# CONTEXT-AWARE MARKOV DECISION PROCESSES

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

BY

ÖMER EKMEKCİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

**CONTEXT-AWARE MARKOV DECISION PROCESSES**

submitted by **ÖMER EKMEKCİ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

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

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

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

**Examining Committee Members:**

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

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

Prof. Dr. Fatoş T. Yarman Vural
Computer Engineering Department, METU _____

Prof. Dr. Ahmet Cosar
Computer Engineering Department, METU _____

Assist. Prof. Dr. Mehmet Tan
Computer Engineering Department, TOBB ETÜ _____

**Date:** _____

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

Name, Last Name: ÖMER EKMEKCİ

Signature :

# ABSTRACT

CONTEXT-AWARE MARKOV DECISION PROCESSES

EKMEKCİ, Ömer
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Faruk Polat

September 2014, 53 pages

In the 1990s, when artificial intelligence (AI) research become an important area again, researchers quitted trying to solve the ultimate problem, generating autonomous agents with "human-level intelligence". Currently, significantly important part of the research is highly focused on developing autonomous agents particularly dedicated to solve problems only in a chosen domain. Even though models and algorithms provided in reinforcement learning (RL), such as Markov Decision Processes (MDP), are successful at efficiently determining optimal or near optimal policies for these problems, however, these tools, originated from operations research and not customized particularly for AI, ignore using the available information in a given problem which indeed makes them far away from being a suitable model for AI. This leads unstructured representation of that problem which makes these tools significantly less efficient at determining a useful policy as the state space of a task grows, which is the case for more realistic problems.A milestone will be achieved in AI if new state machines are invented that use the information in a given task enabling generate optimal or near-optimal policies up to more realistic tasks having large number of states.If this is successfully achieved, the research will be one step closer to fulfill the ultimate aim of AI. Based on this motivation, this thesis presents a new state machine, based on MDP, for representing and solving more realistic AI problems which is entitled "Context-Aware Markov Decision Process (CA-MDP)". For that matter, CA-MDP,

in comparison to MDP, introduces information based on causal relationship of actions and states therefore enabling compact represention of the tasks and computation of an optimal policy much more efficiently, even for problems having very large number of states that MDP fails to solve efficiently which will make it an important step in integrating the information available to both representation and solution of an AI problem. After a theoretical introduction of the new state machine, an analysis is carried out. Finally, the effectiveness of the model is demonstrated experimentally, concluding with a comparison to existing models.

Keywords: Artificial Intelligence, Artificial General Intelligence, Reinforcement Learning, Markov Decision Process, Planning Domain Definition Language, Probabilistic Planning Domain Definition Language, Planning

# ÖZ

İÇERİKTEN-HABER MARKOV KARAR SÜREÇLERİ

EKMEKCİ, Ömer
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Faruk Polat

Eylül 2014 , 53 sayfa

1990'lı yıllarda yapay zeka (YZ) tekrar önemli bir araştırma alanı olunca, araştırmacılar "nihai problemi", insan seviyesinde zekaya sahip olan otonom vekiller geliştirme problemini, çözmeye çalışmaktan vazgeçmişlerdir. Günümüzde, araştırmanın çok önemli bir kısmı, seçilen belli bir alandaki problemleri çözmeye özelleşmiş otonom vekiller geliştirmeye odaklanmıştır. Her ne kadar ödüllü öğrenmende (ÖÖ) mevcut olan model ve algoritmaları, Markov Karar Süreçleri (MKS) gibi, bu tip özelleşmiş problemler için optimal veya yakın-optimal poliçeler üretmekte başarılı olsalar da, ne yazık ki, yöneylem araştırması alanından çıkmış olan ve YZ ye özelleşmiş olmayan bu araçlar, verilen bir problemin içinde kullanılmaya elverişli olan bilgiyi kullanmayı ihmal ederler, bu da onları YZ'ye uygun bir model olmaktan aslında çok uzaklaştırır. Bunlar problemin yapısal olmayan şekilde tarifine neden olup, bu da bu araçların durum uzayı genişledikçe, daha gerçekçi problemlerde olduğu gibi, kullanılabilir bir poliçe üretmelerinde daha az efektif olmalarına yol açar. Eğer daha büyük sayıda durumu olan daha gerçek hayata yakın problemlerin içindeki bilgiyi kullanıp onları çözmeyi ve optimal veya yakın-optimal poliçeler bulmayı imkanlı kılmasını sağlayacak yeni durum makinaları bulunursa YZ konusunda bir kilometre taşı geçilmiş olacak. Eğer bu başarıyla yapılabilirse, araştırmalar YZ'nin nihai amacının yerine getirilmesinde bir adım daha yaklaşmış olacak. Bu güdüden hareketle bu tez, daha

