1	10263	287
MHVI*	n.a.	23.0	225	98
<b>RockSample</b> [7, 8] (12545s 13a 2o)				
HSVI1[9]	n.a.	15.1	10266	94
HSVI2[45]	n.a.	20.6	1003	2491
MHVI*	n.a.	21.6	1959	140

n.a.=not applicable, n.v.=not available

## CHAPTER 3

### EPISODIC TASK LEARNING IN ASSISTIVE ROBOTS

#### 3.1 Introduction to Episodic Tasks

Real-world autonomous robot applications often contain complex tasks. For example, if a user wants a robot to bring a cup of coffee, the robot has to navigate first to the cup, grasp it, then navigate to the user, and finally put down the coffee for the user. This is a simple example of a coffee task. In the real-world, it is possible that the robot cannot find the cup, or the cup is empty without coffee, in extreme condition, the user may disappear when the robot wants to deliver him the coffee. In fact, the robot may be required to warm up the water and make the coffee, or even buy the coffee. These real-world examples become a new challenge to the state of the art of assistive robots: The decision of the robot has to be real-time, in order to immediately respond to any changes in the environment, and the robot needs to be able to make decisions on how to handle complex tasks that have multiple subtasks.

It is especially important for socially assistive robot [46] living in a home, to safely and reliably serve the elderly and the disabled. The robot has to take into consideration the state of the world, and finish its goal, which is defined as a task. Since robots must know the state of the humans in their control loop [19, 47], this greatly increases the complexity of the computing.

Early work about robot tasks is to model it as a task-sequencing planning problem, in order to determine the optimal sequence of paths for each robot such that all the tasks are executed with a total minimum cost (e.g. time) [48]. Such problems are solved according a proposed decomposition scheme. The ways in which agent actions affect the world can

be modeled compactly using a set of relational probabilistic planning rules. An algorithm called probabilistic relational dynamic [49] is proposed to learn a prior distribution over task-specific rule sets. It uses a hierarchical Bayesian approach. A recent work shows how to use MDPs to assist the memory-impaired elders for their activities of daily living (ADLs) using dynamic multi-tasks planning, and the system is designed as a reminder system with dialogue management [50].

In this work, we propose a novel approach for the solving of complex tasks, which we call episodic task learning. The idea is originated in cognitive science. It addresses the way humans analyze and handle their everyday activities. In the next section, we will first introduce the terms of “episode” and “episodic memory”.

### 3.1.1 Episode and Episodic Memory

From cognitive science, an *episode* is a basic unit of information that human memory operates on [5]. The decision as to what constitutes a meaningful episode is domain dependent and left to the external applications to make. In general, an episode is a sequence of actions with a specific goal.

In recent years, there has been renewed interest in AI applications to enhance intelligence agents with a memory of their past functioning. The inspiration comes from human *episodic memory*, “a functionally distinct subsystem of human memory that is concerned with storing and remembering specific sequences of events pertaining to a person’s ongoing perceptions, experiences, decisions and actions” [1]. Episodic memory is concerned with “unique, concrete, personal experiences dated in the remember’s past” (e.g. a trip to a park). On the other hand, *semantic memory* [26] refers to “a person’s abstract, timeless knowledge of the world” (e.g. the color of the sky).

“Both episodic and semantic memories are thought to be propositional in nature - they can be contemplated introspectively, can be communicated to others in some symbolic



form and questions about their veridicality can be asked.” These characteristics contrast with those of *procedural memory* [51], a memory subsystem concerned with the acquisition and utilization of procedures and skills, which is non-propositional.

A memory system is widely believed to have three high-level activities: encoding, storage and retrieval [1]. “*Encoding* is the process that converts a perceived event into a memory representation. It consists of *activation* (the process of determining when a new episode needs to be recorded), *salient feature selection* (deciding what information will be stored), and *cue selection* (deciding what features of the state will cue the memory). From a computational point of view, another aspect of encoding is the particular representation of episodes and their organization in memory.” “*Storage* deals with how stored episodes are maintained over time.” “*Retrieval* is the process by which encoded episodes become available again to the agent. Retrieval is triggered by the agent’s state and is based on *cues*, especially salient or significant parts of the retrieval information. *Cue construction* is the process of constructing the data used to retrieve a memory. *Matching* uses these cues in order to search for similar episodic memories. *Recall* means retrieving the memory from storage and placing it into working memory. After the process completes, the memory becomes available for the agent to use.”

An example about the episodic memory is provided in Figure 3.1 [5]. When the character in the figure sees an object, the episodic memory system is triggered, and a past situation where this object appeared is recovered as an episode. The character is now able to use this information to decide what to do in the current situation.

Recently, research about how to apply the episodic memory approach for autonomous agents has become a new direction in AI research. In path-planning [5], information regarding obstacles, food and other creatures perceived by the visual system are recorded within an episode. Instead of storing this information in a “world map”, they maintain episodes within the episodic memory. During the planning process, episodes are recol-

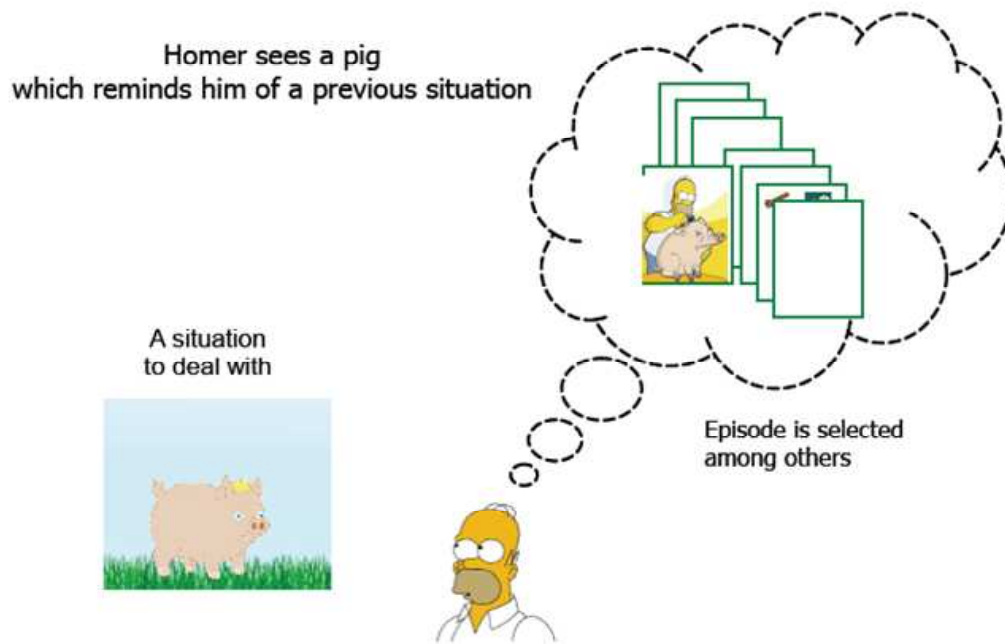


Figure 3.1. An Example of Episodic Memory [5].

lected in the working memory, and only “remembered” things are considered during the decision-making process. A similar approach using episodic memory for the planning is the obstacle avoidance in a Bubble Family game [52]. It is also reported that episodic memory system can be interfaced with a machine emotion system with the goal of producing intelligent behaviors in a cognitive humanoid robot [53].

### 3.1.2 Episodic Task Planning and Markov Decision Processes

In this work, we adopt the episodic approach for robot planning. We focus on a computational framework using episodic memory and show how to use the mathematical models of MDPs and POMDPs to solve episodic task planning. Similar to existing episodic memory approach, the salient features, or episodic snapshots [20] are kept in our approach of episodic task planning. We model these parameters into *abstract states*. A global map of the entire state space is held, for computing of optimal solution. After we have computed the optimal policies, we will obtain a so-called *experience*. We store the experiences as

*abstract actions*, which are highly related with the abstract states. These experiences are the episodes of that occurred before. During the planning, the planner will recall the episodes and execute them as abstract actions, according to the abstract states that the planner has discerned from the observations. Thus, an experience will become an episode plus the optimal policies in this episode. It is modeled as a temporally-extended action, i.e. an abstract action in the framework. To obtain a good experience is important to a planner. What we need to consider is to maximize the possibility of reuse. In other words, we need to extract the minimized experience. Otherwise, if an experience is still separable into multiple experiences, then we need to continue the task abstraction of the original domain, until every experience is minimized.

In episodic task planning, a task is an objective process, with a goal that the agent wants to achieve. An episode corresponds to an experience. It is related with an abstract state, which is appointed considering the context of the task. More importantly, the task needs to be abstracted into a sequence of abstract states, with connections to guide them to the final goal of the task.

Now, the challenge is finding the optimal way to create an optimal abstraction of a task, so that the entire story can be organized into episodes. More importantly, how to construct experiences, so that we can optimize the planning and execution of an assistive robot. In order to answer these questions, we will first review some existing approaches that use the MDPs for the planning of complex tasks in the next section. Then we introduce our idea of *task-oriented design* in Section 3.3, by which we will make clear the relationships further among tasks, MDPs and experiences. In Section 3.4, we introduce the approach for episodic task state abstraction, and the algorithm to learn and construct experience. Such an approach can also be extended to POMDPs, which we will introduce in Section 3.5. In Section 3.6, we present an alternative approach for the modeling of problem domains. In Section 3.7, we further make a comparison of the techniques between episodic task learning

and episodic memory. We provide a performance evaluation of the episodic task planning approach in Section 4.6. Finally we conclude this chapter.

### 3.2 Hierarchical Approaches for MDPs

The hierarchical approaches for Markov decision processes (MDPs) seem to be most suitable to solve the complex tasks. For example, MAXQ [2] is a classical approach that exploits a hierarchical decomposition to accelerate the computation of optimal policies. It has been proved to be better than the flat MDP approach that does not incorporate any hierarchy. The algorithm needs the designer to construct a MAXQ graph for every domain (Figure 3.2). The MAXQ graph contains two types of nodes: Max nodes represent the task and its subtasks, and Q nodes represent an action that can be performed to achieve its parent's subtasks. The basic idea of MAXQ is an exhaustive decomposition from the original task to the subtasks, until every subtask reaches an atomic action. The exhaustive decomposition is a common approach for task analysis. Hierarchical Task Network (HTN) planning [15] and hierarchical partially observable Markov decision processes (HPOMDPs) [16] also incorporate such an idea of exhaustive decomposition. In these approaches, the subtasks may be named as “subtasks”, “abstract actions”, or even “options” [54]. Although there is slightly different for their definitions, each of them is used to represent the sub-level tasks in the hierarchical structure. Although the parent-child relationship between a task and its subtasks can be discovered in the exhaustive decomposition based approach, it provides no further information about the other relationships between different subtasks.

Compared to MAXQ, the VISA algorithm [18] is more close to the episodic relationship because it can depict the causal relationships between different state variables. Unfortunately, the causal relationships of VISA are correlated with state variables, rather than states, not to mention task-related states. A state variable is simply a property of a

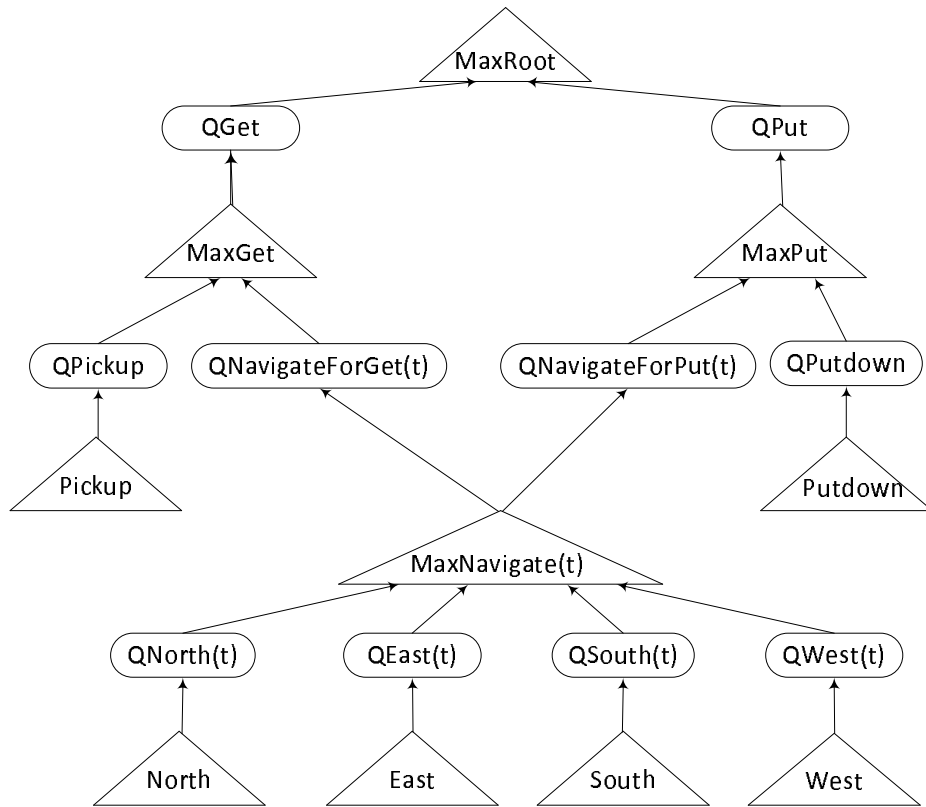


Figure 3.2. MAXQ Framework for Taxi.

state, but only the states are related directly with a task and its subtasks. Thereafter, the state variable influence graph used by VISA cannot depict directly the relationships among multiple tasks/subtasks.

The author of HEXQ [3] clearly proposes a concept of skill-reuse. If a reinforcement learning agent can find and learn reusable subtasks and in turn employ them to learn higher level skills, then it should be more efficient than an agent without the skill-reuse. This idea is close to our proposition of task learning. Unfortunately, as examined in our experiments, the performance of HEXQ is not as good as other hierarchical approaches. This conclusion is coherent with the result from [55]. The authors of [55] also claimed that there exists a concurrent HEXQ whose performance can be better than an un-concurrently designed MAXQ. Generally, an un-concurrently designed algorithm can be implemented

as a concurrent algorithm with better performance. However, the performance of concurrence is not within our discussion. Although HEXQ incorporates the idea of task learning, it is still constraint to the hierarchical decomposition. Therefore, HEXQ is a hybrid algorithm between task learning and hierarchical decomposition. Unfortunately, such a hybrid only results in that it can neither be a good task learning approach, nor a high performance algorithm.

We cannot obtain a complete road-map of the task relationship using these existing hierarchical approaches. While in our opinions, the details about the road-map of the task relationship is more important for complex task planning than simply decomposing a task into atomic subtasks. For example, for the exhaustive decomposition based approach, such as MAXQ, we obtain a clear road-map of the hierarchies of the abstract actions, and how the atomic actions are related with every abstract action. However, this does not provide a clue about how these abstract actions are executed in the whole system. In other words, the exhaustive decomposition provides the details in task decomposition, and clear relationship about different levels of subtasks, but it does not interpret how these subtasks are linked with each other, rather than simply to link with its parents or children.

Under such a background of algorithm design, we decide to withdraw the original objective of directly implement the real-world robot application, and turn to the development of an easy, effective and task-oriented approach, in order for the agent to automatically learn the task model from the original domains. We develop an approach called episodic task learning (ETL) for the complex task problems in real-world applications. Although there still exists hierarchical structure in ETL, different from the existing approaches, ETL will only decompose a flat MDP into two hierarchies, one for the representation of subtask relationships, and the other for describing the details of experiences. By our analysis and experiments, ETL outperforms the existing hierarchical approaches, such as MAXQ and HEXQ.

### 3.3 Task-Oriented Design

#### 3.3.1 Tasks and MDPs

A task is a basic unit of the daily activities of humans and intelligent agents. Generally speaking, a task contains a set of states, a set of actions and some certain relationships, with an initial state  $s_0$ , where it starts from, and a single or several absorbing states  $s_g$  (goals and/or termination states), where it ends in (RockSample [9] is a typical example using termination state instead of goals. Theoretically, the infinite tasks may not have goal or termination state, we can simply set  $s_g = null$ ).

A task planning is very close to an MDP planning. On one hand, a task domain consists of a series of task states whose transitions conform to the Markov properties. Therefore, a task planning conforms to an MDP planning. On the other hand, every MDP, whether it has a finite or infinite state space, we can describe it as a process with an initial state, a set of absorbing states, which conforms to the definition of the task. Therefore, the optimal policies of an MDP correspond to the optimal policies of a task. A task-oriented design addresses the discerning of the internal task-relationships in a problem domain, which the planner can utilize for the planning.

A series of prior works have considered the task abstraction for MDPs [2, 3, 56]. A key point of the research is how to extract the abstraction, rather than whether or not a planner needs the abstraction. Considering an exhaustive decomposition based approach, MAXQ decomposes a task into subtasks, until it reaches atomic actions. An abstraction of the task is correspondingly obtained in each decomposition. HEXQ relies on state variables for the task abstraction. It stochastically explores the state space, so as to gather the states that change value with the same frequency and put them in a level of the hierarchies. Such a frequency analysis approach has been challenged in [18], for a reason that it cannot work

well in every problem domain. For example, it wrongly finds out a state influence graph in the simple but famous coffee task [57].

Before continue our discussion, let us review and make clear some notations. Subtasks are the children of task decomposition. A subtask owns every property of a task. The objective of the decomposition is to obtain a sub state space, a sub action space, and a set of subtask correlated absorbing states. An atomic action is the action defined in the MDP. It may be a set of instructions for controlling the robot. An abstract action is a temporally-extended atomic actions. In the task-oriented design, it corresponds to a subtask. An option [54] is a specially defined subtask. Its entrance is defined as a set of states, and its exit a set of termination states with corresponding probabilities, although most of the termination probabilities of the current domains are still 1, and they have only a single element for the set of entrance states, which is in accordance to the specification of the subtask and the abstract action. In fact, the difference of these terms exists more on the naming. Both the subtask and the abstract action can be extended smoothly to the specific definition of the option. On the other hand, an MDP together with a set of subtasks, a set of abstract actions, or a set of options, will constitute a semi-Markov decision process.

Let us bypass the minor difference among these naming mechanisms and return back to our discussion. Different from the traditional hierarchical decomposition approach, the task-oriented design is based on an idea of experience extraction. This idea has been expressed in HEXQ [3]. Unfortunately, the algorithm of HEXQ makes a statistic for the frequency changes of state variables. In fact, although such a statistic also belongs to the analysis of experience, it is an approach of general data mining, which is quite different from our task-oriented design.



### 3.3.2 Experience and Task Abstraction

In order to understand the experience, let us first take a review of the stochastic. An MDP models a stochastic process in a planning. It is based on such an assumption: the transition from a state to another state is a stochastic process, with a transition probability. Although every transition of the states is stochastic, since the observation of every state is determined in an MDP, the planner can always find out its current state. Now that the process is stochastic, whether it is possible to discover some useful experiences, with which the planner can speed up its future planning and learning is an interesting question, both for theoretical analysis and for engineering design.

Firstly, an experience must be a relatively fixed portion of a stochastic process. Only a relatively fixed process is possible to be further reused. Secondly, an experience must be related with a temporally-extended action, i.e. an abstract action. Since a single atomic action can be reused directly, there is no need to model it as an experience. Thirdly, an experience must be related with several state abstractions, one for the initial state, and the others for the absorbing states. Formally, an experience must be an abstract action, which is an MDP, denoted as  $M^e$ , plus its optimal policies  $\Pi^*$ . This indicates, in a given task state, if the planner takes a corresponding abstract action, without any additional computing, the planner can perform  $\Pi^*$  directly.

In fact, the concept of experience exists in every hierarchical approach, even if it is not specially addressed. Then, by what standard or principle it is possible to extract the most useful experience is the real art. First, an optimal experience  $e$  should have the maximum possibility to reuse: If there exists another classification of the experiences, its maximum frequency of reuse should not be greater than that of the optimum experience. Second, in order to maximize the reuse, the  $e$  should be minimized: There exists no experience that is possible to be extracted from  $e$ .

Now that the focus of the task-oriented design is to extract useful experiences, the approach such as exhaustive decomposition just helps us to achieve such an objective. A *task state abstraction* is a process to obtain task-related abstract states and abstract actions, indicating to what degree we have accomplished a task. Every experience  $e$  corresponds to a set of abstract task states, with an initial task state representing the execution condition, and other task states representing the results.

A *Markov stochastic task domain*  $M^a$  is a task-oriented equivalence model of a Markov stochastic domain  $M$ , such that the optimal policies of  $M$  correspond to the optimal policies of  $M^a$ . We have,  $M = \langle M^a, E \rangle$ , where the  $M^a$  is a task abstraction of the original Markov stochastic domain  $M$ , and the  $E$  is an experience space. We will introduce the  $M^a$  and the  $E$  further in the following parts.

### 3.4 Episodic Task Learning

Episodic task learning (ETL) is an approach to learn a Markov stochastic task domain from a Markov domain. After the Markov stochastic task domain is learned, we will use it as an equivalent model for the planning of the original domain. In other words, we will obtain a more efficient model, without changed the controlling effect of the system. The Markov stochastic task domain is more close to the cognitive thinking, which provides an efficient computation model for the intelligent agent.

In this part, we introduce this approach beginning with the review of states in MDPs.

#### 3.4.1 Differentiate the States

An MDP consists of a set of states. A transition between a pair of states is performed by taking an action from the action space. The task analysis needs to decompose the original MDP  $M$  into a set of sub-MDPs, with sub state spaces and sub action spaces (In this paper, we consider the distinct states, rather than the continuous states; in fact, any contin-

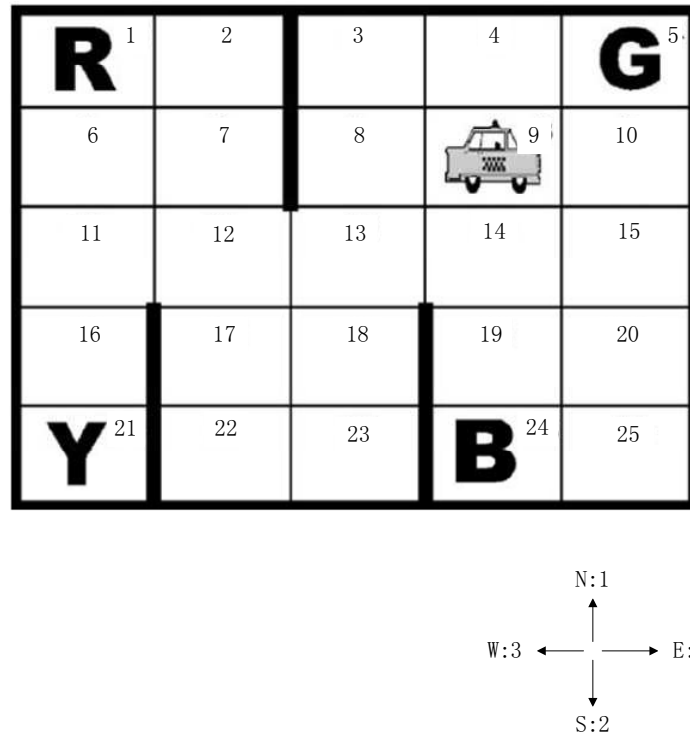


Figure 3.3. Taxi Domain.

uous state space can be approximated by a distinct state space.). An alternative description of such a decomposition is the use of state variables [3, 18]. Let us take the taxi domain [58] as an example to explain this approach.

The taxi domain is introduced as an episodic task: A taxi inhabits a  $5 \times 5$  grid world (Figure 3.3). There are four specially-designated locations  $\{R, B, G, Y\}$  in the world. In each episode, the taxi starts from a randomly-chosen location. There is a passenger at one of the four locations (chosen randomly), and he wishes to be transported to one of the four locations (also chosen randomly). The taxi has six actions

$\{\text{North, South, East, West, Pickup, Putdown}\}$ .

The episode ends when the passenger is deposited at the destination. We have three state variables: the taxi location, the passenger location  $\{R, G, Y, B, \text{Taxi}\}$ , and the destination  $\{R, G, Y, B\}$ . A combination of the state variables constitutes the states used in the flat MDP

solution (totally 500 states). For every state  $s$ , it is a distinct combination of  $n$  state variables  $x_i$ , i.e.  $s = x_1 x_2 \dots x_n$ . For any two distinct states,  $s_1 = x_1^{(1)} x_2^{(1)} \dots x_n^{(1)}$ ,  $s_2 = x_1^{(2)} x_2^{(2)} \dots x_n^{(2)}$ ,  $\exists i \in [1, n], x_2^{(1)} \neq x_2^{(2)}$ .

If a state  $s$  is composed of a single state variable, it is a *primitive state*. Otherwise, if  $s$  is a combination of multiple state variables, it is a *complex state*. Due to the Markov property, the states from a single state space are composed of the same set of state variables. Thus, for a state space with primitive states, we call it primitive state space. Otherwise, for a state space with complex states, we call it complex state space. For any single step transition  $T(s, a, s')$ , there exists and only exists a single state variable  $x_i$ , such that  $i \in [1, n], x_i^{(1)} \neq x_i^{(2)}$ , and  $\forall j \neq i \wedge j \in [1, n]$ , we have  $x_j^{(1)} = x_j^{(2)}$ .

### 3.4.2 Episodic Task State Abstraction

An abstraction of an MDP is a compaction of the original state space into a state space with abstract states. Given a state  $s_i$  in a complex space  $S$ , if  $s_i = x_1^{(i)} x_2^{(i)} \dots x_n^{(i)}$ , and  $\forall s_j \in S$  we have  $x_k^{(i)} = x_k^{(j)}$ , then we obtain an abstract complex space  $S - \{x_k\}$ , with  $s'_i = x_1^{(i)} x_2^{(i)} \dots x_{k-1}^{(i)} x_{k+1}^{(i)} \dots x_n^{(i)}$ . Here the state variable  $x_k$  is verbose and it is eliminated in the task abstraction. Such a *verbose state variable elimination* is a simplest task abstraction of a complex space. In the taxi domain, since the destination is designated in every episode, the state variable  $x_3$  (the destination) is verbose.

Another task abstraction of a complex space is the *state value abstraction*. Normally, a state variable  $x_i$  has a value space  $V(x_i)$ . In regard to the passenger location,  $V = \{R, G, Y, B, Taxi\}$ . Although a value abstraction does not change the state variable, it may change its meaning. Since the value abstraction for  $\{R, G, Y, B\}$  is a complex state *OffTaxi/UserLocation*, we have  $V = \{OffTaxi/UserLocation, Taxi\}$ . In fact, a more reasonable explanation of the state variable is the taxi status, with  $V = \{Empty, UserIn\}$ . However, there is no difference for the value space of the taxi status and the passenger

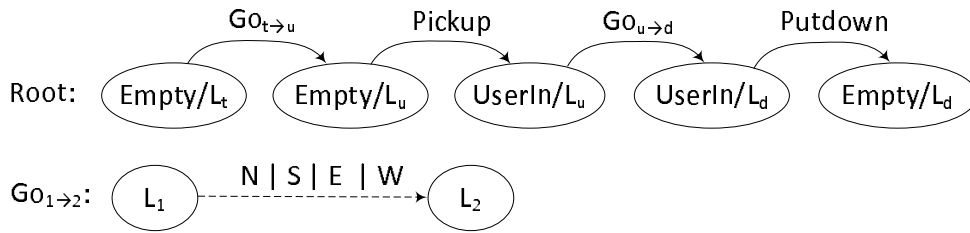


Figure 3.4. Episodic Task Abstraction for Taxi.

location. Hence it is feasible that we automatically obtain the abstraction without changing the original state variable.

An abstract task state (or simply task state) is a task-related complex state that is defined closely with the experience. Every experience corresponds to a set of task states, designating its entrance and exit. The state space of an experience is primitive. Therefore, one of the state variables of the experience’s task states can distinctly appoint the initial state, absorbing states as well as other states of the experience. The taxi domain has a primitive state space of the grid world. We obtain two experiences as  $Go_{t \rightarrow u}$  and  $Go_{u \rightarrow d}$  (Figure 3.4). The  $Go_{t \rightarrow u}$  has an entrance  $Empty/L_t$  (the taxi is empty and locates at its initial position) and an exit  $Empty/L_u$  (the taxi is empty and locates at the user location). The  $Go_{u \rightarrow d}$  has an entrance  $UserInL_u$  (the taxi has a user and at the user location) and an exit  $UserInL_d$  (the taxi has a user and at the destination). The abstract actions of  $Go_{t \rightarrow u}$  and  $Go_{u \rightarrow d}$  are still MDPs with the same state space, action space and transition probabilities, with only the initial state and the absorbing states being different. We call such an abstraction *isomorphic*.

In our episodic task abstraction, considering  $Empty/L_u \rightarrow UserIn/L_u$  and  $UserIn/L_d \rightarrow Empty/L_d$ , both of the transitions are direct: the abstract task states are the same with the complex states. Therefore, there is no need to create experiences for them. As a result, the actions of *Pickup* and *Putdown* are directly used in the task abstraction. Such an abstraction has already been used widely. In fact, both *Pickup* and *Putdown* cannot be finished in

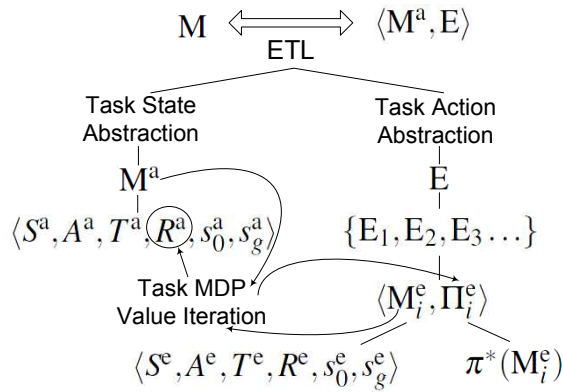


Figure 3.5. Episodic Task Learning Approach Diagram.

a single step, hence they are still abstract actions. However, we overlook the details and cite them simply as primitive actions, i.e. experiences. This indicates that our two-hierarchical task model, one for the application of experiences for the task abstraction, and the other for the extraction of experiences, is reasonable and feasible.

Now, we are able to obtain the abstract states for the task abstraction  $M^a$ . For  $M = \langle M^a, E \rangle$ , if the  $M$  is an MDP, then the  $M^a$  is also an MDP. We have  $M^a = \langle S^a, A^a, T^a, R^a, s_0^a, s_g^a \rangle$ , where the  $S^a$  is the state space of the abstract task, the  $A^a$  is the action space, including the abstract actions and the primitive actions, the  $T^a$  defines the transition probabilities, the  $R^a$  is the reward array. The system can learn the initial task abstract state  $s_0^a$  and the goal  $s_g^a$  easily from  $M$ . For an MDP, the planner can observe the states deterministically, hence the transition of the task states has a probability 1. Here the  $E$  is the experience space,  $E = \{E_1, E_2, E_3, \dots\}$ . For each  $E_i$ , we have  $E_i = \langle M_i^e, \Pi_i^e \rangle$ , where the  $M^e = \langle S^e, A^e, T^e, R^e, s_0^e, s_g^e \rangle$  is the tuple that defines the experience's MDP. The  $\Pi_i^e$  is the optimal policies of  $M_i^e$ , i.e.  $\Pi_i^e = \pi^*(M_i^e)$ . The ETL approach is depicted in Figure 3.5.

Compared to our the hierarchical approaches, the exhaustive decomposition approach (MAXQ) and a hybrid hierarchy approach (HEXQ) have multiple hierarchies in the taxi domain. These frameworks are depicted in Figure 3.6(a) and Figure 3.6(b).

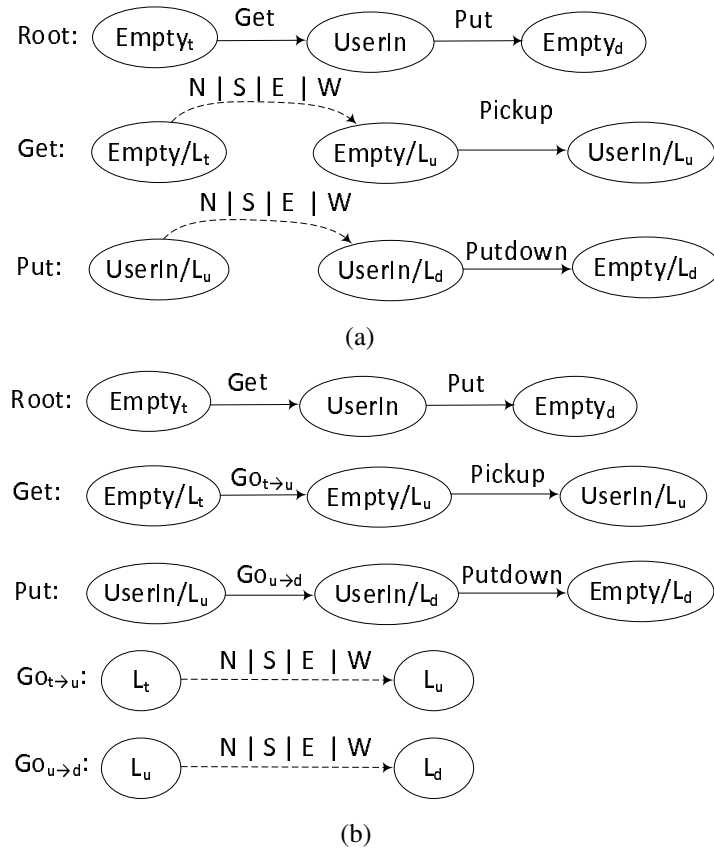


Figure 3.6. Task Abstractions for HEXQ and MAXQ: (a) Taxi HEXQ; (b) Taxi MAXQ.

### 3.4.3 Knowledge Acquisition by TMDP

The taxi task has an equivalence model  $\langle M_{mdp}^a, E_{mdp} \rangle$ , with two isomorphic abstract actions  $a_1^a, a_2^a$ . The idea is, with the knowledge of these isomorphic abstract actions, which can be acquired from the  $E_{mdp} = \langle \{M_{i,mdp}^c, \Pi_{i,mdp}^c\} | i=1..|a^a| \rangle$ , the  $M_{mdp}^a$  will become a standard MDP. Currently, the only knowledge missing for the  $M_{mdp}^a$  is the reward  $R^a(s, a_i^a)$ . We will obtain  $R^a(s, a_i^a)$  by a Task MDP (TMDP) value iteration. Since it is possible that an isomorphic abstract action  $a^a$  relates with multiple states, we assign an index for each state  $s$ , denoted as  $y(s)$ .

Details about the TMDP value iteration are listed in Algorithm 3. The difference between the TMDP algorithm and the MDP algorithm is, there is a single step value iteration

```

repeat
  for  $s \in S$  do
    for  $a \in A$  do
      if  $T(s, a, s) < 1$  then
         $V = \sum_{s' \in S^a} T^a(s, a, s') V_{t-1}^a(s')$ 
        if  $a \in A^a$  then
           $R^a(s, a) = M_{mdp}^e ssvi(a, y(s), V)$ 
        end if
         $V_t^a(s, a) = R^a(s, a) + \gamma V$ 
      end if
    end for
     $V_t^a(s) = \max_a V_t^a(s, 1 : |A|)$ 
     $\pi_t^a(s) = \arg \max_a V_t^a(s, 1 : |A|)$ 
  end for
   $t = t + 1$ 
until converge

```

**Algorithm 3:** TMDP Value Iteration

( $M_{mdp}^e ssvi$ ) (Algorithm 4). When  $T(s, a, s) = 1$ , the action  $a$  is not related with the state  $s$ . Thus we rely on  $T(s, a, s) < 1$  to bypass these unrelated actions. For the state  $s$  in  $a^a$ , the optimal policy  $\pi_t^e(a^a, y, s)$  in the experience is determined. Finally, we obtain the reward  $R^a(s, a)$  by the difference  $V_t^e(a^a, y, s_0^e) - V_t^e(a^a, y, s_d^e)$ .

### 3.5 Extended to POMDP Task

A more interesting and promising part of the episodic task abstraction is its extension to the partially observable domains. For a POMDP task  $M_{pomdp} = \langle S, A, O, T, \Omega, R, b_0, b_g \rangle$ , where  $b_0$  is the initial belief state,  $b_g$  are the absorbing belief states (goal belief states and/or



$$R^e(a^a, 1 : |M^e(a^a)|, s_d^e) = r$$

**for**  $s \in S^e(a^a)$  **do**

$$V_t^e(a^a, y, s) = \max_a [R^e(a^a, s, a) + \gamma \sum_{s' \in S^e(a^a)} T^e(a^a, s, a, s') V_{t-1}^e(a^a, y, s')]$$

$$\pi_t^e(a^a, y, s) = \arg \max_a [R^e(a^a, s, a) + \gamma \sum_{s' \in S^e(a^a)} T^e(a^a, s, a, s') V_{t-1}^e(a^a, y, s')]$$

**end for**

$$\text{return } V_t^e(a^a, y, s_0^e) - V_t^e(a^a, y, s_d^e)$$

**Algorithm 4:**  $M_{mdp}^e \text{ssvi}(a^a, y, r)$

termination belief states). The equivalence model can be  $\langle M_{pomdp}^a, E_{mdp} \rangle$ ,  $\langle M_{mdp}^a, E_{pomdp} \rangle$  or  $\langle M_{pomdp}^a, E_{pomdp} \rangle$ . In this work, we only consider the most popular model  $\langle M_{pomdp}^a, E_{mdp} \rangle$ .

The POMDP problem of RockSample $[n, k]$  [9] has a task domain of  $\langle M_{pomdp}^a, M_{mdp}^a \rangle$ . Difficulty of RockSample POMDP problems relies on the big state spaces. RockSample $[n, k]$  describes a rover samples rocks in a map of size  $n \times n$ . The  $k$  rocks have equally probability to be *Good* and *Bad*. If the rover samples a *Good* rock, the rock will become *Bad*, and the rover receives a reward of 10. If the rock is bad, the rover receives a reward of  $-10$ . All other moves have no cost or reward. The observation probability  $p$  for  $Check_i$  is determined by the efficiency  $\eta$ , which decreases exponentially as a function of Euclidean distance from the target.  $\eta = 1$  always returns correct value, and  $\eta = 0$  has equal chance to return *Good* or *Bad*.

The RockSample $[n, k]$  [9] has a task domain of  $\langle M_{pomdp}^a, E_{mdp} \rangle$ . It describes that a rover samples rocks in a map of size  $n \times n$  (Figure 3.7). The  $k$  rocks have equally probability to be *Good* and *Bad*. The rover can select from  $k+5$  actions:  $\{North, South, East, West, Sample, Check_1, \dots\}$ . If the rover samples a *Good* rock, the rock will become *Bad*, and the rover receives a reward of 10. If the rock is bad, the rover receives a reward of  $-10$ . All other moves have no cost or reward. The observation probability  $p$  for  $Check_i$  is determined by the efficiency  $\eta$ , which decreases exponentially as a function of Euclidean distance from the target.  $\eta = 1$







1	 2	3	4	 Exit
5	6	 7	 8	
 9	10	11	12	
13	 14	15	16	

Figure 3.7. RockSample[4,4] Domain.

always returns correct value, and  $\eta = 0$  has equal chance to return *Good* or *Bad*. In the task abstraction graph for RockSample[4,4] (Figure 3.8), the  $S_0$  represents the rover in the initial location, the  $R_i$  represents the rover staying with the  $rock_i$ , and the *Exit* is the absorbing state. Except for the node of *Exit*, there are 16 task states related with other nodes, indicating  $\{Good, Bad\}$  states for 4 rocks. Thus,  $|S^a| = 81$ . For the observations,  $|O^a| = 3k + 2$ : 1 observation for the rover residing on place without rock,  $k$  observations for the rover residing with a rock,  $2k$  observations for *Good* and *Bad* of each rock, and 1 observation for *Exit*. There are  $2k^2 + k + 1$  actions in the task domain:  $Check_1, \dots, Check_k, Sample$ . All of the abstract actions for a specific RockSample[ $n, k$ ] problem are isomorphic.

As a result of the learning, we got the knowledge of  $R^a(s, a_i^a)$  for the task abstraction, and  $\pi_i^e(a^a, s)$  for the experience. Thus, we can focus on the task abstraction  $M^a$  alone in future computation. After the fully observable task abstraction  $M_{mdp}^a$  is learned, the

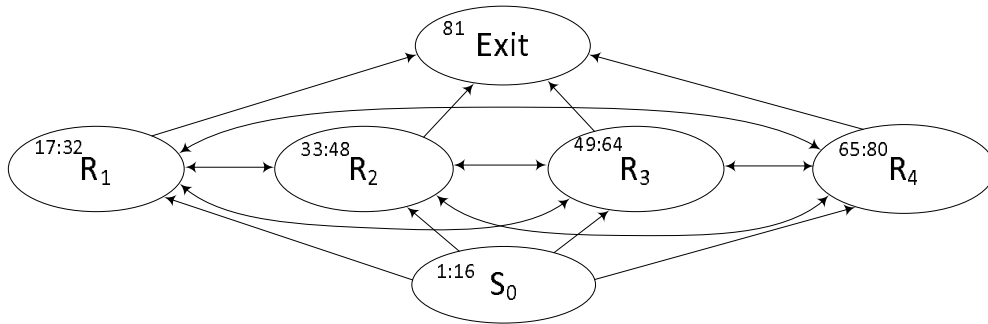


Figure 3.8. Task Abstraction for RockSample[4,4] (POMDP with abstract actions).

partially observable task abstraction  $M_{pomdp}^a$  becomes a general POMDP. We can solve it using any existing POMDP algorithms.

*RockSample*[ $n, k$ ] is an example of the equivalence model  $\langle M_{pomdp}^a, E_{mdp} \rangle$ . In our POMDP value iteration algorithm, the computational cost is  $O(|A^a||S^a|^2 + |A^a||B^a||S^a|) = O(|A^a||S^a|^2)$ , when  $|S| \gg |B|$ . The sizes for different arrays are listed in Table 3.1:

Table 3.1. Model Parameters of RockSample[ $n, k$ ] ( $n^2$  map,  $k$  rocks)

b	$ S^a $	$ A^a $	$ O $
$M_{pomdp}$	$n^2 2^k + 1$	$k + 5$	$n^2 + 2k + 1$
$M_{pomdp}^a$	$(k + 1)2^k + 1$	$2k^2 + k + 1$	$3k + 2$

In each round of value iteration, by rough estimation, we get the computational complexity of  $M_{pomdp}$  as  $O(kn^4 2^{2k})$ , and  $M_{pomdp}^a$  as  $O(2k^4 2^{2k})$ . This conclusion can be utilized to general POMDP problems that can be transformed to a task equivalence model with complex action  $M_{mdp}^a$  (the number of states being  $|S^a|$ ), and a task view has  $k$  task nodes (not including the initial and absorbing nodes). When  $|S^a| > \sqrt{2k^3}$ , the equivalence model created by ETL approach can greatly improve the computational capacity. However, if  $|S^a| < \sqrt{2k^3}$ , the equivalence model cannot improve the performance. Alternatively, it may

degrade a little the computational capacity. We can take this conclusion as a condition for applying the ETL approach on POMDP tasks, to improve the performance, although it can be used on every existing POMDP problem.