gerçekçi YZ problemlerini sunmak ve onlara çözümler üretmek için MKS temelli "İçerikten-Haber Markov Karar Süreci (İH-MKS)" isimli yeni bir durum makinası sunmaktadır. Bu sebeple, İH-MKS, MKS'ye kıyasla, MKS'nin verimli çözmekte başarısız olduğu çok büyük sayıda durumları olan problemler için bile, görevlerin kompakt sunumu ve optimal poliçeyi daha verimli hesaplamak için, bilgi olarak durum ve hareketlerin nedensel ilişkilerini ileri sürer, bu da onun bir YZ probleminde kullanışlı olan bilgiyi onun tarifi ve çözümüne ilave edilmesi için önemli bir adım olmasını sağlayacaktır.Yeni durum makinamızın kuramsal bir tanıtımından sonra, analizleri yapılacaktır. En sonunda, modelimizin etkililiği, diğer modellerle karşılaştırıp deneysel olarak sonuca varılarak, sunulacaktır.

Anahtar Kelimeler: Yapay Zeka, Yapay Genel Zeka, Ödüllü Öğrenme, Markov Karar Süreci, Planlama Alan Tanımı Dili, Olasılıksal Planlama Alan Tanımı Dili, Planlama

*To my father,*

*may he rest in peace and come alive in my theories and melodies and*

*in the very body of the magical flowers in the mountains and oasis of Nallıhan*

*in every spring*

# ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Dr. Faruk Polat and Prof. Dr. Fatos T. Yarman Vural for their constant support, guidance and friendship. It was a great honor to work under supervision of them for many since my undergraduate years. They vastly influenced my perspective of science and the very world itself. I also would like to thank Prof. Dr. İsmail Hakkı Toroslu and Assoc. Prof Dr. Tolga Can for their precious support and guidance. I also thank my thesis committee members Prof. Dr. Ahmet Coşar and Assoc. Prof. Dr. Mehmet Tan for their valuable commentaries and suggestions.

Although I am sincerely very grateful to my every friend that prove him/herself to be helpful during my studies, yet it is not quite possible for me to show my gratitude to them name by name due to the risk of overlooking. However, I would like to write my kind words for my dear friends who helped the most. I would like to thank Volkan Şirin who is as much passionate about science as me and shares the similar crazy science ideas with me, helped me to enhance my perspective and knowledge in every topic of science. Also, I would like to thank Elvan Gülen countless times for her every kind of support and help. I am greatly indebted for Dr. Utku Erdoğdu's time and effort for supplying resources while I was in the process development of my theories. Finally, I thank Murat Öztürk for his psychological support all the time.

Last of all and most definitely the most important of all, I am very fortunate to have my family. I owe my intelligence and talents to my mother who taught me to read and do math when I was only a toddler, and my father who showed me every aspects of computers, music and everything else in life. Science and music would only be a dream without them...

# TABLE OF CONTENTS

# LIST OF TABLES

TABLES

# LIST OF FIGURES

FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| CA-MDP | Context-Aware Markov Decision Process |
| AI | Artificial Intelligence |
| AGI | Artificial General Intelligence |
| RL | Reinforcement Learning |
| MDP | Markov Decision Process |
| PDDL | Planning Domain Definition Language |
| PPDDL | Probabilistic Planning Domain Definition Language |
| POMDP | Partially Observable Markov Decision Process |
| DBN | Dynamic Bayesian Network |
| DP | Dynamic Programming |
| CA-VI | Context-Aware Value Iteration |