In sum, the ETL approach first analyzes the task model, and creates a task view and an action view. The action view is responsible for learning the model knowledge. The trained action view will then be saved for future computing in task view. The task view is an equivalence model with better computational capacity than the original POMDP model.

### 3.6 An Alternative Approach for the Modeling

In order to know the details of task states, in the ETL approach, we develop a *Task State Navigation* (TSN) graph to clearly depict the task relationship. A TSN graph contains a set of grids. Each grid represents a state of the task, labeled by the state ID. Neighboring grids with ordinary line indicate there is a transitional relationship among these states. Neighboring grids with bold line indicate there is no transitional relationship.

Let us take the taxi task [58] as an example, to interpret how to build the TSN graph, as well as how to construct the equivalence model  $\langle M_{\text{mdp}}^a, M_{\text{mdp}}^a \rangle$ . The taxi task is introduced as an episodic task. A taxi inhabits a  $5 \times 5$  grid world. There are four specially-designated locations  $\{R, B, G, Y\}$  in the world. In each episode, the taxi starts in a randomly-chosen state. There is a passenger at one of the four randomly chosen locations, and he wishes to be transported to one of four locations. The taxi has six actions  $\{North, South, East, West, Pickup, Putdown\}$ . The episode ends when the passenger has been putdown at the destination.

This is a classical problem, used by many hierarchical MDP algorithms to build the models. We present the ETL solution here. First, we build the TSN graph for taxi task in Figure 3.9. Label  $T_e$  represents the taxi is empty, and  $T_u$  indicates the taxi has user.  $L_t$  is

$T_eL_t$	$T_eL_u$	$T_uL_u$	$T_uL_d$	$T_eL_d$ ☆
1	2	3	4	5

Figure 3.9. TSN Graph for Taxi Task (MDP).

the start location of taxi,  $L_u$  is the location of user, and  $L_d$  is the location of destination. There are 5 task states in the TSN graph,  $\{T_eL_t, T_eL_u, T_uL_u, T_uL_d, T_eL_d\}$ . The initial state is  $s_0 = T_eL_t$ , representing an empty taxi in the random location. The absorbing state (goal) is  $s_g = T_eL_d$ , representing the taxi is empty and at the user's destination. We mark a star in the grid of the absorbing state. A reward of +20 is given for a successful passenger delivery, a penalty of  $-10$  for performing Pickup or Putdown at wrong locations, and  $-1$  for all other actions.

From the TSN graph, it is clear that the taxi task is a simple linear problem. The transition probabilities for the neighboring states are 1. There are four actions in the task domain,  $a^a = \{Go_{t \rightarrow u}, Go_{u \rightarrow d}, Pickup, Putdown\}$ , where  $Go_{t \rightarrow u}$  is the complex action going from  $L_t$  to  $L_u$ , and  $Go_{u \rightarrow d}$  is the complex action going from  $L_u$  to  $L_d$ . This model has two abstract actions  $a^{a(1)}, a^{a(2)}$ , and it is easy to know that  $S^{(1)} = S^{(2)}, T^{(1)} = T^{(2)}, a^{a(1)} = a^{a(2)} = \{n, s, e, w\}$ . However,  $a^{a(1)}$  and  $a^{a(2)}$  have different  $s_0$  and  $s_g$ . We call this kind of abstract actions *isomorphic action*.

Some existing POMDP problems have simple relationship in task domain, such that there is no abstract action. Thereafter, the equivalence model becomes a single model,  $\langle M_{pomdp}^a \rangle$ . The coffee task [57] has this kind of equivalence task model. It can be solved using action network and decision tree in [57]. Here, we propose the ETL approach for coffee task. The TSN graph for coffee task is shown in Figure 3.10. The  $L$  is an appointed as the initial state in the office. From the beginning  $O$ , since the weather has 0.8 probability to be rainy, denoted as  $R$ , and 0.2 probability to be sunny, denoted as  $\neg R$ , we get the transition

probability from  $L$  to  $R$  and  $\neg R$ . If the weather is rainy, the agent needs to take umbrella with successful probability of 0.8, and 0.2 to fail. We denote the agent with umbrella as  $U$ , and  $\neg U$  if it fails to take umbrella. If the agent has umbrella, it has probability 1 to be  $\neg L/\neg W$  (dry when it comes to the shop). If it has no umbrella and the weather is rainy, the agent will be  $\neg L/\neg W$  for 0.2, and be  $\neg L/W$  (wet in shop) for 0.8.  $\neg L/W$  has probability 1 to be  $C/W$  (coffee wet), and  $\neg L/\neg W$  has probability 1 to be  $C/\neg W$ . Whether it be  $C/W$  or  $C/\neg W$ , the agent has 0.9 probability to deliver the coffee to user  $H/W$  (user has wet coffee), and  $H/\neg W$  (user has dry coffee). The agent has 0.1 probability to fail to deliver the coffee  $\neg H/W$  (user does not have coffee and coffee wet),  $\neg H/\neg W$  (user does not have coffee and coffee dry).

There are 11 observations for this problem:  $r$  (rainy),  $\neg r$  (sunny),  $u$  (agent with umbrella),  $\neg u$  (agent without umbrella),  $w$  (agent wet),  $\neg w$  (agent dry),  $nil$  (none),  $h/w$  (user with coffee and coffee wet),  $\neg h/w$  (user with out coffee and coffee wet),  $\neg h/\neg w$  (user without coffee and coffee dry),  $h/\neg w$  (user with coffee and coffee wet). The observation probability for rainy when it is raining is 0.8, and the observation probability is 0.8 for sunny when it is sunshine. The agent gets a reward of 0.9 if the user has coffee, and 0.1 if it stays dry.

### 3.7 Comparison of Episodic Task Learning and Episodic Memory

Although the approach of ETL incorporates some ideas from episodic memory (EM), which makes it a computational framework with cognition of the relationship between tasks and the world, there are still some differences between these two techniques.

Episodic memory is first introduced in [1] as a theory of cognitive science. It analyzes how a human uses a memory to maintain a record of previous events, then the human makes use of these records as experiences to help decision-making and action planning. The

$S/\neg W$ 7		$uC/$ $W$ <sup>10</sup> ☆	
$\neg U$ 4	$S/W$ 6	$C/W$ 8	$\neg uC/$ $W$ <sup>11</sup> ☆
$R$ 2	$O$ 1	$\neg R$ 3	$\neg uC/$ $\neg W$ <sup>12</sup> ☆
$U$ 5	$S/\neg W$ 7	$C/\neg W$ 9	$uC/$ $\neg W$ <sup>13</sup> ☆

Figure 3.10. TSN Graph for Coffee Task (POMDP without Abstract Actions).

episodic, together with procedural and semantic memory system is further proposed for the planning of a cognitive robot [24]. Further research concerning how episodic memory is related with complex event processing is proposed in [5].

Episodic memory concerns what to decide after observing an event, such as which episode should be extracted according to the current observation. While an episode is a sequence of actions with a common goal. Although this is similar with the definition of ETL, the ETL approach that we propose addresses the optimization of complex tasks planning of a robot. The EM system holds records of specific, temporally-based past experiences, or episodes. A single episode is a period of task execution of the robot during which the goal of the robot does not change. This is coherent with the definition of abstract action from ETL and other hierarchical MDP approaches. Although both of the experiences of EM and ETL are just episodes, ETL does not hold temporally-based past experiences. Instead, the ETL approach holds a unique experience with an abstract state. The experience is modeled as  $E = \langle M^e, \Pi^e \rangle$ , where  $M^e$  is a Markov decision processes, while  $\Pi^e = \pi(M^e)$  is the optimal policies of  $M^e$ .

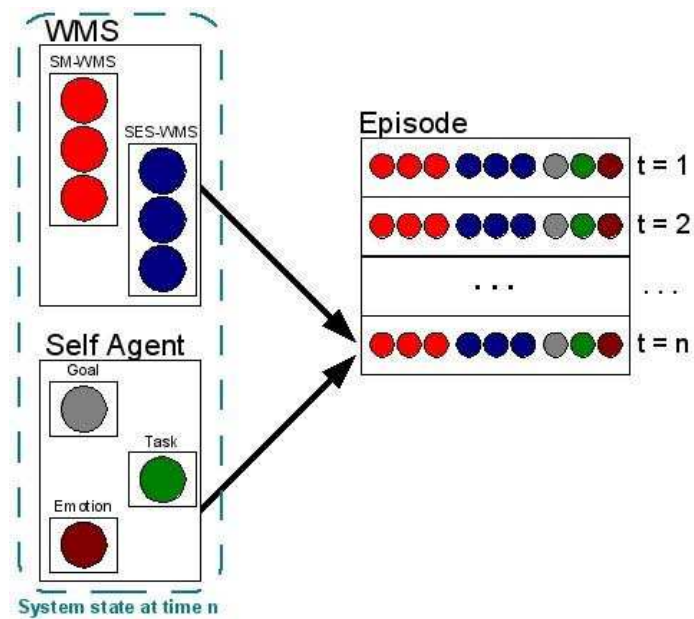


Figure 3.11. Episodic Memory Formation.

As shown in Figure 3.11, each episode contains link to the entire contents of the working memory system (WMS) and the output of the agent for the duration of a task. All constituent links points to Semantic Memory (SM) units, so statistics may be calculated about the frequency of use of each SM. Now that multiple episodes may relate with current cue, the resulting memory needs to contain common elements to be relevant to the current situation, and recently formed memories are more likely to contain applicable information. The third selection point is used to enhance the score of cues that are rare over those that are commonly seen. An episode is defined as a triple [5] of context, contents and outcome. Context is the general setting in which an episode happened; for the planning, this might be the initial state and the goal of the episode. This is similar with the definition of abstract action from ETL. Contents is the ordered set of events/actions that make up an episode; in the case of a planner, it is the planner itself. This is coherent with the experience  $E$  from ETL. The outcome of an episode is an evaluation of the episode's effect (success or failure). This is indicated by states in ETL, not necessarily to pointed out specifically.



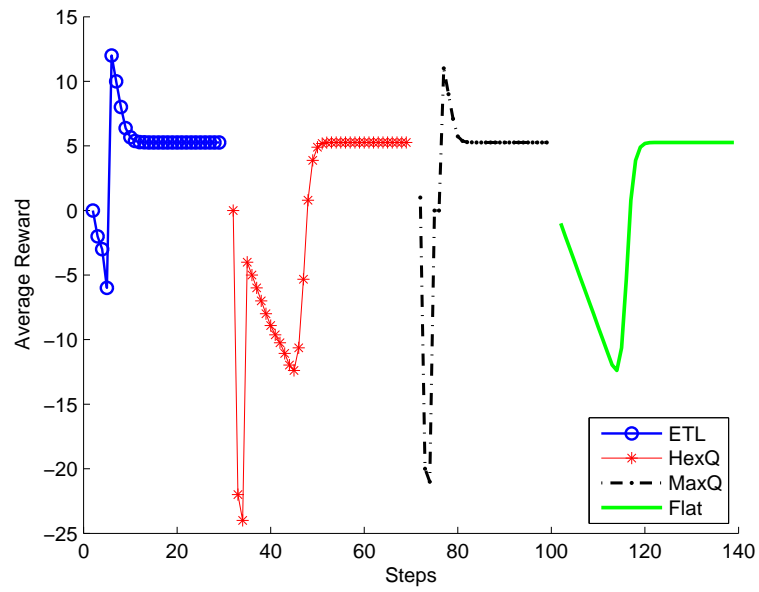
Another similarity between ETL and EM is that they both define the process that is used for planning in the situations that are dynamic in nature, changing the state of the world in complex ways. In this way, the procedural memory which addresses the skills that are constant to given conditions will not be addressed any more in both techniques. In fact, ETL takes the constant planning as atomic action. As we have discussed in Section 3.4.2, atomic actions may simply be experiences that we do not address any more, because they are constant with an scenario, just like what is discussed in procedural memory, which is a skill that is possessed by a human.

The difference between EM and ETL is, the EM framework considers more about the usability of memory, therefore it also incorporates the SM. While ETL aims to optimize the tasks, therefore it has a task model  $M^a$ , which is very important for the optimization of the planning and execution of tasks. And it provides the computation framework by state space, action space and the relationship between tasks and experiences.

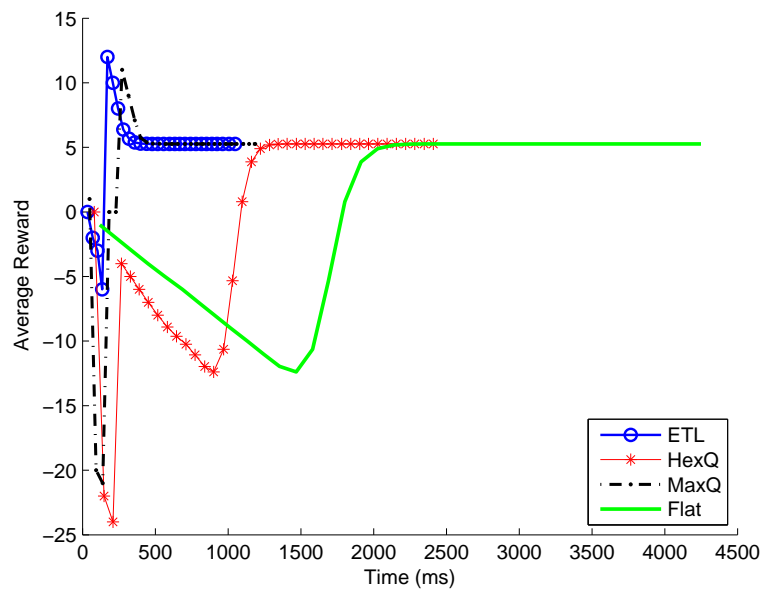
### 3.8 Experimental Evaluation

In the experiments, we implement every algorithm using dynamic programming. In the taxi domain, we guarantee every trial has the same configuration: the same user location and destination. Thus, the performance is only determined by the approach of hierarchical abstraction. Every result is an average of 10 times running. The error bars in the figures depict the standard deviation. However, only in Figure 3.13(b) the error bars are clearly displayed. There is almost no error bar in other figures because their standard deviations are close to zero.

In the first experiment, we examine the performance of every algorithm using a single trial. As a result, every algorithm can converge with no more than 50 iteration steps. We compare these algorithms by the iteration steps, as well as the convergence time. As



(a)



(b)

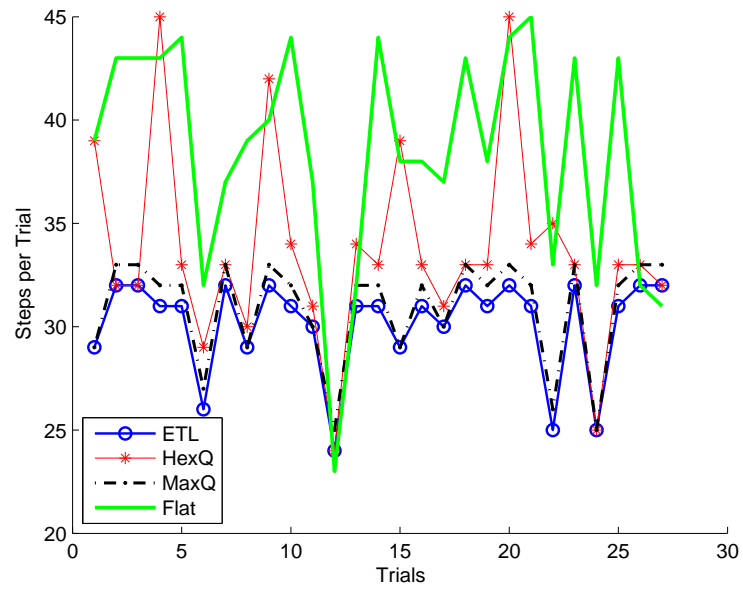
Figure 3.12. Comparison for Taxi Domain (Single Trial): (a) Reward by Iteration Steps; (b) Reward by Time.

depicted in Figure 3.12(a), the iteration steps of ETL and MAXQ are the same. HEXQ requires more iteration steps, and even more is the flat MDP. Concerning the convergence time (Figure 3.12(b)), ETL is the fastest among all algorithms. HEXQ is worse than MAXQ, and the flat MDP is the worst.

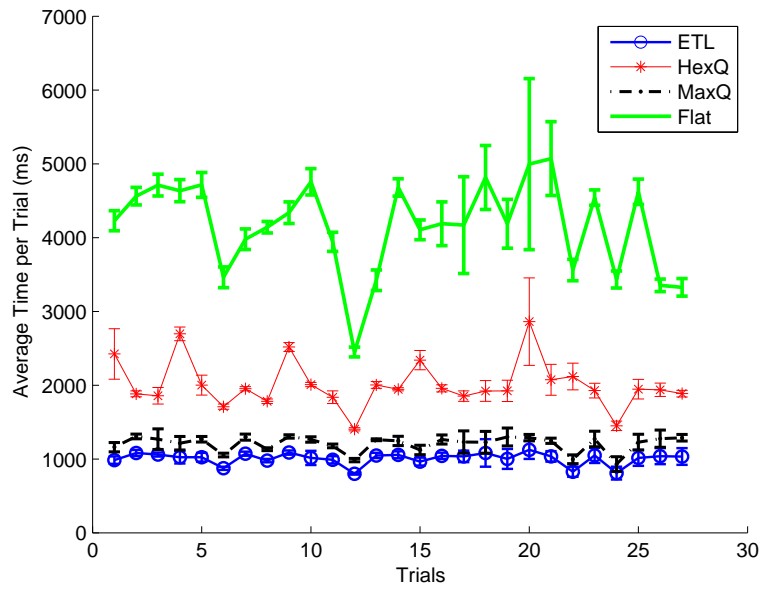
We extend the experiment to different trials. Altogether there are 27 distinct trials configured for the second experiment. From Figure 3.13(a), ETL has the least iteration steps in almost every trial. The iteration steps for MAXQ are close to ETL, HEXQ is worse than MAXQ and ETL, and the flat MDP is the worst. Concerning the convergence time (Figure 3.13(b)), the results are still consistent to the results in the experiment 1. The ETL approach is always the best in each experimental comparison.

In fact, both ETL and MAXQ will decompose MDPs into primitive state spaces. Since HEXQ is a mixed hierarchical approach, unless for specific cases that the frequencies exactly transform the MDPs into primitive state spaces, its performance will not be comparable to the ETL algorithm or the exhaustive decomposition base approach. Since the flat MDP has more complex states, its performance is often the worst. Comparing ETL and MAXQ, since ETL obtains a highly abstract task domain, which contains and only contains the task relationships for the planning, its performance is better than MAXQ, which does not have a clear task domain, but retains redundant hierarchical decomposition relationships of the task.

The third experiment is to examine the POMDP tasks. We use the same POMDP algorithm for each task. For the equivalence models, the experience  $E_{pomdp}$  is pre-computed. The system uses the trained data of  $M_{pomdp}^e$ . As shown in Table 3.2, the performance of each task abstraction  $M_{pomdp}^a$  improves greatly than the original POMDP model  $M_{pomdp}$ . The bigger the state space, this effect is more apparent. Considering  $RockSample[n, k]$ , the greater of  $n$  and  $k$ , the greater the performance of  $M_{pomdp}^a$  comparing with  $M_{pomdp}$ .



(a)



(b)

Figure 3.13. Comparison for Taxi Domain (Different Trials): (a) Convergence Steps; (b) Convergence Time.

Table 3.2. Performance Comparison of POMDP Tasks

<b>Domain</b>	<b>Model</b>	$ S $	$ A $	$ O $	<b>Reward</b>	<b>Time(s)</b>	$ B $
<b>Coffee</b>	$M_{pomdp}^a$	13	5	11	0.71	0.02	5
<b>RockSample</b> [4, 4]	$M_{pomdp}$	257	9	25	18.4	6.64	74
	$M_{pomdp}^a$	81	37	14	18.5	4.3	68
<b>RockSample</b> [5, 5]	$M_{pomdp}$	801	10	36	20.39	13.56	83
	$M_{pomdp}^a$	193	56	17	19.5	11.0	78
<b>RockSample</b> [5, 7]	$M_{pomdp}$	3201	12	40	22.4	289	98
	$M_{pomdp}^a$	1025	106	23	21.7	274	126
<b>RockSample</b> [7, 8]	$M_{pomdp}$	12545	13	66	21.6	1959	140
	$M_{pomdp}^a$	2305	137	26	21.8	896	232
<b>RockSample</b> [10, 10]	$M_{pomdp}$	102401	15	121	20.5	707600	231
	$M_{pomdp}^a$	11265	211	32	20.6	10309	391

### 3.9 Conclusion

We address the problems of how to obtain an effective and efficient hierarchical abstraction for MDPs. We propose a task-oriented design approach, the ETL algorithm. Different from the traditional hierarchical approaches, which addresses the task decomposition, ETL focuses on the task abstraction and experience extraction. With ETL, we obtain a task abstraction that retains the task relationships of the original MDP with much concise structure, so that we can achieve the same controlling effect with the original model with much improved performance. The ETL approach can also be applied to the POMDP task abstraction. Our experimental evaluation indicates a promising prospect of the ETL algorithm and the task-oriented design approach.

## CHAPTER 4

### PLANNING FOR MULTIMODAL PERVASIVE APPLICATIONS

#### 4.1 Introduction

With the development of sensor networks and pervasive technology, HCI needs to adapt to pervasive environments and changes from a single-computer based interaction to a pervasive interaction. In a pervasive interaction system, the communication between human and machine does not constraint to a single computer, but it involves the entire computer networks, robots and sensor networks. How to build up applications for a pervasive interaction becomes an important issue in pervasive computing, artificial intelligence and HCI. A typical example of the single computer based interaction is the human-robot interaction. A robot is often designed as an autonomous agent, with all kinds of sensors equipped and a single “mind”. For the pervasive interaction, we consider a virtual agent in the environment. The virtual agent has the power to utilize every sensing ability of the pervasive environment. As a whole, it provides an integrated interface to the human. Further discussion about robot and virtual agent can be found in [59].

In traditional dialogue management systems, human-computer interactions are tightly coupled with users’ speech responses. An agent speaks a sentence to a user, and waits for the responses. After the user’s speech is received and recognized, the system proceeds to the next interaction. Pervasive environments break through the constraint that a user has to interact with a single computer. The sensor network can also be an attendee of an interaction. After an agent asks a question or informs an affair, the user can optionally choose to reply or simply keep silence. Without the user’s responses, the agent can still discover the user’s intentions or preferences by reasoning information from the sensor network. This

kind of human-computer interaction forms a pervasive interaction. In this system, the communication between human and machine does not constraint to a single computer, but it involves the entire computer networks, robots and sensor networks. How to build up applications for pervasive interactions becomes an important issue in pervasive computing, artificial intelligence and HCI.

A typical example of a single computer based interaction is the human-robot interaction. A robot is often designed as an autonomous agent, with all kinds of sensors equipped and a single “mind”. For the pervasive interaction, we consider a virtual agent in the environment which has the power to utilize every sensing ability of the pervasive environment. It provides an integrated user-interface for the entire pervasive environment. Further discussion about robot and virtual agent can be found in [59].

In a pervasive interaction framework, an intelligent agent does not simply recognize the user’s speech. Instead, all kinds of events captured by sensor networks are accepted and analyzed by the system. Human activity recognition [60] becomes an important element for event capture. On the other hand, users do not have to focus their attentions on the interaction with a single computer. Now that the pervasive environment provides services as an integrated system, the user will be released from the tightly coupled interaction, without tampering the effect of the original “conversation”. As a result, the interactions will be loosely coupled with individual devices. In fact, the more loosely coupled a functional HCI application with users, the more degree of freedom users will obtain, thereafter, it will be the more friendly and easier to maintain the interactive relationship.

This paper provides a hierarchical multimodal framework for pervasive interactions. The application background is a reminder system interacting pervasively with humans, using decision-theoretic planning. We apply this technique on the planning system that reminds individuals with cognitive impairment, to support their activities of daily living (ADL).

## 4.2 Technical Preliminary

A Markov decision process (MDP) models a synchronous interaction of a user with a fully observable environment. It can be described as a tuple  $\langle S, A, T, R \rangle$ , where the  $S$  is a set of states, the  $A$  is a set of actions, the  $T(s, a, s')$  is the transition probability from a state  $s$  to another state  $s'$  using an action  $a$ , the  $R(s, a)$  defines the reward when executing an action  $a$  in a state  $s$ . By MDP, we want to find out the policy of every state, such that the overall planning is optimal.

A POMDP models an agent's action in uncertainty world. A policy for POMDP is a function of action selection under stochastic state transitions and noisy observations. At each time step, the agent needs to make a decision about the optimal policy based on historical information. A multimodal POMDP is an extensive form of POMDP that involves multiple types of observation sources. Although different types of observation sources share the observation space, their observation probabilities may be different. The traditional POMDP can be seen as a special form of multimodal POMDP that has only a single type of observation source.

A multimodal POMDP can be represented as  $\langle S, A, \Theta, O, T, \omega, \Omega, R \rangle$ , where the  $S$  is a finite set of states. In each time step, the agent lies in some state  $s \in S$ . The  $A$  is a set of actions. After taking an action  $a \in A$ , the agent goes into a new state  $s'$ . The  $\Theta$  is the set of multimodal observation types, and the  $O$  defines the types of observations. The  $T$  is the set of transition probabilities. The conditional probability function  $T(s, a, s') = p(s'|s, a)$  presents the probability that the agent lies in  $s'$ , after taking action  $a$  in state  $s$ . The agent makes an observation to gather information. The observation result is a pair  $(\theta, o)$ , where  $\theta \in \Theta$  is the observation type, and  $o \in O$  is the observation. This is modeled as a conditional probability  $\Omega(s, a, \theta, o) = p(\theta, o|s, a)$ . The  $\omega$  is the probability an observation belonging to an observation type,  $\omega(\theta) = p(\theta|event)$ . In a system that is modeled as multimodal POMDP, we have the following conditions for different observation types,



- (i)  $\forall(s, a, o), 0 < \sum_{\theta \in \Theta} \Omega(s, a, \theta, o) \leq 2;$
- (ii)  $\forall(s, a, \theta), \sum_{o \in O} \Omega(s, a, \theta, o) = 1.$

When belief state is taken into consideration, the states of the multimodal POMDP are changed to belief states. The original partially observable POMDP model changes to a fully observable MDP model, denoted as  $\langle B, A, \Theta, O, \tau, \omega, R, b_0 \rangle$ , where  $B$  is the set of belief states, i.e. belief space. The  $\tau(b, a, b') = p(b'|b, a)$  is the probability the agent changes from  $b$  to  $b'$  after taking action  $a$ . The  $R(b, a) = \sum_s R(s, a)b(s)$  is the reward for belief state  $b$ . The  $b_0$  is an initial belief state.

A POMDP framework is used to control an agent. The utility is a real-valued pay-off to determine the action of an agent in each time step, denoted as  $R(s, a)$ , which is a function of the state  $s$  and the action  $a$ . The optimal action selection becomes a problem to find a sequence of actions  $a_{1..t}$ , to maximize the expected sum of rewards  $E(\sum_t \gamma^t R(s_t, a_t))$ . In this process, what we concern is the controlling effect, achieved from the relative relationship of the values. After we use a discount factor  $\gamma$ , the relative relationship remains unchanged, but the values can guarantee to converge. If states are not fully observable, the goal becomes maximizing the expected reward of each belief. The  $n^{th}$  horizon value function can be built from previous value  $V_{n-1}$ , using a *backup* operator  $H$ , i.e.  $V = HV'$ . The value function is formulated as the following Bellman equation

$$V(b) = \max_{a \in A} [R(b, a) + \gamma \sum_{b' \in B} \tau(b, a, b') V(b')]$$

Here,  $b'$  is the next step belief state,

$$\begin{aligned} b'(s) &= b_t(s') \\ &= \eta \Omega(s', a, \theta, o) \sum_{s \in S} T(s, a, s') b_{t-1}(s) \end{aligned}$$

where  $\eta$  is a normalizing constant.

When optimized exactly, this value function is always piece-wise linear and convex in the belief space.

For the transition probabilities of belief points, we have

$$\begin{aligned}\tau(b, a, b') &= \sum_{\theta \in \Theta, o \in O} [\Omega(b, a, \theta, o) T(b, a, \theta, o, b')] \\ &= \Omega(b, a, \theta, o)\end{aligned}$$

where  $T(b, a, \theta, o, b') = 1$  if  $SE(b, a, \theta, o) = b'$ ,  $T(b, a, \theta, o, b') = 0$  otherwise.  $SE$  is an algorithm specific belief state estimation function.

We can find out  $\Omega(b, a, \theta, o)$  by the following processes. First we compute  $b'$ , where  $b'(s) = b(s') = \eta \sum_{s \in S} b(s) T(s, a, s')$ , where  $\eta$  is a normalizing factor. By  $\Omega(b, a, \theta, o, s) = \eta b'(s) B(\theta, o, s)$ , we can finally obtain the transition probability for  $\tau(b, a, b')$ .

### 4.3 Pervasive Human-Computer Interaction

#### 4.3.1 Sensor Coverage in Functional Areas

Different sensors can be deployed in the functional areas of a pervasive environment. A simple but effective deployment is to put sensors in each functional area, such that the field of interest is detected. As a result, we build up an activity recognition system for the pervasive environment. there is a labeled sensor. The detection of activities becomes an identification of sensor tag as well as time stamp. The activity for time  $t$  becomes a posterior probability for the sensor with maximum instant change, between the measurement in time  $t$  and the measurement in time  $t - 1$ . For time  $t$ , let  $y_t$  be the activity,  $x_t$  be the label of sensor, the probability for the activity is given by  $p(y_t | x_t, y_{t-1})$ .

Difficulties of activity recognition come from concurrent activities and noises from sensor readings. When participants watch TV, they may go to the kitchen and do some washes. During the wash activity, they may return back to answer a call. These concurrent activities may result in a confusion of the activity recognition system. Existing research

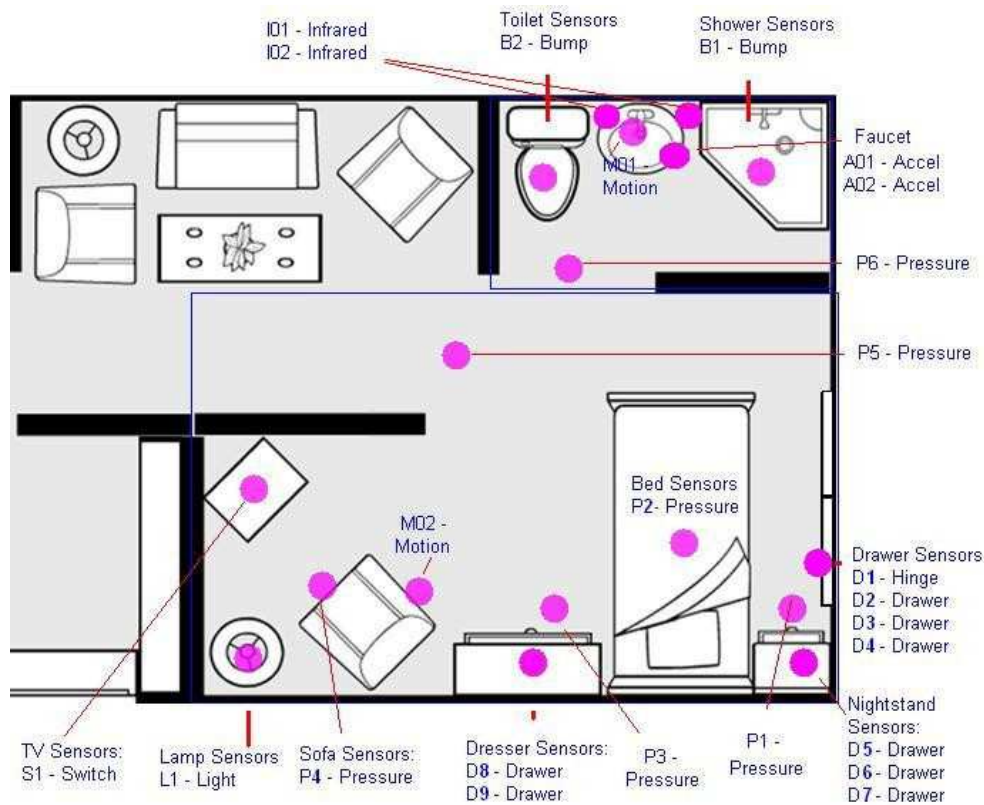


Figure 4.1. Sensor Deployment in A Pervasive Environment.

about the activities falls into several categories. One is the Bayesian inference, to recognize the activities [7]. Another is the classification techniques and vision-based analysis. These are often utilized to recognize human's affective state, physiology and behavior. Further research involves identifying the human's intentions for the activities, using hidden Markov models [61, 60] or conditional random fields [62]. If there are several persons in a home, the identification of activities about a specific person becomes even harder. We have to rely on face recognition or speaker recognition to differentiate the persons.

#### 4.4 Dynamic Fusion of Sensor Data

Results from the activity recognition are events of the user's activities. An event is a segment of information about the user's activity, including activity name, sensor label,

sensor type, time stamp. The sensor types for activity recognition can be accelerometer, infrared, light, pressure, video and audio. Benefit of sensor based activity recognition relies on the fact that, events are certain to be detected by sensor networks. The user does not have to explicitly inform the intentions. Therefore, this is a passive interaction between the user and the pervasive environment.

On the other hand, the user intentions can also be captured by the speech interaction, which is more directly than the recognition of activities. Although the speech interaction is a traditional way, it is an active interaction between the user and the pervasive environment. From the perspective of the users, it seems there is an intelligent agent in the pervasive environment living and interacting with them. This makes the pervasive environment friendlier. Especially for elders living alone, it is a necessity component. Defect of the speech interaction is that it has to rely on an explicit communication with human. Dialogue events come from the speech interaction. Dialogue events have the same format with sensor events, but the sensor type for speech recognition is audio, and it has its own sensor label.

The combination of sensor events and dialogue events provides a friendly human-computer interface. In this way, people do not have to respond to the prompts from dialogue management systems. The sensor networks make up the missing information by providing sensor events. During a time that people would like to interact with the dialogue management system, they can choose to inform the system their activities and preferences by speech interactions.

Results of the combination of sensor events and dialogue events are multi-observation of the reminder planning system (Figure 4.2). The system has to identify, analyze, correlate and reason for the observations. Since observations are obtained from different sets of noisy sensors, the reminder planning system needs to deal with the multimodal partially observable information.

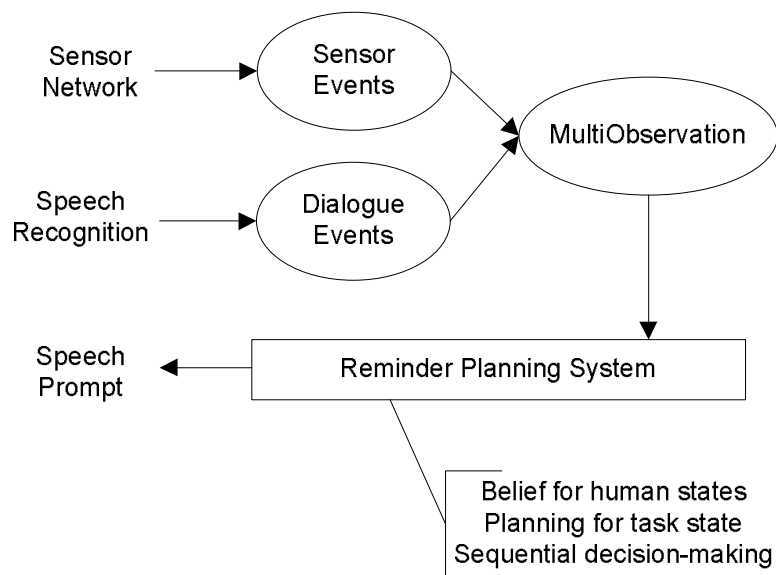


Figure 4.2. Sensor Events and Dialogue Events Become the Source of Multi-observation for Reminder Planning System.

#### 4.4.1 Event System and Multi-observation

Activity recognition creates events about users' activities. Every event is a segment of information about the user's activity, including activity name, sensor label, sensor type, time stamp. The sensor types for activity recognition can be accelerometer, infrared, light, pressure, video and audio. The benefit of sensor based activity recognition relies on the fact that, events are automatically detected by sensor networks, and users do not have to explicitly inform their intentions. Therefore, this is a passive interaction between humans and pervasive environments.

On the other hand, users' intentions can also be captured by speech interaction, which is a more direct interaction than the recognition of activities. From the perspective of users, it seems like an intelligent assistant lives and interacts with them. This makes the pervasive environment more friendly, especially for the elders that are living alone, the speech interaction is a necessary component. A defect of the speech interaction is that it has to rely on an explicit communication with the user. Dialogue events come from the

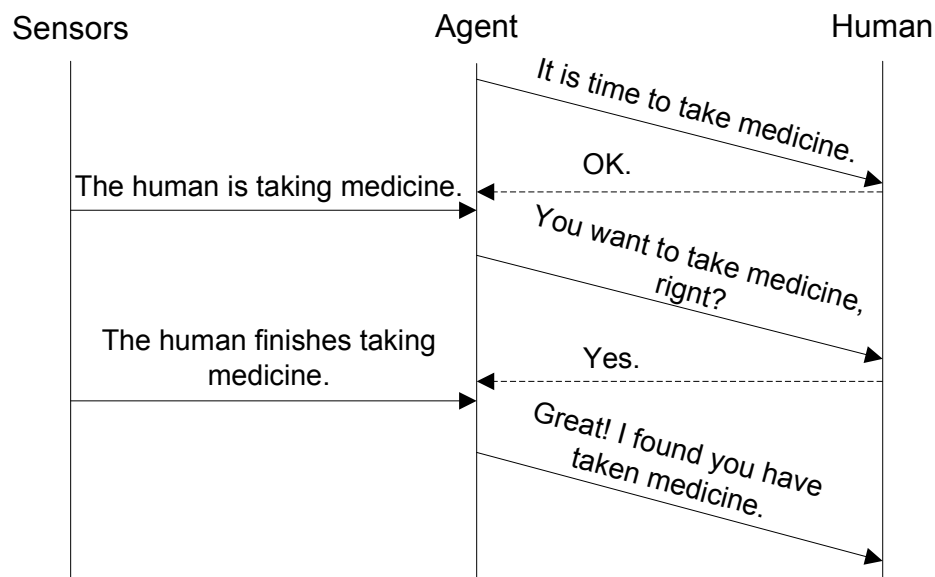


Figure 4.3. A Scenario for Loose Coupling Pervasive Interaction (Remind the User to Take Medicine): dashed-lines indicate optional speech response.

speech interaction. In the pervasive interaction, we can simply put it as an audio event, with a sensor-label of microphone.

The combination of sensor events and dialogue events provides a friendly and integrated interface. People do not have to respond to prompts from a dialogue management agent. The events from sensor networks make up the missing information. If people would like to interact with the agent, they can also choose to “tell” the system about their activities and intentions directly. Figure 4.3 describes a scenario of the loose coupling pervasive interaction. When an agent prompts a user to take medicine, the user can choose to reply or not. As soon as the user takes medicine, the agent will learn the states of the user’s activities by analyzing events from sensor networks. On the contrary, if it is an autonomous robot, the human’s responds (marked as dashed-line in the figure) normally should not be omitted. Since the interaction is tightly coupled with the robot, the human has to make explicit respond to every scenario of the dialogue. The difference of a pervasive interaction from a single-agent dialogue is, the pervasive interaction adapts to the environments exactly, rather

than adapting to the powerful function of a single agent. Thereafter, a pervasive interaction can provide a more friendly interface to users. It makes the pervasive environment to be an integrated intelligent system.

We can put the events of a pervasive interaction system into different categories, with every category belonging to a set of observations, and every observation comes from a set of noisy sensors. The application system has to identify, analyze, correlate and reason for the observations. We summarize the features of pervasive interactions as follows:

**Adaptation** Pervasive interaction makes users adapt to the environments perfectly;

**Redundance** Pervasive environments provide redundancy interfaces to sense and track the users' activities and intentions;

**Cross-Reference** Intelligent systems using pervasive interactions need to have the ability to learn the information by cross-reference of multiple interfaces.

#### 4.5 Hierarchical Multimodal Markov Stochastic Domains

A reminder is a planning system to prompt and remind the ADL of individuals with cognitive impairments. The system requires a multimodal partially observable framework that concerns both the events from speech recognition, and the events from sensor networks. We organize the system in two levels: MDPs based activity planning, and multimodal POMDPs based action planning. An action control component is used to manipulate the reminding tasks. Figure 4.4 provides an overview of the multimodal planning agent's working scheme. We introduce the components separately in the following parts.

##### 4.5.1 Activity Planning by MDPs

The activity planning is used to control the states of human activities. Every activity is represented as a state of the MDP model. We have the followings reasons to adopt MDP rather than POMDP for the activity planning:

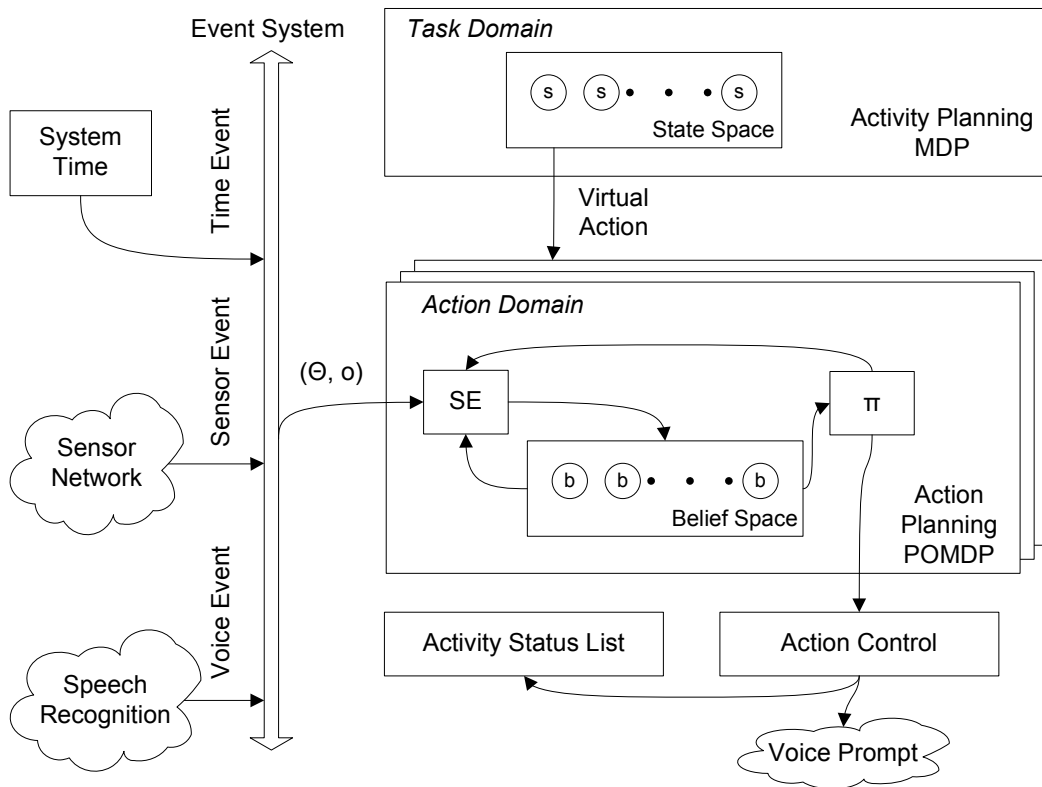


Figure 4.4. Hierarchical Multimodal Framework for Reminder Planning System.