# CHAPTER 1

# INTRODUCTION

In the first half of the 20th century, many hypotheses proposed for measurement of machine intelligence which leaded artificial intelligence (AI) research field become a very important research area. At the very beginning it started as a very promising research area sparkling with serious developments. For example, in the 1950, a detailed analysis of autonomous chess playing were published [20], first AI computer programs were developed; Samuel's checkers playing program [23] and Allen Newell, J.C. Shaw and Herbert A. Simon's General Problem Solver [17], LISP programming language was invented by John McCarthy, in the 1960s Ray Solomonoff proposed the fundamentals of mathematical theory of AI, the first symbolic integration program SAINT is written by James Slagle [21], an interactive autonomous English dialogue program ELIZA is built by Joseph Weizenbaum, in the 1970s Patrick Winston generated the program called ARCH for demonstrating learning of concepts from examples in the world of children's blocks, an automated planner called STRIPS is developed by Richard Fikes and Nils Nilsson [8], in the 1980s Marvin Minsky proposed his cognitive theory stating that humanf mind is a collection of "mindless" agents for the purpose of presenting a whole new perspective to AI in the book entitled "Society of Mind" [16], an automated planning and scheduling system called ADL is proposed by Pednault [19], in the 1990s web crawlers and other data and information extraction agents were become popular, two world champions in the fields of chess and checkers, Garry Kasparov and Tinsley are defeated by autonomous agents and Gerry Tesauro's backgammon program proved itself to be powerful enough to compete world class players [24]. Nevertheless, despite all of the progress from 1950 to 2000 AI field faced disappointment and criticism leading to cancellation of fund-

ings for the research. Beginning from the 1990s due to the increasing popularity of statistical learning methods for information processing like Neural Networks [2] and Support Vector Machines [4] and the emergence of Reinforcement Learning [23] and PDDL [9, 10] in the planning domain, AI reconstituted from its very roots, again become a hot topic. Since then, however, researcher community of AI has never followed the original "achieving human-level intelligence" goal. As Marvin Minsky puts it in 1996 : "*No one has tried to make thinking machine. The bottom line is that we really haven't progressed too far toward a truly intelligent machine. We have collections of dumb specialists in small domains. The true majesty of general intelligence still awaits our attack. We've got to get back to the deepest questions of AI and general intelligence and quit wasting time on little projects that don't contribute to the main goal.* ".

The direction is, indeed, steered to develop autonomous "intelligent" softwares dedicated to solve particular, generally highly constrained, problems in various fields rather than trying to achieve the ultimate goal of AI. These dedicated autonomous agents never benefit from the both of the vital concepts, statistical information processing and reinforcement learning. Based on this key observation, this thesis aims to build a model which is based on a model in reinforcement learning benefitting from information processing approaches in a fundamental way which will take an important step towards more *general intelligences*, i.e. autonomous agents capable of solving problems in any kind of domain, which will eventually lead to *Artificial General Intelligence* (AGI), i.e. agents capable of performing any intellectual task that a human can, namely "human level intelligence".

Reinforcement learning aims to present models and algorithms to solve both stochastic and deterministic decision making problems, generally modeled as Markov Decision Processes (MDP) [1] if environment is fully observable in contrast to Partially Observable Markov Decision Processes (POMDP) [13] for partially observable environments. MDP uses state, action, transition and reward information to represent a given problem and uses dynamic programming (DP) oriented algorithms to generate an optimal policy for the agent by propagating immediate and future rewards along given state transitions. Several of these algorithms can guarantee generating an optimal or near-optimal policy for an MDP problem given enough time and space.

However, an important downside is they ignore using the available information which leads to an unstructured representation of the problem and inefficiently generation of optimal policies when the state number grows large, as the case in real world tasks. Many methods like function approximation, state abstraction and hierarchical decomposition are proposed to overcome this problem, yet neither of them tried to use the extra information available that are not included by the MDP or other RL algorithms, since MDP is originally developed from general dynamical systems not specifically for AI. As a conclusion, AI problems model real world and contain lots of useful information, without processing of information, the true nature of task will not be revealed hence definitely inefficiency occurs in both representation and solution.