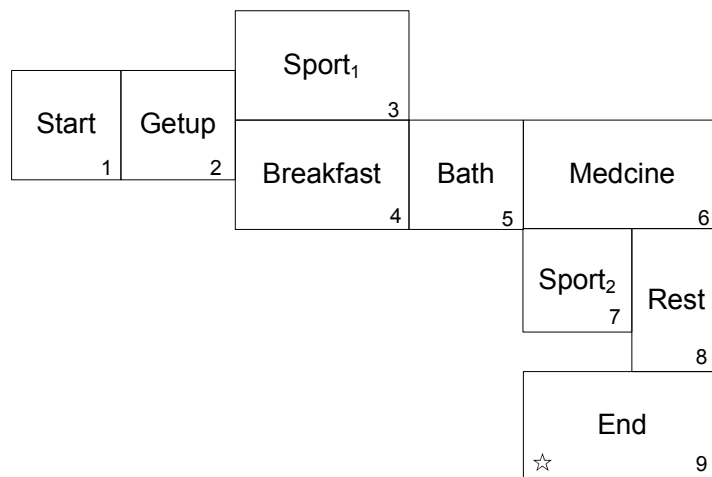


Figure 4.5. Activities for Reminding Tasks.



(i) The uncertainty of multimodal observations is considered in a lower action planning level, instead of the level of activity planning;

(ii) Since every termination state of action planning can be determined, we can distinct the states in activity planning.

Figure 4.5 is a *Task State Navigation* (TSN) graph to demonstrate some example activities for the reminder system. A TSN graph contains a set of grids, with each grid represents a state of the task, labeled by the state ID. Neighboring grids with ordinary line indicate there is a transitional relationship among these states. The reminding tasks start from the activity *Getup*, and end in *Sport<sub>2</sub>* or *Rest*, whose grids are labeled with a star. For most of the states, it is certain to transmit to the next state,  $T(s, a, s') = 1$ . As discussed above, the transitions from *Getup* and *Medicine* to the next states are both according to the transition probabilities. We have,  $T(\textit{Getup}, a, \textit{Sport}) = 0.5$ ,  $T(\textit{Getup}, a, \textit{Breakfast}) = 0.5$ ,  $T(\textit{Medicine}, a, \textit{Sport}) = 0.5$  and  $T(\textit{Medicine}, a, \textit{Rest}) = 0.5$ . The reward for state *End* is 10, and the rewards for other states are all 0.

We have only one action for the transition of different states. This is an abstract action, to indicate the end of an activity, and the start of next activity. In fact, the abstract action  $a$  for a state  $s$  is determined by the action planning subtask, which is related with the activity  $s$ . When the action planning subtask reaches the *End* state, it implies the abstract action is executed.

## 4.5.2 Action Planning by Multimodal POMDPs

### 4.5.2.1 Action Planning Subtask

For each activity of the reminding task, the system needs to undertake a subtask, to guide the user to execute the activity. It is an abstract action that is unrelated with the actual prompting speech of the system. The prompts are determined in the subtasks of

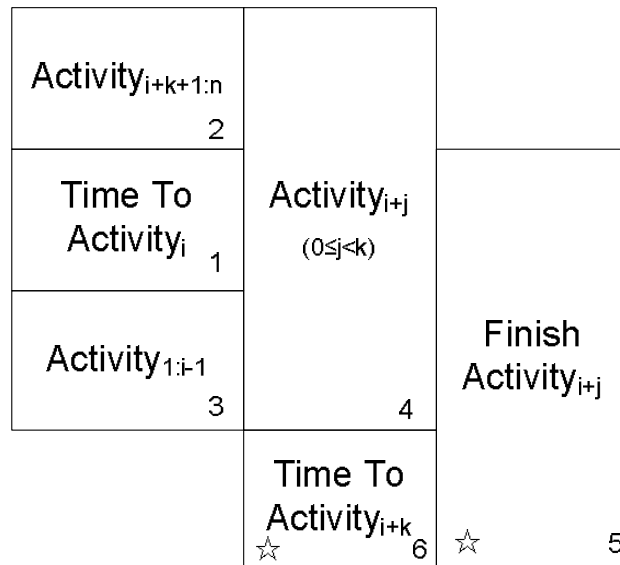


Figure 4.6. The Action Domain TSN Graph for Every Activity.

each activity. We put the execution of different subtasks to a unifying form. These subtasks accept partial observations from sensor events as well as speech recognition events. It forms two sets of noisy observations, the sensor observations and the dialogue observations. Thus the system structure becomes a multimodal POMDP.

The TSN graph for the multimodal POMDP is presented in Figure 4.6. For each activity, there are six states in the action domain. For the initial state *TimeToActivity<sub>i</sub>*, there are three possible transitions, including a state *Activity<sub>1:i-1</sub>* for previous activities, a state *Activity<sub>i+k+1:n</sub>* for future activities, and a state *Activity<sub>i+j</sub>* for current activity. If the system assumes the user is taking activity *Activity<sub>i+j</sub>*, it may accept the observation from previous activities, or future activities. Due to the noises of sensors and speech recognition, these observations may be correct or not. Thus, the transitions from the state *Activity<sub>i+j</sub>* to *Activity<sub>1:i-1</sub>* and to *Activity<sub>i+k+1:n</sub>* are both small probabilities. The transitions from the state *Activity<sub>i+j</sub>* to *FinishActivity<sub>i+j</sub>* and to *TimeToActivity<sub>i+k</sub>* have higher probabilities. The state to which it transmits is related with the current observation.

Since both  $TimeToActivity_{i+k}$  and  $FinishActivity_{i+j}$  are termination states in the action domain, when the system reaches them, it will transmit to the next activity in the activity domain MDP. However, in the state  $TimeToActivity_{i+k}$ , we cannot determine whether or not the activity is finished. Only in  $FinishActivity_{i+j}$ , will the system record a status of  $Activity_{i+j}$ .

There are three activity states,  $\{unfinished, executing, finished\}$ . We establish a list to record the status of every activity in a day. This will be helpful for the system to find out unfinished activities, for future data analysis, or for reminding the user directly.

#### 4.5.2.2 Multimodal POMDP Framework

Details of different activities are defined in a POMDP framework. Each state is related with two actions, *Prompt* and *Wait*. The *Prompt* indicates the planning system needs to provide prompt to the user. The actual content of a *Prompt* is determined by an action control, which we will introduce in next part. The *Wait* means the system takes an action of waiting.

The number of observation types (denoted as  $|\Theta|$ ) is 2, representing dialogue events and sensor events. There are  $|S|$  different observations in observation space, i.e.  $|O| = |S|$ . The dialogue events and sensor events are mapped into observations of different types. In fact, even if the system takes the action of *Wait*, it still has a transition probability to next states. The states to which the system transmits can be determined by the observations. Normally, the observation probabilities of dialogue events are lower than that of the sensor events, and it will decrease with the increase of age for the elderly.

The rewards are assigned as follows. For the termination states,

$$R(TimeToActivity_{i+k}, \cdot) = 1, \text{ and } R(FinishActivity_{i+j}, \cdot) = 2.$$

In multimodal POMDP, the  $\omega(\cdot)$  is a Bayesian statistic for an event belonging to an observation set. We have  $\omega(x) = p(x|event)$ , where  $x$  is an observation set. For other states,  $R(\cdot, Wait) = 0$ , and  $R(\cdot, Prompt) = -0.1$ .

#### 4.5.3 Action Control for Reminding Tasks

The action control helps to determine atomic actions of the planning system. Since the reminder system has complex task structure and actions, the atomic actions are not correlated strictly with the actions in multimodal POMDP subtasks. Thus, we can incorporate the benefit of POMDP framework, and constrain the state space to the core part of controlling tasks. We separate the atomic actions into two categories.

The first class atomic actions are related with the actions of multimodal POMDP subtasks. Let the atomic action be  $\mathbf{a}$ , we have  $\mathbf{a} = f(s_{activity}, s_{action}, a_{action})$ , where  $f(\cdot, \cdot, \cdot)$  is the combination function, to merge different states and actions into the text of speech. Considering the concrete actions, we have

$$\mathbf{a} = \begin{cases} Wait, & a_{action} = Wait \\ f(s_{activity}, s_{action}, Prompt), & a_{action} = Prompt \end{cases}$$

Thereafter, the actions in multimodal POMDP are translated to atomic actions in the reminder system, represented as the text of speech. Here is an example of the combination. Suppose  $s_{activity} = Medicine$ ,  $s_{action} = TimeToActivity_i$ , and  $a_{action} = Prompt$ , the atomic action will be a prompt like “It is time to take medicine”.

The second class atomic actions are related with the states of multimodal POMDP subtasks. Table 4.1 provides the correlations of the states and the second class atomic actions. These atomic actions are used to update the status types for the activities.

Table 4.1. Correlations for States and the Second Class Atomic Actions

ID	State	Atomic Action
1	$TimeToActivity_i$	none
2	$Activity_{1:i-1}$	update status for the activity identified from the event
3	$Activity_{i+k+1:n}$	none
4	$Activity_{i+j}$	none
5	$TimeToActivity_{i+k}$	$S(Activity_{i+k}) = unfinished$
6	$FinishActivity_{i+j}$	$S(Activity_{i:i+j-1}) = finished$

#### 4.6 Experimental Evaluation

The experiments are taken in a simulation environment. Seven activities are used as an example of the daily activities (Figure 4.5). We have discussed other settings in previous sections. We want to find out effects of the reminding tasks.

Figure 4.7 demonstrates learning curve of the average reward in each time step, for the multimodal POMDP. The average reward converges to 1.3.

We made a statistic of the actions in multimodal POMDP subtasks (Figure 4.8). The activity state 1 represents the state  $TimeToActivity_i$ , and the activity state 4 represents  $Activity_{i+j}$ . Both states are important for the reminding subtask. The time for executing an activity is often longer than preparing the activity. This effect is reflected in the action assignments of the planning tasks. We have two actions in every state, *Prompt* and *Wait*. In the state  $TimeToActivity_i$ , the actions of *Prompt* are about 2/3 of all actions, and *Wait* actions occupy the rest 1/3. This is reversed for the state  $Activity_{i+j}$ . During the activity, the reminder system will choose *Wait* about 2/3 of all actions, only 1/3 of them are *Prompt*. This indicates our multimodal POMDP planning for the reminder system provides reasonable prompts for the user, because the system does not have to prompt many times during an activity.

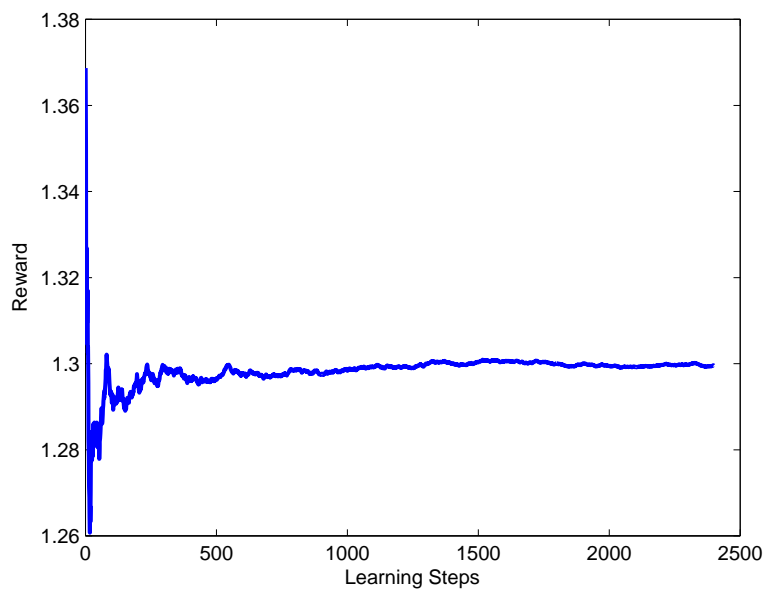


Figure 4.7. Learning Curve for the Multimodal POMDP.

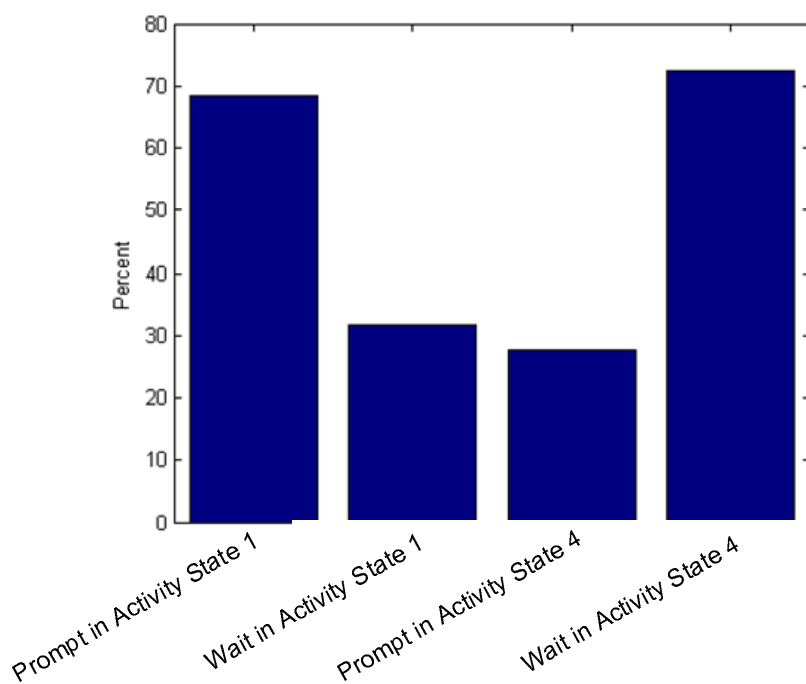


Figure 4.8. Analysis of the Actions in Multimodal POMDP Subtasks.

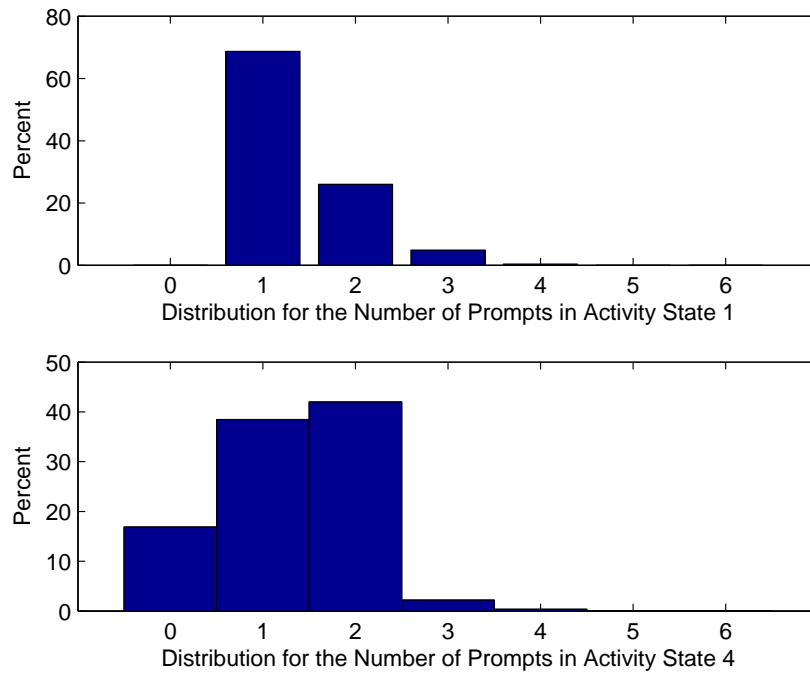


Figure 4.9. Statistic for the Prompt Action in the Reminder System.

Figure 4.9 provides a further statistic of the *Prompt* actions for different activities. For the activity state 1,  $TimeToActivity_i$ , there is not a single time the system does not prompt for the user. For about 70% of the activities, the system only prompts once. This percent decreases when the number of prompts increases. Only very few occasions the system prompt more than 4 times. This is comparable with the traditional constant of prompting tasks in a reminder system. The multimodal POMDP controls the system to adapt to multi-observation by which the system interacts with the user. The actions of the planning processes are more like a probability distribution, rather than a constant. For the state 4,  $Activity_{i+j}$ , there exists about 15% activities, in which the system goes to the next activity directly, without *Prompt*. For about 39% activities, the system prompts once for the user, and about 43% activities twice. When there is no prompt action, the system will perform *Wait* in every time step.

The distribution of prompts is determined by the observations accepted from the multimodal readings. The user can choose to respond to the system by speech or just keep silence. Without speech responses, the user's action is determined by sensor readings. This is the reason why the number of prompts in the reminder system is not constant. These results indicate the planning system helps us build a flexible mechanism for the reminding tasks, and it is a suitable design for the pervasive interaction based reminder systems.

#### 4.7 Related Work

Plan management technologies are used to model an individual's daily plans in [63, 64]. The authors consider reasoning in execution of the plans, so as to make flexible and adaptive decisions for reminding tasks. A decision-theoretic planning based on POMDP is proposed to assist people with dementia to wash hands [65]. A time-state aggregated POMDP method is proposed for the planning of human-robot interaction [66].

The development of multimodal and human-friendly HCI is a hot topic for assistive technology [67]. In this paper, we further discuss the framework of the planning system for pervasive interaction.

Multi-task management is a challenging topic in Markov stochastic domains. It has been considered as a dynamically merging problem for MDPs [17]. Another approach is the hierarchical POMDP [16, 68]. In these works, different actions are organized in the hierarchical structure of POMDP for the interaction. The designers implement the models as hierarchical decomposition of abstract actions, until every action becomes atomic. Our system adopts the idea of hierarchical design. The task domain is modeled as MDP, whereas the activity domain is modeled as POMDP.

Another related work tries to automatically construct HPOMDP, through data-mining techniques on collected data [69]. It assumes that a collection of data may contain many



repetitive patterns. If an algorithm is provided some preliminary about some given patterns, then it can take a data-mining on more data, to discover interesting pattern, such as a permuted of then original pattern. The authors want to make use of a Hidden Markov model to analyze this information, to help to construct the model of HPOMDP. Designing a learning approach to build up a Markov automata belongs to a stand-alone topic. Although the approach is interesting, we have to build up well about the pre-requisite conditions. Generally, researchers often rely on a simple Bayesian analysis to model POMDP.

#### 4.8 Conclusion

Designing an integrated, friendly and loose coupling user interface is important to pervasive environments. The pervasive interaction considers not only dialogue events, but also sensor events. The observations from both of these events are used to build up the belief space of a planning system. The reminder is a typical system for pervasive interaction. It is used to remind people with cognitive impairment, about the activities of their daily livings. We model the reminding tasks in MDPs. We normalize the reminding task of every activity so that it conforms to a multimodal POMDP model. Solving the hierarchical multimodal Markov stochastic domains becomes a solution of the pervasive interaction based reminder system.

## CHAPTER 5

### PLANNING FOR RISK-SENSITIVE TASKS

#### 5.1 Introduction

In many real-world applications of sequential decision-making, such as medical planning, surgery, therapy, marketing and robot planning, decision under uncertainty is a trade off return against risk [70]. Uncertainty is the unpredictability of a process [71]. Risk relates with the unpredictability of alternative outcomes [72]. More and more research works realize the importance of risk sensitivity for planners [73, 74]. In pervasive assistive environments, service robots may also have to make decisions according to their risk attitudes. For example, when the sensor network reports an event that the elder falls, the robot will face a time-critical task to help the elder directly or trigger a remote doctor to help. How could the robot decide what to do under this situation is determined by its attitude about the risk in different planning options. Another example is the gamble game. Suppose an investor with assets of \$10k has an opportunity to invest \$5k in a venture, which is equally possible to gain either \$15k or nothing. A risky option for the investor is to invest his money, which he may have a chance of 0.5 to gain \$15k, and a chance of 0.5 to lose \$5. While not to invest is an alternative option that will reward \$0 to the investor with certainty.

As a major branch of decision theory, utility theory explains this risk-aware behavior [31, 75]: Every intelligent agent has a monotonically non-decreasing utility function that maps its wealth level of every decision step to the resulting real-valued utility. An agent maximizes the expected utility of its future wealth level. The utility function determines its risk attitude. Linear utility functions imply a risk-neutral risk attitude. An agent is risk-neutral iff it makes decisions that maximize the expected future wealth level. Concave

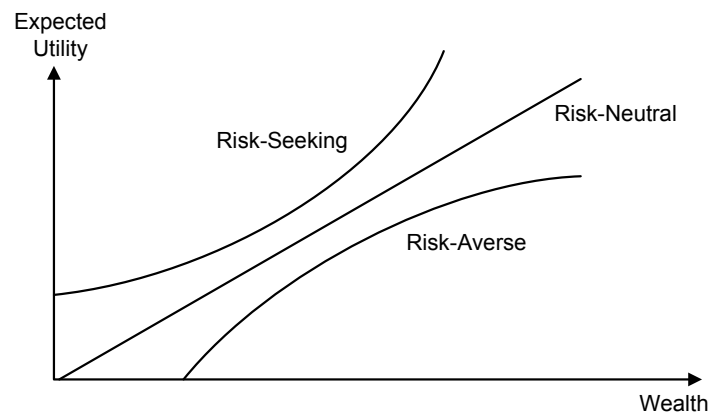


Figure 5.1. Relationship of Wealth, Risk Attitude with Utility.

utility functions imply a risk-averse risk attitude. An agent is risk-averse iff it makes decisions that do not maximize its expected future wealth level provided that the variance of its future wealth level is sufficiently reduced. On the contrary, an agent that is risk-seeking will have more preference to a risky option than a certain one, and it has a convex utility function. Since different agents can have different utility functions and thus different risk attitudes, they can make different decisions. The relationship of wealth, risk attitude and expected utility is depicted in Fig. 5.1.

Risk is often not independent of wealth. When the wealth increases to a certain level, the agent may change the order of preference. In financial decisions, managers may face a decision about the chosen of an optimal option among several investment plans. They have to evaluate the risk according to their wealth conditions. If the managers have more money, they will have stronger toleration of the risk. On the contrary, if they are pertaining less money, generally they will be less tolerable of the risk. The decisions from linear utility functions are independent of the initial wealth level and thus do not change as the wealth level increases. These utility functions are also known as zero-switch utility functions [76]. Comparably, for each pair of alternative decisions, however the initial wealth level and the wealth level of every decision step changes, if the agent only changes its preference once,

the utility function is one-switch [76]. Normally, if the utility function of an agent is one-switch, and it has two options  $p_1, p_2$  in its current decision step. It may prefer an option  $p_1$  when the wealth level  $w$  is lower than a threshold. However, if the  $w$  is greater than the threshold, it may change its preference to  $p_2$ , even if all other conditions hold the same. But if the utility function is zero-switch, the agent's preference will not change according to the  $w$ . An  $n$ -switch utility function permits at most  $n$  switches of preference between a pair of alternatives. Bell [76] has proposed that the one-switch utility function is more adaptable to the change of preference of real-world decision-making applications. In the following parts, we will address the most useful one-switch utility functions. In fact, the solving of a zero-switch utility function based problem domain is much easier than that is based on a one-switch utility function.

The measurement of risk falls into two types: cost based analysis and wealth based analysis. Cost, reward and wealth are all related with resource. Cost is the consumption of resource, reward may be consumption or gain, and wealth is the total amount of resource (money, energy, time, and others) an agent holds. In gambles, lotteries, and investments, utility is a function of wealth [77]. Decision tree is a prevalent method for risk-aware planning when wealth analysis is utilized. While the decision-theoretic planning finds optimal policies using value iteration [78] or policy iteration, using backward induction. This process proceeds reasoning backwards, from the end of a problem to determine a sequence of optimal actions. In this way, risk is easier measured by reward. Since cost is a special form of reward, the existing Markov decision processes (MDPs) based risk-sensitive planning approaches address cost analysis as a measurement of risk [72, 79, 80].

On the other hand, it is hard for a planning system to calculate the wealth of every step directly in sequential decision-making. In some simple problems, the expected wealth for every state can be calculated easily. Considering the gamble game we have mention

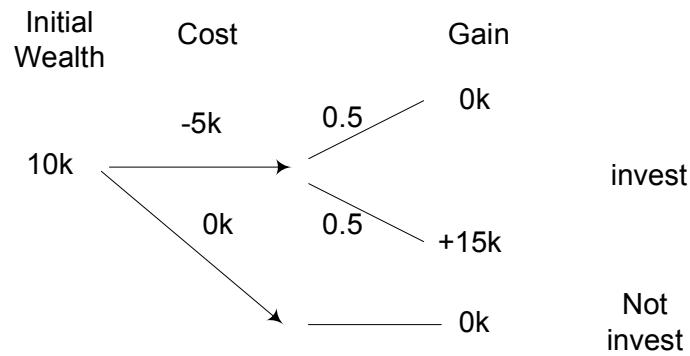


Figure 5.2. Decision Tree for the Investment of a Venture.

above. We can use a decision tree (Fig. 5.2) to compute the expected wealth of the investment,

$$0.5 * (10k - 5k + 0k) + 0.5 * (10k - 5k + 15k) = 12.5k.$$

Here, the  $5k$  is the cost of the venture, and the  $15k$  is the gain. Initially, the wealth of the investor is  $10k$ . If he does not invest the venture, the initial wealth of  $10k$  will be reserved. If he invests, then the wealth will expect to be  $12.5k$ . Since the expected wealth ( $12.5k$ ) of the investment is greater than the wealth of no investment ( $10k$ ), the investor will prefer to invest. However, for complex problems that cannot be organized into decision tree, the outcomes will not be so apparent. The block world domain [72] is such an example, which we will introduce in the experiment section.

In this example, we assume a zero-switch utility function  $u(x) = x$ , i.e.  $u_{invest} = u(12.5k) = 12.5k$ ,  $u_{-invest} = u(10k) = 10k$ . Since  $12.5k > 10k$ , the investor will prefer to invest the venture. If the utility function  $u(x)$  is one-switch, then given a wealth level  $W$ , which is determined by the planner's specific  $u(x)$ , when  $x_1 > W$  and  $x_2 > W$ , even though  $x_1 > x_2$ , the utility function may change, such that  $u(x_1) < u(x_2)$ . However, it is  $u(x_1) > u(x_2)$ , when  $x_1 < W$  and  $x_2 < W$ . As a result, when using a one-switch utility function to make the decision, the investor may change his preference according to the

computing results according to the level of his wealth. However, in a one-switch utility function, there is only a single change of the preference according to the wealth level.

Although wealth analysis is more reasonable to applications, cost analysis is often easier for the computation and thereafter often favored by researchers [72]. Cost analysis considers the resource that is consumed. It has a hidden presumption: the utility is independent of wealth (the initial wealth is zero). Take the robot planning as an example: If we concern the minimization of the total amount of time interval a robot spends in a planning task, we minimize the sum of the time-cost from every decision step. This will ease the computing of the optimized results. However, concerning risk attitudes and one-switch rules [76], the estimated optimal policy  $\hat{\pi}$  discovered simply by cost will be challenged in real world risk-aware applications. In worst cases, the  $\hat{\pi}$  may even be reverse to the real optimal policy  $\pi$ , after the wealth is considered not to be independent of risk. In the robot planning, if we simply consider the summary time-cost, we overlook how this planning is related with the initial wealth - in this case, it is the scheduled total time interval to finish this task, which is important to planning with risk attitudes. For time-critical tasks such as helping a falling senior, rescuing victims in disaster [81], also considering the total energy of an autonomous robot is often certain, to minimize the total time-cost within scheduled time interval will be more reasonable to real applications than simply to concern the time-cost itself. This is also known as planning under time constraints in prior research [82].

This paper further discuss the discrepancy between a cost based planning and a wealth based planning. We introduce a bridge algorithm to connect cost and wealth. Given a utility function  $u(x)$  (either zero-switch or one-switch), and an initial wealth  $w_0$ , a bridge algorithm helps us find the optimal policies  $\pi$ , based on the suboptimal policies of cost analysis. With such a bridge between cost and wealth, we can solve any wealth dependent risk planning problems that have been solved previously only by decision tree. Since the

backward induction based MDP is more powerful framework, the bridge algorithm also helps us solve complex problems that decision tree cannot provide a solution.

Although the prior research in [79, 83] also considered to incorporate a wealth level  $w$  in a risk-sensitive planning, it is distinctly different from our work from the definition of question domains to the framework and the algorithm. The initial wealth level  $w_0$  is constraint to zero in this prior work, and it decreases as the time step  $t$  increases. This is to consider the accumulated cost as the wealth level. Our framework allows the  $w_0$  to be any real value. The prior work relied on an “augmented” MDP whose states are defined as the pairs  $(s, w)$ , while our research still inherits the traditional and commonly accepted MDP with the state  $s$ . The prior work developed a functional value iteration that maintains a value function for each state of the given MDP. We incorporate the classical Bellman value iteration, with a slight modification to a utility value iteration. We solve the problem domain using a bridge algorithm.

## 5.2 Decision Theories and Utility Functions

Expected utility ( $Eu$ ) theory [31] provides a quantification for decision-making under risk and uncertainty. Let  $X = (E_1 : x_1, \dots, E_k : x_k)$  denote a prospect. The  $E_i (i = 1, \dots, k)$  denotes the possible events, of which exactly one is true and the others are not true. The  $x_i$  designates the amount of reward when the  $E_i$  is true. In the decision theory,  $X \succeq Y$  if the agent prefers to choose  $X$  from  $\{X, Y\}$ . When choosing from multiple prospects, the agent selects one that equally or more than  $\succeq$ -dominates the others. Because we do not know which event is true, we do not know for sure what outcome will result from a prospect. The expected utility can be a decision criterion. Let  $p(E_i) = p_i (i = 1, \dots, n)$  be the probability

distribution of the events, then  $Eu(X) = \sum_{i=1}^n p_i u(x_i)$ , where  $u(x_i)$  is the utility for  $E_i$ . We have

$$X \succeq Y \iff Eu(X) \geq Eu(Y),$$

i.e.

$$\sum_{i=1}^n p_i [u(x_i) - u(y_i)] \geq 0.$$

If an agent always obeys the expected utility decision criterion, it is risk consistent. certainty equivalent is a quantification of the wealth under uncertainty. Denote  $\tilde{x}$  as the certainty equivalent of the wealth  $x$ , then  $u(\tilde{x}) = u(x)$ .

Both the linear and the exponential utility functions are zero-switch. The agent with zero-switch utility functions never changes its decision according to different wealth levels. General Markov decision processes of decision-theoretic planning use linear utility functions, which is risk-neutral [28]. When the utility function takes a linear form  $u(x) = x$ , the expected utility becomes the expected value ( $E$ ) of wealth. The decision criterion becomes  $X \succeq Y$  if and only if  $E(X) \geq E(Y)$ .

In risk-sensitive MDPs [4, 84], an exponential utility function is in the form

$$u(x) = -\text{sgn}(\tau)e^{-x/\tau} \quad (5.1)$$

with inverse

$$u^{-1}(x) = -\tau \log(-\text{sgn}(\tau)x) \quad (5.2)$$

where  $\tau (\tau \neq 0)$  is the risk tolerance coefficient. The greater  $\tau$  means the stronger risk tolerance. When  $\tau > 0$ , the greater  $\tau$  results in the less risk-aversion; when  $\tau < 0$ , the larger  $\tau$  results in the greater risk-seeking.

Bell [76] proved that, only the linear-exponential utility functions satisfy the one-switch rule:

$$u(x) = x - ae^{-x/\tau} \quad (5.3)$$



where  $a > 0, \tau > 0$ . Given  $u(x) = v$ , the  $y = u^{-1}(x)$  can be achieved by solving the equation  $y - ae^{-y/\tau} - v = 0$ . There are various methods to address the solution of such an equation. We skip this part of contents.

### 5.3 Risk-Aware MDPs

An MDP models a dynamic system as a set of Markov states, with a pre-knowledge of the transition probabilities. MDP based dynamic systems assume that every state shares the same action space. Moreover, the Markov property enables us to utilize dynamic programming to find out optimal policies. Since an agent's risk attitude is determined by its utility functions, by adjusting these functions, we can model the Markov decision problems with any risk attitudes using risk-aware MDPs.

A risk-aware MDP is a tuple  $\langle S, A, T, R, s_0, G, U \rangle$ , where the  $S$  is a set of states, the  $A$  is a set of actions, the  $T(s, a, s')$  is the transition probability from the state  $s$  to  $s'$  using an action  $a$ , and the  $R(s, a)$  defines the reward when executing an action  $a$  in the state  $s$ . The  $s_0 (s_0 \in S)$  is the initial state, and the  $G (G \subset S)$  is a set of absorbing states. The  $U$  is the set of utility functions,  $U = \{u(x), u^{-1}(x)\}$ . An absorbing state does not have to be a goal. In the general case planning problems, such as an agent plans to invest an amount of money (or energy) for a gamble, the actual goal is to win the gamble. But if the agent wants to avoid the gamble due to its risk attitude, the system also reaches an absorbing state. Thus, the  $G$  includes the goals as well as the termination states (for the infinite planning without goal or termination state, we can simply set  $G = null$ ).

A policy represents the action that an agent wants to undertake in a given state. In the traditional risk-neutral MDP model, the optimal state-action mapping for the  $t^{th}$  time step, denoted as  $\pi_t$ , can be reached by the optimal  $(t - 1)$ -step value function  $V_{t-1}$ :

$$Q_t(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{t-1}(s') \quad (5.4)$$

where the  $\gamma$  is the discount factor. The optimal policies are calculated using the discounted sum of rewards  $E(\sum_t \gamma^t R(s_t, a_t))$ . The optimal policy of a state  $s$  in a time step  $t$  is obtained by

$$\begin{aligned}\pi_t(s) &= \arg_a V_t(s) \\ &= \arg \max_a Q_t(s, a)\end{aligned}$$

Continue the value iteration until it comes to the convergence condition

$$\max_s |V_t(s) - V_{t-1}(s)| < \varepsilon.$$

Then for each state  $s$ , we obtain the convergence value  $V^*(s)$ , the convergence value  $Q^*(s, a)$  for each action  $a$ , and the convergence policy  $\pi^*(s)$ .

The optimal action selection becomes a problem to find a sequence of actions  $a_{1..t}$  to maximize the expected sum of rewards  $E(\sum_t \gamma^t R(s_t, a_t))$ . In this process, what we concern is the control effect, achieved from the relative relationship of the values. When we use a discount factor  $\gamma$ , the relative relationship remains unchanged, but the values can converge to a fixed number.

To adapt to the expected utility theory, let  $\gamma = 1$ , and apply the utility function on the values, we get

$$Q_t(s, a) = u \left[ R(s, a) + u^{-1} \left( \sum_{s' \in S} T(s, a, s') V_{t-1}(s') \right) \right] \quad (5.5)$$

Equation 5.5 puts every value into a utility function, hence, we call it *utility value iteration*. Specifically, when  $u(x) = x$ , it becomes the traditional Bellman value iteration of Equation 5.4.

The same with value iteration, utility value iteration can also be solved using backward induction, using dynamic programming. The backward induction determines a sequence of optimal actions from the end of a problem. This process continues backwards

until one has determined the optimal action for every state. When the value for every state converges, the program terminates.

**Observation 1** *After the utility value iteration converges, given the state  $s$ , we have that,*

- (i) the  $V^*(s)$  is an expected utility;*
- (ii) the  $u^{-1}(Q^*(s,a))$  is a certainty equivalent of action  $a$ ;*
- (iii) the  $u^{-1}(V^*(s))$  is a certainty equivalent of the optimal action.*

Utility value iteration guarantees the convergence of linear or exponential functions [4]. Unfortunately, it does not hold for the one-switch linear and exponential functions, which are more useful in real-world risk-aware applications. However, as discussed [76], the one-switch utility functions adapt to the change of risk attitudes of the investors better than zero-switch functions.

Although the traditional work of risk-sensitive MDP [4] only addresses the use of zero-switch functions, an “augmented” MDP solution is proposed to solve the one-switch function based utility value iteration [79, 83]. However, even if such an approach can solve the one-switch utility value iteration, it requires the change of the classical MDP to an “augmented” version, simply because of the risk-attitude. More importantly, such a seemingly tiny modification increases the state space as well as the complexity of the system structure, laying unexpected burden to the planning tasks. In Section 5.5, we will introduce a bridge algorithm to address how to solve the one-switch utility value iteration. It makes use of the classical MDP, rather than an “augmented” or other modified version of MDP for the utility value iteration. By the bridge algorithm, we can achieve the goal of solving the risk-aware MDP, without changing the model of the original planning system. Thus, with the bridge algorithm, Equation 5.5 will be useful either the utility function is zero-switch or one-switch.

## 5.4 Decision-Making with Cost and Wealth

### 5.4.1 Decision Processes and Decision Graph

A decision graph [72] consists of decision nodes and action edges. We formalize the definition and correlate it with decision theories. When risk and preference is considered in decision processes, each state becomes a risk-aware decision node. The action edges indicate the transitional relationship, as well as the reward or cost. Correlating decision theories with the MDP model, a prospect of a state  $s$  corresponds to an action of  $s$ .

A well-defined decision graph has the following properties: Every decision node has one or more paths to absorbing states, and it is reachable from the root decision node. Given a decision node  $s$ , its prospect is *certain* if and only if there is only a single future state  $s'$ ,  $T(s, a, s') = 1$ . If a decision node has only a single prospect, then the node is *certain*. Given a path from a decision node to an absorbing state, if every decision node along the path is certain, then the path is certain. On the contrary, a decision node is *risky* if it has at least two paths to the absorbing states, and one is not certain.

### 5.4.2 Reward, Cost and Wealth

Reward is used in almost every research work of MDPs. Seldom there is a strict definition of this term. Normally, reward can be benefit ( $R > 0$ ), hazard ( $R < 0$ ) or even cost ( $R \leq 0$ ). Although we inherit the rough definition of reward from prior works, in this paper, we strictly define cost as the quantity of consumption, and wealth as the quantity of resource. Beside these definitions, we also mention a term of prospect. A prospect of risk-aware MDP is a state-action pair  $(s, a)$ . Each prospect has a value, which we obtain by Equation 5.5.

**Definition 1** A path  $p$  is a feasible solution from the initial state  $s_0$ , to an absorbing state  $s_g \in G$ . It is composed by a set of decision nodes and the corresponding decision edges that connect these decision nodes.

We have multiple paths from the initial state  $s_0$  to the goals  $G$ . Thus, in order to find out the optimal solution of the problem domain, it corresponds to find out the optimal path. Suppose  $P$  is a set that includes all of the feasible paths for a risk-aware MDP. Denote  $n$  is the total number of paths for the problem domain, we have  $P = \{p_1, p_2, \dots, p_n\}$ .

If we choose a decision node from every path, then we obtain at most  $m \leq n$  decision nodes. This is because different paths may share common decision nodes.

**Definition 2** *Given  $m$  decision nodes  $s_1, s_2 \dots s_m$ , each coming from a distinct path, they are equipotential iff, for  $\forall i, j \in [1, m]$ , the value of every prospect of  $s_i$  equals to the value of a corresponding prospect of  $s_j$ , i.e.  $Q^*(s_i, \cdot) \equiv Q^*(s_j, \cdot)$ .*

Reward is often defined in a flexible way to ease the research and analysis of problems. It can be defined as either positive gain or negative loss, or both of them. For example, the  $4 \times 3$  Grid [36] defines reward as benefit, while the Tiger-grid [10] defines reward as hazard. In Hallway navigation domain [10], the cost of every step is a small negative constant, but there is a big positive gain at the destination. In fact, it maps many real-world measurements into a single parameter of reward. In the robot navigation domain, the cost of every step is related with the energy consumption of the robot, whereas the positive gain may represent the object that the robot expects to capture.

In a decision system that is measured simply by reward, one assumes that, by rough estimation, benefit, hazard, or cost can be quantified as real-valued parameters. For a reward based decision process, the absorbing states are not equipotential. The absorbing states with benefit provide positive reward, while the absorbing states with hazard provide negative reward. Since the results are known, by backward induction, an agent can easily find a stochastic path to reach the absorbing state with better reward. Considering a reward based decision process, since the choices of optimal policies are originated from unequipotential decision nodes (the goals normally have better reward than others), we call it *unequipotential planning*.

Cost is a special type of reward. It indicates the quantity of consumption. Compared to reward, by which we only make a rough estimation of the values, the cost analysis is more precise to reflect the consumption. In fact, both the benefit and the consumption can be quantified as reward. However, the benefit of a process may be totally different from the consumption. For example, we try to write an article, the benefit is fame and/or money. However, the consumption is the time we input to such a process. If we quantify these different metrics into a single metric of reward, the result has to be a rough estimation. If we simply consider cost to analyze the process, it will be more reasonable.

In a cost based decision process, the absorbing states are equipotential. Let  $f(S)$  denote  $\{f(s), \forall s \in S\}$ . The reward array for the cost based decision processes satisfies

$$R(s, G) = 0 \wedge R(s, S \setminus G) < 0.$$

Since the choices of optimal policies are originated from equipotential decision nodes, we call it *equipotential planning*. A hidden presumption for the cost based decision processes is, the wealth for the  $s_0$  is zero.

Another equipotential planning is the wealth based decision process. Wealth indicates how much resource an agent possesses. In a wealth planning, the value of every decision node is the amount of wealth (time, energy, money, etc.). Wealth planning is based on the following assumption: the prospect of every decision node is related with the agent's current wealth and the reward of its prospects. In many decision-making applications, like investment and gamble, the policies are not independent of wealth. Therefore, a wealth planning reflects one's risk attitude precisely. For a simple decision graph, of which the decision nodes can be organized in a tree, i.e., there are only burst nodes (splitting paths), rather than sink nodes (converging paths, or loop), the solution for the wealth planning is a greedy forward searching in the decision tree. In complex decision problems, the deci-

sion graphs may not be organized into trees. Therefore the decision tree approach is not applicable. We will address this problem further in next section.

A comparison of reward, cost and wealth in planning is summarized in Table 5.1.