Based on these motivations mentioned, the aim for this thesis is to build a model that will also use the extra information, if any, in the environment, which will definitely help developing an autonomous agent to deal with problems from different domains, i.e. with general intelligence and also alleviate the curse of dimensonality occuring from the number of states to solve real worl tasks efficiently. In this particular study, we will only focus on the dependencies between the states and actions as the available information in the environment. Inspired from PDDL and its probabilistic extension PPDDL [26], conditions and effects of actions will be introduced to MDP, hence getting a new model called Context-Aware Markov Decision Process (CA-MDP) which will be a more suitable model than MDP for AI and easier to solve than PPDDL since DP oriented algorithms will be proposed for solving. We hope, a task with CA-MDP formulation will have states and actions as localized structures therefore making the problem suitable for decomposition. For a DP oriented solution algorithm rather than considering the whole structure, consideration of only the admissible actions and their effect area will definitely lead to faster updates hence faster convergence and determination of an optimal policy.

## 1.1 Organization of this Thesis

- Chapter 2: In this chapter, background knowledge, needed to be known to understand this thesis, is given combined with literature survey detailing the previous approaches to the problem if any.

- Chapter 3: The core theory of this work is presented. Context-Aware Markov Decision Process (CA-MDP) is introduced in a detailed manner. After, two-way conversion between MDP and CA-MDP is presented, demonstrating the expressiveness power of CA-MDP over regular MDP.

- Chapter 4: A modified version of the DP-oriented algorithm, Value Iteration, is introduced for CA-MDP, namely Context-Aware Value Iteration (CA-VI). The weaknesses of the VI algorithm are stated and modifications to remove those inefficiencies are expressed in a detaled manner. Then, a sample implementation is provided for the CA-VI algorithm. Finally, an analysis is carried out for showing that CA-VI handles computations faster than VI and generates an optimal policy.

- Chapter 5: In this particular chapter, conclusions are drawn and some of the future work is pointed out.

# CHAPTER 2

# BACKGROUND AND LITERATURE SURVEY

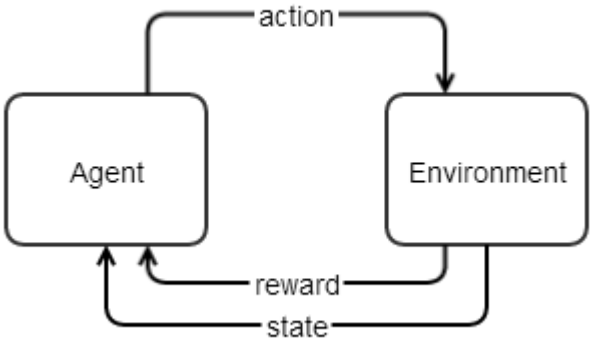
## 2.1 Reinforcement Learning

Unlike supervised and semi-supervised learning, reinforcement learning provides tools for autonomous agents to learn optimal action selection strategies without the explicit training data necessity. The agent is not provided with training data to learn the best actions to choose rather provided rewards according to their actions in a given state. It has to discover which actions give the most expected reward. In order to determine which of these actions yield the most expected reward, noting that no explicit training data is provided, the agent has to try new actions which have not been tried before. Since these are stochastic tasks, diverse set of actions should be tried over and over again for reliable estimation of expected reward for individual actions. Then the action(s) with the highest expected reward is(are) favored in the corresponding states.

A typical reinforcement learning has a number of key elements; reward function, value function and a policy. Before explaining these concepts, the model of a reinforcement learning problem should be presented. The model of the interaction between the agent and the environment is depicted in Figure 2.1;

This interaction summarizes the flow of events in a typical reinforcement learning problem. The agent starts in a state, let's denote it as $s_0$, then it needs to take an action, $a_0$, which in turn gets a reward according to the defined reward function, be that value $r_0$, and current state $s_0$ is changed to next state, let's denote it as $s_1$. The same action-reward-state transition sequence continues for the upcoming steps. The goal is to maximize the expected reward by first exploring the environment i.e.

**Figure 2.1** Agent-Environment model




learning by reward and exploiting it to maximize the reward accordingly.

These rewards are drawn from a reward function which maps the states to numerical values, i.e. rewards, where higher values indicating high desirability of the state. The total amount of expected reward, accumulated from the different explorations, i.e. succeeding transitions, starting from a state, is indicated by a function called value function. After some number of different iterations such that the value function is converged, the policy is generated. A policy defines an agent's action selection strategies in a given state, more formally a policy maps states to actions. In addition, policies may be stochastic which selects an action $a \in A$ in state $s \in S$ with probabilty $\pi(s, a)$ such that $\forall s \in S, \sum_{a \in A} \pi(s, a) = 1$. There can be many policies leading different actions in different states, the aim is to find the policy that maximizes the accumulated reward.