Table 5.1. Comparison of Reward, Cost and Wealth in Planning

	Reward	Cost	Wealth
Equipotential	y/n	y	y
Precision	rough	precise	precise
Presumption	$w_0 = 0$	$w_0 = 0$	n.a.
Outcome	known	unknown	unknown
Model	MDP	r.a. MDP	decision tree
Algorithm	b.i.	b.i.	f.g.

n.a.=not applicable      r.a.=risk-aware  
b.i.=backward induction      f.g.=forward greedy

## 5.5 Bridging Cost and Wealth

### 5.5.1 Problem Formulation

In risk-aware sequential decision-making that is not independent of wealth, every decision node does not simply consider reward or cost, but it also relates the risk-attitude with wealth. Many existing research works on the risk-sensitive MDP simply relies on reward or cost analysis for the optimal policies of decision nodes [4, 72, 79]. These results do not hold if the one-switch utility functions are used to reflect the planner's risk-attitude, because the optimal policies that are discovered simply by cost analysis may be different from the optimal policies when the wealth level is considered, and these backward induction approaches cannot utilize wealth analysis directly just like the decision tree approach. However, as we have discussed in previous sections, the one-switch utility functions adapt to the risk-attitudes better than zero-switch utility functions, in real-world risk-aware plan-

ning applications. Also, it is more realistic for real-world applications that risk-attitudes of the planners are not independent wealth. As a result, these traditional approaches about the risk-aware planning will not be able to find out the optimal policies in many real-world applications. For example, two investors  $A$  and  $B$  want to invest a venture. The investor  $A$  has an initial wealth of  $10k$ , while the investor  $B$  has an initial wealth of  $1000k$ . It is conceivable that the optimal policies for  $A$  and  $B$  should be different, because they have different amount of initial wealth. If we simply consider the analysis of reward, we may not find out the optimal policies that adapt to the potentially different requirements of  $A$  and  $B$ . Although recent works in [79] provide an “augmented” MDP solution for the one-switch risk-aware MDP, the solution changes the original state space of the MDP. It greatly increases the size of state space, which will result in an unexpected burden to the planning tasks. On the other hand, although decision tree is a prevalent solution for the wealth planning in investment or other financial applications, it works only in relatively simple sequential decision. It cannot solve complex sequential decision problems whose decision graphs are not trees.

The risk-aware MDP utilizes backward induction that is more powerful than the greedy forward search based decision tree. Therefore, it can solve the complex sequential decision problems. Then, how to solve the problems caused by the one-switch rules when wealth is considered not independent to the planners’ risk attitude, it becomes a key-point for such an approach to be used in more general applications. This is because, such a backward induction can only be used to analyze reward, without considering how the initial wealth will be related with the planners’ sequential decision. Thus, if we can find out an approach to solve the one-switch rules based risk-aware planning, and we accept that the planners’ risk-attitudes are not independent with wealth, we will solve the hardest and most meaningful part of the risk-ware planning tasks. It will change the current status of the wealth based planning being mainly solved by decision-tree. It will enable the



solution of more complex planning tasks. Also, we do not want to build up new models for the problem domains, simply because the reason of incorporating the one-switch utility functions. These new models, such as an “augmented” MDP, will change the state space of original domains, which means we use a different framework for the planning tasks.

Under these backgrounds, we want to develop a bridge algorithm, such that we can solve the problems in wealth based analysis using traditional reward or cost analysis based approaches. More importantly, such a bridge algorithm should be a “transparent” operation. That is, we do not have to make any change of the original models to an “augmented” or any other kinds of different models. From the perspective of planners, they still execute the planning as if such an additional computing does not exist. Right after the use of the bridge algorithm, planners will be able to obtain the optimal policies according to their risk-attitudes.

### 5.5.2 Equipotential Operations

If we simply consider cost, the risk-aware MDP becomes

$$M_c = \langle S, A, T, R | (R(G, \cdot) = 0), s_0, G, U, w_0 = 0 \rangle.$$

Otherwise, when wealth is considered, the risk-aware MDP becomes

$$M_w = \langle S, A, T, R, s_0, G, U, w_0, W \rangle,$$

where  $W(s)$  is the certainty equivalent for the state  $s$ .

Let the solution of the risk-aware MDP be  $\langle \pi^*(S), V^*(S) \rangle$ , indicating the optimal policy and the optimal value for every state. We can solve  $M_c$  using utility value iteration. As for  $M_w$ , since the initial state has equipotential prospects, we are unable to use backward induction to transfer the value to other decision nodes. This is the reason why existing research only considers reward and cost. In the following parts, we will introduce a bridge

algorithm, which is a bi-directional value iteration. With this algorithm, we are able to solve the wealth based planning problems that have complex decision graphs, which are not trees.

**Definition 3** *Two distinct risk-aware MDPs  $M_1$  and  $M_2$  are isomorphic iff,  $M_1 \cap M_2 = \langle S, A, T, R, s_0, G, U \rangle$ .*

**Definition 4** *Two distinct isomorphic risk-aware MDPs  $M_1$  and  $M_2$  are homeotypic iff,  $M_1$  and  $M_2$  have the same set of equipotential decision nodes.*

In order to understand these abstract terms, let us see some specific examples.

- If two risk-aware MDPs  $M_1$  and  $M_2$  are isomorphic, then they are used in a same domain.
- The equipotential decision nodes for the cost based decision processes  $M_c$  are absorbing states. Their values are all zeros.
- There is only a single equipotential decision node for the wealth based decision processes  $M_w$ , the initial state  $s_0$ .
- Since  $M_c$  and  $M_w$  do not have the same set of equipotential decision nodes, they are not homeotypic.
- Since,

$$\begin{aligned}
 M_c \cap M_w &= \langle S, A, T, R | (R(G, \cdot) = 0), s_0, G, U, w_0 = 0 \rangle \cap \langle S, A, T, R, s_0, G, U, w_0, W \rangle \\
 &= \langle S, A, T, R | (R(G, \cdot) = 0) \cap R, s_0, G, U, w_0 = 0 \cap w_0 (> 0) \rangle \\
 &= \langle S, A, T, R, s_0, G, U \rangle,
 \end{aligned}$$

$M_c$  and  $M_w$  are isomorphic.

**Theorem 1** *Either two distinct risk-aware MDPs  $M_1$  and  $M_2$  are isomorphic or homeotypic, they define the same problem domain, with the same risk-attitude.*

From the definition of homeotypic, if  $M_1$  and  $M_2$  are homeotypic, then they are isomorphic.

Thus we obtain  $M_1 \cap M_2 = \langle S, A, T, R, s_0, G, U \rangle$ . This conforms to the definition of an MDP

that has an initial state  $s_0$ , as well as a set of goal states  $G$ . Also,  $M_1$  and  $M_2$  share the same utility function  $U$ . This will lead to a fact that  $M_1$  and  $M_2$  define the same risk-attitude.

**Observation 2** *A cost based risk-aware MDP  $M_c$ , and a wealth based risk-aware MDP  $M_w$ , if they are isomorphic, they define the same problem domain, with the same risk-attitude.*

**Observation 3** *A cost based risk-aware MDP  $M_c$  has a set of equipotential decision nodes at the absorbing states, a wealth based risk-aware MDP  $M_w$  has a single equipotential decision node at the initial state.*

**Observation 4** *A cost based risk-aware MDP  $M_c$ , and a wealth based risk-aware MDP  $M_w$ , given that they are isomorphic, we have*

(i) *They hold the same optimal policies provided that their risk-attitudes are independent with wealth;*

(ii) *Otherwise, if their risk-attitudes are not independent with wealth, their optimal policies may be different.*

From the above discussions, we are able to solve a risk-aware MDP whose equipotential decision nodes are absorbing states, by transforming the  $M_c$  into its homeotypic model  $M_w$ .

Let  $ep(M)$  denote the equipotential decision nodes of  $M$ .

**Definition 5** *An equipotential transposition of a risk-aware MDP  $M$ , denoted as  $ET(M)$ , is a transformation from  $M$  to  $M'$ , where  $M'$  is not homeotypic with  $M$ . It holds the following properties*

$$(i) ET(ET(M)) = M;$$

$$(ii) V^*(ep(ET(M))) + V^*(ep(M)) = w_0.$$

**Observation 5** *A transformation from a cost based risk-aware MDP  $M_c$  to a wealth based risk-aware MDP  $M_w$  or vice versa is an equipotential transposition.*

For the cost analysis, we have  $w_0(\mathbf{M}) = 0$ , by equipotential transposition, we get  $w_0(ET(\mathbf{M})) > 0$ . The  $w_0(ET(\mathbf{M}))$  is the initial wealth that we assume the agent has.

**Definition 6** *An equipotential induction of risk-aware MDP  $\mathbf{M}$  is a backward induction, from its homeotypic counter-partner's equipotential decision nodes, to any other nodes in the decision graph.*

Let  $EI(\mathbf{M})$  denote the equipotential induction of  $\mathbf{M}$ . The  $EI(\mathbf{M})$  is executed from the decision nodes  $ep(ET(\mathbf{M}))$ , to any other decision nodes. The equipotential induction is used for wealth planning, i.e.  $w_0 > 0$ .

In our computing, we use equipotential induction to transfer a tuple of unequipotential risk-aware MDP into an equipotential tuple of risk-aware MDP. In other words, an equipotential induction helps to turn the measurement from wealth to cost.

**Observation 6** *When a planner's risk-attitude is independent of wealth, equipotential operations will not affect the optimal policies of a decision process, i.e.*

$$(i) \pi^*(ET(\mathbf{M})) = \pi^*(\mathbf{M});$$

$$(ii) \pi^*(EI(\mathbf{M})) = \pi^*(\mathbf{M}).$$

Now that under the assumption that the risk-attitude is independent of wealth, equipotential operations do not affect the optimal policies of a decision process, the equipotential transposition  $ET(\cdot)$  and the equipotential induction  $EI(\cdot)$  serve as important operations we use to construct a bridge between cost and wealth.

### 5.5.3 Bi-Directional Value Iteration

Denote  $uVI(\mathbf{M})$  as the utility value iteration that is taken on a tuple  $\mathbf{M}$ . For a risk-aware MDP  $\mathbf{M}_c$  that is based on cost, and a risk-aware MDP  $\mathbf{M}_w$  that is based on wealth, if we can find a bridge between  $\mathbf{M}_c$  and  $\mathbf{M}_w$ , we are able to find a solution for  $\mathbf{M}_w$ , with the help of  $\mathbf{M}_c$ . Since the solution requires taking value iteration in two directions of the equipotential decision nodes (the initial state:top, the absorbing states:bottom), the bridge

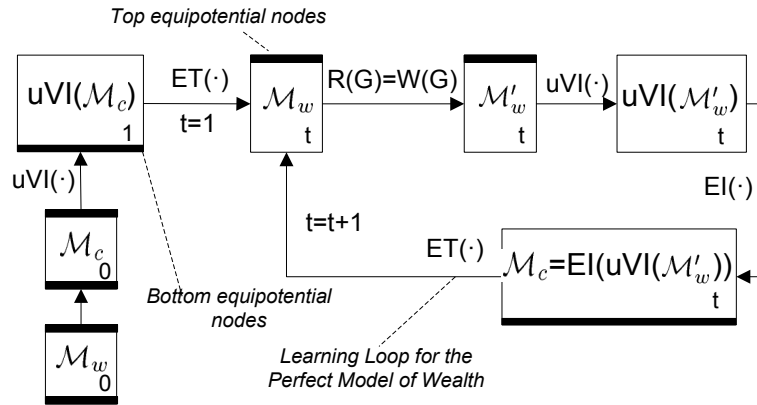


Figure 5.3. Bi-directional Value Iteration for Bridging Cost and Wealth (equipotential positions of decision nodes are visualized as bold lines).

algorithm is a *bi-directional value iteration*. A bi-directional value iteration process consists of a cost based utility value iteration, a wealth based utility value iteration, and equipotential operations (Fig. 5.3). The algorithm proceeds as follows:

1. In order to analyze a system whose risk attitude is not independent of wealth, first we assume it is independent of wealth.
2. The bi-directional value iteration process starts from the  $M_c$ .
3. By utility value iteration, we obtain  $uVI(M_{c,0})$ , with equipotential decision nodes at the bottom (the absorbing states  $G$ ).
4. Since our purpose is to achieve the  $M_w$ , we have to take an equipotential transposition,  $M_{w,t=1} = ET(uVI(M_{c,0}))$ , where  $t$  is the time-step.
5. The equipotential transposition will not make a change on the optimal policies. Let a matrix  $W$  from  $M_{w,t}$  be the current certainty equivalent of wealth.
6. Set  $R(G) = W(G)$ , we get  $M'_{w,t}$ .
7. Proceed the utility value iteration, we get  $uVI(M'_{w,t})$ , which is unequipotential.

8. We transform it into cost using equipotential induction,  $M_{c,t} = EI(uVI(M'_{w,t}))$ .
9. When we take an equipotential transposition on  $M_{c,t}$ , we get  $M_w$  for  $t + 1$  time-step, which is  $M_{w,t+1} = ET(M_{c,t})$ .
10. Continue this process until  $M_{w,t} \approx M_{w,t+1}$ , i.e. the wealth matrix  $W$  converges.

**Theorem 2** *Bi-directional value iteration algorithm guarantees the convergence of utility value iteration with the one-switch linear and exponential functions.*

From the execution process of the bi-directional value iteration algorithm, at the beginning of every external loop, we let  $R(G) = W(G)$ , by which we appoint the reward of every absorbing node as its current certainty equivalent of wealth. Although when  $t = 1$ , this value cannot reflect the exact value of certainty equivalent, it will proceed to come closer to the exact certainty equivalent after we iterate the external loop multiple times. The reason is, after  $R(G)$  is assigned, we obtain  $M'_{w,t}$ . The utility value iteration from the current estimation of  $R(G)$  will estimate again the values of other decision nodes, making them more reasonable. Then we use an equipotential induction  $EI(\cdot)$  on the result, which will reserve the difference between the decision nodes. Finally we apply an equipotential transposition  $ET(\cdot)$  to change the cost based result into a wealth based result  $M'_{w,t+1}$  of next step. Every time we do the external loop, the estimated  $W(G)$  will come closer to the exact certainty equivalent, and  $M_{w,t}$  will also be closer to  $M_{w,t+1}$ . Thus, the algorithm guarantees  $M_{w,t}$  to converge, even if we use the the one-switch linear and exponential functions in utility value iteration.

**Theorem 3** *For a risk-aware MDP whose risk-attitude is not independent of wealth, the bi-directional value iteration algorithm helps to find out the optimal policies when its iteration converges.*

The difference between the  $M_{w,t}$  and the  $M_{w,t+1}$  is only reflected in the reward matrix  $R$ . In fact, the entire iteration of  $M_{w,t}$  is a learning process for modeling a perfect  $M_w^*$ , such that

we can find the most suitable certainty equivalent wealth for every state. Since we cannot obtain the optimal result  $\langle \pi^*, V^* \rangle$  from  $M_w$  directly, if we compute it from  $M_c$ , we will obtain an approximate result  $\langle \widehat{\pi}^*, \widehat{V}^* \rangle$ . With a perfect model  $M_w^*$ , every state has the wealth, therefore, we are able to compute  $\langle \pi^*, V^* \rangle$ . Now that when the algorithm converges, the wealth of every state is already reflected in  $M_w^*$ . The current certainty equivalent matrix of the wealth  $W_t$  is already approximately the same with the matrix of the next step  $W_{t+1}$ , when the iteration converges. Therefore, the current policies are exactly the optimal policies that we want to obtain, even if we have an assumption that the risk-attitude is independent of wealth.

The above descriptions of the bi-directional value iteration algorithm are entirely based on abstract tuple operations. We will display more concrete contents using the example in next section. For equipotential operations, the equipotential transposition (Algorithm 5) is a forward iteration algorithm, and the equipotential induction (Algorithm 6) belongs to the backward induction.

This algorithm is an anytime algorithm [85]. The expected total utility of no state can decrease from one iteration step to the next, but the expected total utility of at least one state strictly increases, until the optimal state-action mapping is found in finite time. Time complexity of the bi-directional value iteration is determined by  $uVI(\cdot)$  operation, whose time complexity is  $O(|A||S|^2O(u^{-1}))$ . The learning loop iterates  $k$  times, which is determined by  $\tau$ . When  $\tau$  increases,  $k$  decreases to a small number. Thus, the time complexity is  $O(k|A||S|^2O(u^{-1})) = O(|A||S|^2)$ .

## 5.6 Example

We use the block world problem [72] as a concrete example to interpret the bi-directional value iteration algorithm. The decision graph for the block world problem

```

 $CE(S,A) = u^{-1}(Q(S,A))$ 
 $W(s_0, \cdot) = w_0$ 
repeat
  for  $\forall T(s, a, s') > 0$  do
    for  $a' \in A$  do
      if  $CE(s', a') \neq \min(CE)$  then
         $W(s', a') = W(s, a) + CE(s, a) - CE(s', a')$ 
      else if  $a' = \pi$  then
         $W(s', a') = W(s, a) + CE(s, a)$ 
      else
         $W(s', a') = 0$ 
      end if
    end for
  end for
until  $W(G) = 0$ 

```

**Algorithm 5:** Equipotential Transposition

is depicted in Figure 5.4. From the initial decision node 1, the robot has two actions, *Paint* and *Move*, with an action cost  $-3$  minutes and  $-1$  minutes respectively. A *Paint* action changes a block from black to white, or vice versa. In the decision graph, the path  $1 \rightarrow 2 \rightarrow 3$  contains only the actions of *Paint*. For the action of *Move*, it has only a probability  $0.4$  to successfully move a block on another. If it fails, the block falls on the table. As shown in the decision graph, there are 10 decision nodes, each for a state. Every edge denotes an action. If an edge splits into two edges, it implies in this prospect, there are two future states, representing success and fail. We add a logical action 3 to the state 6, because it has two *Move* actions ( $Move_1 : 6 \rightarrow 4, Move_2 : 6 \rightarrow 7$ ). Other states also need



```


$$CE(S) = u^{-1}(V(S))$$


$$CE(S,A) = u^{-1}(Q(S,A))$$


$$C(S,A) = 0$$


$$C(G,A) = CE(G,A)$$

repeat
  for  $\forall T(s,a,s') > 0$  do
    for  $s' \in S$  do
      if  $s' \in G$  then
        
$$C(s,a) = CE(s,a) - CE(s')$$

      else
        
$$C(s,a) = \max(C(s',\cdot)) + CE(s,a) - CE(s')$$

      end if
    end for
  end for
until  $C(s_0,\cdot) > 0$ 

$$CE = C$$


$$CE(S) = \max(CE(\cdot,S))$$


```

**Algorithm 6:** Equipotential Induction

to add an action, in order to keep the Markov property of the unified action space for every state. As a result, we use an action space of  $|A| = 3$ . Also, we need to add a virtual state 11 to indicate a termination for invalid actions. For example, in the state 2, there is only one action, the *Paint*. If the agent takes other actions, such as *Move*<sub>1</sub> or *Move*<sub>2</sub>, then  $T(2, Move_{1:2}, 11) = 1, R(2, Move_{1:2}) = -\infty$ . In this decision graph, only the states of 1, 6 are risky decision nodes.

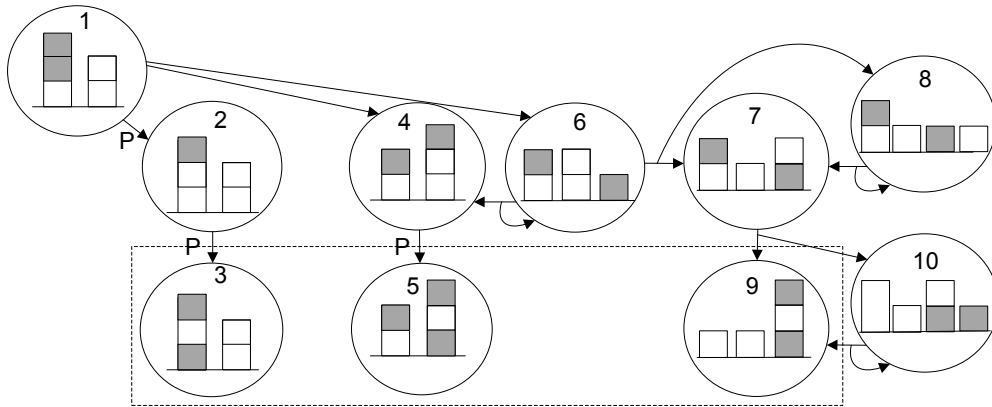


Figure 5.4. Decision Graph for the Block-World Problem.

If there is no time-constraint, i.e. the risk-attitude is independent of  $w_0$ , then the outcomes we obtain from a risk-aware MDP become the final result. Otherwise, if there exists a time-constraint, suppose it be  $w_0 = 20$  minutes. Provided the utility functions are linear or exponential, the optimal actions should be the same, with or without the  $w_0$ . However, if we choose the one-switch utility functions, the optimal actions may change after the  $w_0$  is taken into consideration. For Equation 5.1 and Equation 5.3, let  $a = 0.5$ ,  $\tau = 2$ , we get the exponential functions and the one-switch utility functions. Altogether, there are four optional plans for the block world problem. Every plan contains a sequence of policies the agent undertakes in all of 10 states (Table 5.2).

Table 5.2. Four Optional Plans for the Block-World Problem

$P_1$	1, 1, 1, 1, 1, 2, 2, 2, 1, 2
$P_2$	2, 1, 1, 1, 1, 3, 2, 2, 1, 2
$P_3$	1, 1, 1, 1, 1, 3, 2, 2, 1, 2
$P_4$	2, 1, 1, 1, 1, 2, 2, 2, 1, 2

Table 5.3 displays part of the output data of the bi-directional value iteration. The outcome of every step has the data for each state. For the case of exponential utility func-

tion, the plans keep to be  $P_1$ . This is in conformance with the fact that the exponential utility functions meet the zero-switch rule. No matter what level of wealth is taken into consideration, the plan should not change. For the one-switch utility function, the plan keeps to be  $P_3$  until it converges in  $M_{w,n}$ . This is coherent with the one-switch rule.

Process	Outcomes	Plan
<b>Exponential</b>	$(u(x) = -e^{-x/2})$	
$uVI(M_{c,0})$	-6, -3, 0, -3, 0, -11.23, -8.23, -16.46, 0, -8.23	$P_1$
$M_{w,1}$	20, 17, 14, 11.77, 8.77, 20, 11.77, 20, 3.54, 11.77	$P_1$
$uVI(M'_{w,1})$	8, 11, 14, 5.77, 8.77, -2.46, -4.7, -12.9, 3.54, -4.7	$P_1$
$M_{c,1}$	-6, -3, 0, -3, 0, -11.23, -8.23, -16.46, 0, -8.23	$P_1$
$M_{w,2}$	20, 17, 14, 11.77, 8.77, 20, 11.77, 20, 3.54, 11.77	$P_1$
$M_{w,n}$	20, 17, 14, 11.77, 8.77, 20, 11.77, 20, 3.54, 11.77	$P_1$
<b>One-Switch</b>	$(u(x) = x - 0.5e^{-x/2})$	
$uVI(M_{c,0})$	-6, -3, 0, -3, 0, -7.34, -2.96, -7.34, 0, -2.96	$P_3$
$M_{w,1}$	20, 17, 14, 15.7, 12.7, 20, 15.6, 20, 12.7, 15.6	$P_3$
$uVI(M'_{w,1})$	8, 11, 14, 9.7, 12.7, 7.65, 10.15, 7.65, 12.65, 10.2	$P_3$
$M_{c,1}$	-5.2, -3, 0, -3, 0, -5, -2.5, -5, 0, -2.5	$P_3$
$M_{w,2}$	20, 17, 14, 17.797, 14.797, 19.801, 17.298, 19.801, 14.797, 17.298	$P_3$
$M_{w,n}$	20, 17, 14, 17.799, 14.799, 19.8, 17.299, 19.8, 14.799, 17.299	$P_2$

Table 5.3. Example Data from the Bi-directional Value Iteration Process

More results are shown in Figure 5.5. We vary the  $\tau$  from 1 to 20, and record every result. The initial plans only consider the cost ( $w_0 = 0$ ), and the optimal plans consider the wealth ( $w_0 = 20$ ). The  $x$ -axis is the risk-tolerance coefficient  $\tau$ , and the  $y$ -axis is the index of the corresponding plan. For the exponential utility function, a cost based initial plan is the same with a wealth based optimal plan. However, the plan changes when  $\tau = 2$  for the decision process with one-switch utility function. This result demonstrates that, with one-switch utility functions, the optimal plan for the cost based analysis and the wealth based analysis may be different in an application system.

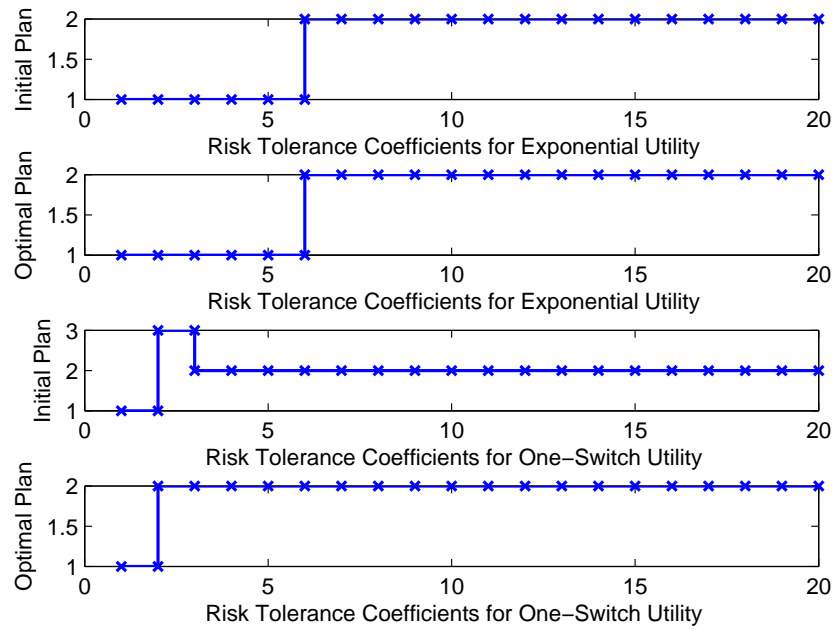


Figure 5.5. Comparing the Initial Plan ( $W_0$ ) and the Optimal Plan ( $W_0 = 20$ ): x-axis be  $\tau$ , y-axis be the index of plan.

## 5.7 Conclusion

The profound influence of the bridge algorithm between cost and wealth relies on the followings: (i) This approach enables us to relate risk attitudes of a planning with the actual wealth, instead of simply considering cost or reward; (ii) The bridge algorithm provides a powerful alternative for the wealth based decision problems, which are previously solved by decision tree.

## CHAPTER 6

### CONCLUSION AND FUTURE WORK

#### 6.1 Summary

Pervasive environments provide a challenge to the robot planning. We propose an episodic task planning and learning approach to guide robots to achieve the objective to serve the elderly and the disabled.

We first provide an introduction of MDPs, POMDPs and an MHVI algorithm to solve the POMDPs, by which we solve the following problems. In the episodic task planning and learning (ETL) approach, we propose a task-oriented design. This is a computational framework that models the tasks to serve people as MDPs and POMDPs. The tasks are further abstracted into episodes, a planner learns the experiences and then extract useful knowledge about these experiences for future planning. We made an evaluation and the ETL approach has better performance than the existing approaches of MAXQ, HEXQ and plain MDPs. More importantly, our approach aims to discover a cognitive structure and the problem domains. The planner is able to find out a clear chart to guide every step of the planning, rather than simply to decompose the abstract actions such as MAXQ. Thus, the ETL approach serves as a computational framework of episodic memory, which is more close to the thinking of humans.

We further provide an example of the tasks in assistive environments, the reminder system. In this work, we introduce a hierarchical multimodal framework for the pervasive interaction. Then, we answer another question about how to deal with risk-aware planning for an intelligent agent. For example, if the time is limited in an assistive task, how could

a robot make an optimal server for people. We provide a bi-directional value iteration algorithm for the solution.

In sum, we answer a question that, how a robot organizes its experiences to finish the tasks to assist people.

## 6.2 Future Work

A difference of the ETL approach with the episodic memory is that, the experiences in ETL are obtained from learning and optimization of global data, while episodic memory records episodes simply according in their emerging times. However, both of ETL and episodic memory require the creation of abstract state or snapshot of a scenario, with the experiences corrected to the abstractions and snapshots. Further research how to utilize ETL and episodic memory for robot planning involves a cross-sectional topic that how to correlate AI, cognitive science as well as HRI to optimize the actions of robots. One thing is determined, that the further step this research is taken, the closer the intelligence of robots is closer to humans.

In fact, even in human society, difference persons may have different performance in the processes of task handling. Generally, a boss may prefer a more efficient clerk than a slower one. In this sense, the research of how robots undertake tasks efficiently has both theoretical and realistic meanings.

## 6.3 Closing Remarks

In this dissertation, we introduce the approaches for robot planning to serve people in assistive pervasive environments, with the episodic task planning and learning as the core part. This research provides a human-like thinking for the robot to finish the designated

tasks. Such a computation framework will also be helpful for the design of cognitive robot, and be a promising part of future research of AI.

## REFERENCES

- [1] E. Tulving, *Elements of Episodic Memory*. Oxford, UK; New York: Oxford University Press, 1983.
- [2] T. G. Dietterich, “The maxq method for hierarchical reinforcement learning,” in *ICML*, San Francisco, CA, USA, 1998, pp. 118–126.
- [3] B. Hengst, “Discovering hierarchy in reinforcement learning with hexq,” in *ICML ’02: Proceedings of the Nineteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 243–250.
- [4] R. A. Howard and J. E. Matheson, “Risk-sensitive markov decision processes,” *Management Science*, vol. 18, no. 7, pp. 356–369, Mar. 1972.
- [5] D. Tecuci and B. Porter, “An Episodic Based Approach to Complex Event Processing,” in *Intelligent Event Processing, Papers from the 2009 Spring Symposium.*, O. E. Nenad Stojanovic, Andreas Abecker and A. Paschke, Eds. Menlo Park, CA: AAAI Press, 2009, technical Report SS-09-05.
- [6] M. A. Batalin and G. S. Sukhatme, “Mobile robot navigation using a sensor network,” in *IEEE International Conference on Robotics and Automation*, 2004, pp. 636–642.
- [7] M. Mühlenbrock, O. Brdiczka, D. Snowdon, and J.-L. Meunier, “Learning to detect user activity and availability from a variety of sensor data,” in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom’04)*. Washington, DC, USA: IEEE Computer Society, 2004, p. 13.
- [8] A. Mihailidis, B. Carmichael, and J. Boger, “The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home,”



- IEEE Transactions on Information Technology in Biomedicine*, vol. 8, no. 3, pp. 238–247, 2004.
- [9] T. Smith and R. G. Simmons, “Heuristic search value iteration for POMDPs,” in *Proceedings of UAI*, 2004.
- [10] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, “Learning policies for partially observable environments: scaling up,” in *Proceedings of ICML*, 1995, pp. 362–370.
- [11] J. Hoey, P. Poupart, C. Boutilier, and A. Mihailidis, “Semi-supervised learning of a pomdp model of patientcaregiver interactions,” in *Proc. IJCAI Workshop on Modeling Others from Observations*, 2005, pp. 101–110.
- [12] J. D. Williams and S. Young, “Partially observable markov decision processes for spoken dialog systems,” *Comput. Speech Lang.*, vol. 21, no. 2, pp. 393–422, 2007.
- [13] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, “Grasping POMDPs,” in *Proceedings of ICRA*, 2007, pp. 4685–4692.
- [14] M. Lekavý and P. Návrát, “Expressivity of STRIPS-like and HTN-like planning,” in *Agent and Multi-Agent Systems: Technologies and Applications, First KES International Symposium*, vol. 4496. Springer, 2007, pp. 121–130.
- [15] F. Deák, A. Kovács, J. Váncza, and T. P. Dobrowiecki, “Hierarchical knowledge-based process planning in manufacturing,” in *Proceedings of the IFIP 11 International PROLAMAT Conference on Digital Enterprise*, 2001, pp. 428–439.
- [16] J. Pineau, N. Roy, and S. Thrun, “A hierarchical approach to pomdp planning and execution,” in *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)*, 2001.
- [17] S. P. Singh and D. Cohn, “How to dynamically merge markov decision processes,” in *Proceedings of NIPS*, 1997.

- [18] A. Jonsson and A. Barto, "A causal approach to hierarchical decomposition of factored mdps," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 401–408.
- [19] M. Cirillo, L. Karlsson, and A. Saffiotti, "A human-aware robot task planner," in *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI, 2009.
- [20] A. Nuxoll and J. E. Laird, "A cognitive model of episodic memory integrated with a general cognitive architecture," in *ICCM*, 2004, pp. 220–225.
- [21] C. M. Atance and D. K. O'Neill, "The emergence of episodic future thinking in humans," *Learning and Motivation*, vol. 36, no. 2, pp. 126–144, May 2005.
- [22] E. Tulving, "Episodic and semantic memory," in *Organization of Memory*. New York: Academic Press, 1972.
- [23] E. Tulving and K. Szpunar, "Episodic memory," *Scholarpedia*, vol. 4, no. 8, p. 3332, 2009.
- [24] W. Dodd, "The design of procedural, semantic, and episodic memory systems for a cognitive robot," Nashville, Tennessee, 2005.
- [25] M. Solms and O. Turnbull, *The Brain and the Inner World*. Cathy Miller Foreign Rights Agency, London, England: Karnac/Other Press, 2002.
- [26] E. A. Feigenbaum, "The simulation of verbal learning behavior," in *IRE-AIEE-ACM '61 (Western): Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*. New York, NY, USA: ACM, 1961, pp. 121–132.
- [27] R. Howard, *Dynamic Programming and Markov Processes*. Cambridge, Mass: MIT Press, 1960.
- [28] M. Puterman, *Markov Decision Processes*. New York, NY.: John Wiley & Sons, Inc., 1994.
- [29] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

- [30] E. Sondik, “The optimal control of partially observable Markov processes,” Ph.D. dissertation, Stanford University, 1971.
- [31] J. von Neumann and O. Morgenstern, *Theory of games and economic behavior*. Princeton: Princeton University Press, 1944.
- [32] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.
- [33] R. Parr and S. Russell, “Approximating optimal policies for partially observable stochastic domains,” in *Proc. of IJCAI*, 1995, pp. 1088–1094.
- [34] X. Boyen and D. Koller, “Tractable inference for complex stochastic processes,” in *UAI*, July 24–26 1998, pp. 33–42.
- [35] S. Thrun, “Monte Carlo POMDPs,” in *Advances in Neural Information Processing 12*, 2000, pp. 1064–1070.
- [36] F. S. Melo and M. I. Ribeiro, “Transition entropy in partially observable markov decision processes,” in *Intelligent Autonomous Systems*. IOS Press, 2006, pp. 282–289.
- [37] J. Pineau, G. J. Gordon, and S. Thrun, “Point-based value iteration: An anytime algorithm for POMDPs,” in *Proc. of IJCAI*, 2003, pp. 1025–1032.
- [38] M. T. J. Spaan and N. A. Vlassis, “A point-based POMDP algorithm for robot planning,” in *Proc. of ICRA*. IEEE, 2004, pp. 2399–2404.
- [39] N. Roy and G. Gordon, “Exponential family pca for belief compression in pomdps,” in *NIPS*. MIT Press, 2003, pp. 1043–1049.
- [40] Y. Virin, G. Shani, S. E. Shimony, and R. I. Brafman, “Scaling up: Solving POMDPs through value based clustering,” in *Proc. of AAAI*, 2007, pp. 1910–1911.
- [41] J. Pineau, “Bayes-adaptive pomdps: A new perspective on the explore-exploit trade-off in partially observable domains,” in *Proceedings of International Symposium on Artificial Intelligence and Mathematics*, 2008.

- [42] J. Wolfe and S. Russell, “Exploiting belief state structure in graph search,” in *ICAPS Workshop on Planning in Games*, 2007.
- [43] K. Matusita, “On the notion of affinity of several distributions and some of its applications,” *Ann. Inst. Statist. Math.*, vol. 19, pp. 181–192, 1967.
- [44] M. T. J. Spaan and N. A. Vlassis, “Perseus: Randomized point-based value iteration for pomdps,” *J. Artif. Intell. Res. (JAIR)*, vol. 24, pp. 195–220, 2005.
- [45] T. Smith and R. G. Simmons, “Point-based POMDP algorithms: Improved analysis and implementation,” in *Proceedings of UAI*, 2005, pp. 542–547.
- [46] D. Feil-seifer and M. J. Mataric, “Defining socially assistive robotics,” in *Proc. IEEE International Conference on Rehabilitation Robotics (ICORR05)*, 2005, pp. 465–468.
- [47] O. C. Jenkins, G. González, and M. M. Loper, “Tracking human motion and actions for interactive robots,” in *HRI '07: Proceedings of the ACM/IEEE international conference on Human-robot interaction*. New York, NY, USA: ACM, 2007, pp. 365–372.
- [48] O. Maimon, “The robot task-sequencing planning problem,” *IEEE Trans. Robotics and Auto.*, vol. 6, pp. 760–765, Dec. 1990.
- [49] A. Deshpande, B. Milch, L. S. Zettlemoyer, and L. P. Kaelbling, “Learning probabilistic relational dynamics for multiple tasks,” in *Probabilistic, Logical and Relational Learning - A Further Synthesis*, vol. 07161, 2007.
- [50] F. Courtemanche, M. Najjar, B. Paccoud, and A. Mayers, “Assisting elders via dynamic multi-tasks planning: a markov decision processes based approach,” in *Ambi-Sys '08: Proceedings of the 1st international conference on Ambient media and systems*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–8.

- [51] T. Winograd, "Frame representations and the declarative procedural controversy," in *Bobrow, D.G. and Collins, A. (eds.) Representation and Understanding: Studies in Cognitive Science*. New York:Academic Press, 1975.
- [52] A. L. R. V. R. I. o. C. T. V. U. o. T. V. Deutsch, T.; Gruber, "Episodic memory for autonomous agents," in *Conference on Human System Interactions*, 2008, pp. 621–626.
- [53] A. Chateauneuf, "On the use of capacities in modeling uncertainty aversion and risk aversion," *Journal of Mathematical Economics*, vol. 20, no. 4, pp. 343–369, 1991.
- [54] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [55] D. Potts and B. Hengst, "Discovering multiple levels of a task hierarchy concurrently," *Robotics and Autonomous Systems*, vol. 49, no. 1-2, pp. 43–55, 2004.
- [56] E. A. Hansen and R. Zhou, "Synthesis of hierarchical finite-state controllers for POMDPs," in *Proceedings of ICAPS*. AAAI, 2003, pp. 113–122.
- [57] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *Proceedings of IJCAI*, 1995, pp. 1104–1113.
- [58] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.
- [59] T. Holz, M. Dragone, and G. M. P. O'Hare, "Where robots and virtual agents meet," *I. J. Social Robotics*, vol. 1, no. 1, pp. 83–93, 2009.
- [60] D. Kawanaka, T. Okatani, and K. Deguchi, "HHMM based recognition of human activity," *Transactions Institute Elec. Info. and Comm. Eng.*, vol. E89-D, no. 7, pp. 2180–2185, July 2006.

- [61] R. Kelley, A. Tavakkoli, C. King, M. Nicolescu, M. Nicolescu, and G. Bebis, “Understanding human intentions via hidden markov models in autonomous mobile robots,” in *Proc. of the 3rd ACM/IEEE international conference on Human robot interaction*, 2008, pp. 367–374.
- [62] D. L. Vail, M. M. Veloso, and J. D. Lafferty, “Conditional random fields for activity recognition,” in *Proceedings of AAMAS*, 2007, pp. 1–8.
- [63] M. E. Pollack, “Planning technology for intelligent cognitive orthotics,” in *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS), April 23-27, 2002, Toulouse, France, 2002*, p. 322.
- [64] M. Pollack, “Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment,” *AI Magazine*, vol. 26, no. 2, pp. 9–24, 2005.
- [65] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis, “A decision-theoretic approach to task assistance for persons with dementia,” in *Proceedings of IJCAI*, 2005, pp. 1293–1299.
- [66] F. Broz, I. R. Nourbakhsh, and R. G. Simmons, “Planning for human-robot interaction using time-state aggregated POMDPs,” in *Proceedings of AAAI*, 2008, pp. 1339–1344.
- [67] H.-E. Lee, Y. Kim, K.-H. Park, and Z. Bien, “Development of a steward robot for human-friendly interaction,” in *Control Applications, 2006. CCA '06. IEEE International Conference*, 2006, pp. 551–556.
- [68] S. Natarajan, P. Tadepalli, and A. Fern, “A relational hierarchical model for decision-theoretic assistance,” in *ILP'07: Proceedings of the 17th international conference on Inductive logic programming*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 175–190.

- [69] G. M. Youngblood, E. O. H. III, D. J. Cook, and L. B. Holder, "Automated hpomdp construction through data-mining techniques in the intelligent environment domain," in *FLAIRS Conference*, 2005, pp. 194–200.
- [70] A. de Palma and et al, "Risk, uncertainty and discrete choice models," *Marketing Letters*, vol. 19, no. 3, pp. 269–285, December 2008.
- [71] G. B. Dantzig, "Planning under uncertainty," *Annals of Operations Research*, vol. 84, p. Preface, 1998.
- [72] S. Koenig and R. G. Simmons, "Risk-sensitive planning with probabilistic decision graphs," in *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, 1994, pp. 363–373.
- [73] A. Pereira and R. Broed, "Methods for uncertainty and sensitivity analysis : Review and recomendations for implementation in ecolego," 2006.
- [74] V. A. Ugrinovskii, "Risk-sensitivity conditions for stochastic uncertain model validation," *Automatica*, vol. 45, no. 11, pp. 2651–2658, 2009.
- [75] P. C. Fishburn, *Foundations of Expected Utility*. Dordrecht: D. Reidel Publishing Co., 1982.
- [76] D. E. Bell, "One-switch utility functions and a measure of risk," *Manage. Sci.*, vol. 34, no. 12, pp. 1416–1424, 1988.
- [77] D. E. Bell and P. C. Fishburn, "Strong one-switch utility," *Manage. Sci.*, vol. 47, no. 4, pp. 601–604, 2001.
- [78] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, 1957.
- [79] Y. Liu and S. Koenig, "Risk-sensitive planning with one-switch utility functions: Value iteration," in *Proceedings of The Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005, pp. 993–999.

- [80] P. Perny, O. Spanjaard, and L.-X. Storme, “State space search for risk-averse agents,” in *Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI, Hyderabad, India, 2007*, pp. 2353–2358.
- [81] G. Nejat and Z. Zhang, “Finding disaster victims: Robot-assisted 3d mapping of urban search and rescue environments via landmark identification,” in *International Conference on Automation, Robotics and Computer Vision*, 2006, pp. 1–6.
- [82] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, “Planning under time constraints in stochastic domains,” *Artif. Intell.*, vol. 76, no. 1-2, pp. 35–74, 1995.
- [83] Y. Liu and S. Koenig, “Functional value iteration for decision-theoretic planning with general utility functions,” in *Proceedings of AAAI*. AAAI Press, 2006, pp. 1186–1186.
- [84] S. Marcus, E. Fernández-Gaucherand, D. Hernández-Hernandez, S. Coraluppi, and P. Fard, *Systems and Control in the Twenty-First Century*, 1997, vol. 29, ch. Risk sensitive Markov decision processes.
- [85] M. S. Boddy and T. Dean, “Solving time-dependent planning problems,” in *IJCAI*, 1989, pp. 979–984.



## BIOGRAPHICAL STATEMENT

Yong Lin received his Bachelor of Engineering degree from Shenyang Jianzhu University, China, in 1997. He received his Master of Science degree from University of Science and Technology of China, in 2003. All of these degrees are in Computer Science and Engineering. He has worked on the areas of computer science, including network management, wireless network, data mining, security and cryptography. His current research interests include pervasive computing, robotics, artificial intelligence, machine learning and game theory.