Having summarized the very fundamentals of reinforcement learning and important concept will be introduced next.

## 2.2 Markov Decision Processes

A finite Markov decision process [1], or MDP is a tuple ;

$$M =< S; A; P; R >, \tag{2.1}$$

where $S$ is a finite set of states, $A$ is a finite set of actions, $P$ is a transition probability function, and $R$ is an expected reward function.

An agent being in state $s \in S$ and executing an action $a \in A$ will cause state transition from $s$ to state $s' \in S$ with probability $P(s'|s,a)$ and provide the agent with a reward R(s,a). $P$ is a proper probability distribution over state-action pairs such that $(s,a) \in S \times A, \sum_{s' \in S} P(s'|s,a) = 1$. The goal is to find a policy $\pi : S \rightarrow A$ that maximizes the reward.

To find the policy maximizing the accumulated reward over states, we need to define a value function $V^{\pi}(s)$ to be the value of policy starting at state $s$. To calculate the expected reward we can write;

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} ... | s_t = s], \tag{2.2}$$

where $\gamma \in [0,1)$ is the discount factor over the values of the future states. If $\gamma = 0$, values of the future states are not considered in computation whereas if $\gamma = 1$, values of all future states are taken into consideration equally. Formally, the goal is to find a policy $\pi$ that maximizes $V^{\pi}(s), \forall s$.

## 2.3 Evaluating and Finding an Optimal Policy

For the purpose of finding an optimal policy, firstly, we need to evaluate the value function given at equation 2.2.

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} ... | s_t = s], \\ &= E_{\pi}[r_{t+1} | s_t = s] + \gamma E_{\pi}[r_{t+2} + \gamma r_{t+3} ... | s_t = s], \\ &= E_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t], \end{aligned} \tag{2.3}$$

In the last derivation of the equation 2.3, if we put the transition probability function

$P(s'|s,a)$ and the reward function $R(s,a)$ we get;

$$V^{\pi}(s) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s'). \tag{2.4}$$

Equation 2.4 is called the Bellman equation. If $\pi$ was a stochastic policy, the equation becomes;

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s') \right]. \tag{2.5}$$

The optimal value function $V^*$ is the solution to the following Bellman optimality equation [1],

$$V^*(s) = \max_{a \in A} \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s') \right]. \tag{2.6}$$

Hence, optimal policy $\pi^*$ is the policy having $V^*$ as its value function.

To compute the optimal policy, DP-oriented algorithms can be used. *Value iteration* is a DP-oriented algorithm that maintains a value table consisting of target value for each state, which is indeed an estimate of $V^*$ [1]. At every step, the algorithm successively iterates by updating the values according to the following equation ;

$$V_{k+1}(s) = \max_{a \in A} \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]. \tag{2.7}$$

The algorithm continues to iterate until a stopping criteria is met. The algorithm is given in algorithm 1 ;

After the algorithm is converged, a deterministic policy $\pi$ needs to be generated according to the equation 2.8 ;

$$\pi(s) = \max_{a \in A} \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]. \tag{2.8}$$

**Algorithm 1** Value Iteration

Initialize V(s) randomly,
**repeat**
  $\delta \leftarrow 0$
  **for all** $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) = \max_{a \in A} \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]$
    $\delta \leftarrow \max(\delta, |V(s) - v|)$
  **end for**
**until** $|\delta| < \varepsilon$

## 2.4 Scaling of Markov Decision Processes

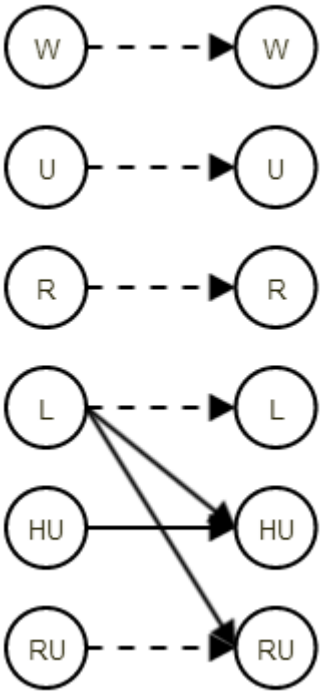
In this section, several important studies regarding scaling problem of solution of MDP problems are given.

### 2.4.1 Dynamic Bayesian Network Model for compactly representing MDPs

In MDP, space complexity becomes very high to handle as the size of state space grows relevant to a problem which makes it impossible to provide a $|S|x|S|$ probability matrix for each action to express the state transitions, and a $|S|$ vector of expected rewards. For that matter, a model for factored repesentation of MDPs and an algorithm for optimal policy construction which exploits the given model are proposed [3], based on Dynamic Bayesian Networks [DBN] which is used for modeling changes in state variables in a stochastic temporal process [6].

Their methodology for representing state transitions in MDP defines DBNs for each action. These DBNs represent the effects of actions on state variables and have two-slice structure due to the Markov assumption. The set of state variables in the first slice represents the current state of the envrionment whereas the set in the second slice represents the world after the action is applied and the edges in between represents the causality.

**Figure 2.2** DBN for action "deliver coffee"




As an example, the notion of using DBN structure in a given problem can be represented in the coffee task, for full specification refer to [3]. Figure 2.2 illustrates the DBN for the "deliver coffee" action for the coffee task. Edges represent the causalities, i.e. which state variables affect which others after taking a particular action, dashed lines specify the state variable not affected after executing the action. The state variables in this task are listed below:

- $W$ is for whether the robot is wet or not,
- $U$ is for whether robot has umbrella or not,
- $R$ is for whether it is raining or not,
- $L$ is for location of robot; office or coffee shop,
- $HU$ is for whether user has coffee or not,
- $RU$ is for whether robot has coffee or not.

Each of these states has a conditional probability distribution table for determining the

resulting value of the state variable associated with the given action which expresses the state transitions in a more compact way. According to the network for the action being demonstrated, the value of the state variable $HU$ is only affected by $L$, $HU$ and $RU$ for the action "deliver coffee", the rest has no effect.

### 2.4.2 State Abstraction and Hierarchical Decomposition

Several important studies have been done for the scalability problem of MDPs with large number of states. Some of these studies define sub-task routines that will lead to decomposition of a task and/or the policies generated for these sub-tasks will be re-used wherever the execution of the same subtask needs to be executed. In addition, with this higher level abstraction autonomous agents can solve the whole task more efficiently. There are three different major approaches defining these sub-task routines :

- Hierarchical Abstract Machines, HAMs [18]
- Options [22]
- MAXQ [7] .

The notion of *hierarchical decomposition* builds on the ideas of the identification and execution of sub-task routines and *state abstraction*, which Although, re-execution of those subroutines are pretty straightforward whereas the identification is another issue. There are several algorithms proposed to identify useful sub-task routines ( [14, 15, 25]). On the other hand, state abstraction is the abstraction of state space of a task by clustering them into certain groups being different on irrelevant information which these these clusters can be regarded as a single state. As a result, the number of computations for generation of a policy are alleviated, whereas the optimality point for a policy can be still reached under certain conditions [5].

One of the most important studies up to date is the Variable Influence Structure Analysis, (VISA), algorithm [12]. The VISA algorithm is a hierarchical decomposition algorithm for factored MDPs with DBNs [3]. It uses the causal dependency information between states gathered from the DBN model such that when an action is taken

then the value of the state in the next time step depends on the values of states having edges linking to that state in the DBN model for that executed action. Then VISA, checks for strongly connected components in the state structure of a given problem to get rid of these cycles for constructing a graph with higher level abstraction. Then, it uses the idea of exits [11], discovering the inter-state relationships that change in values of some states causes to change another particular state, for the purpose of introducing the subroutines which will solve the subtask defined by the exit state(s). The generated causal graph is then used to define the necessary states and subroutines to solve the subtask defined. Hence, as a result the algorithm generates a hierarchy of subroutines representing a solution to the whole task allevating the computational complexity.

### 2.5 Planning Domain Definition Language (PDDL)

PDDL is a language for defining non-stochastic planning domains and problems [9, 10]. A typical PDDL planning domain consists of ;

- a set $T$ of types,
- a subtyping relation $ST \subset TxT$,
- a set $C$ of global objects,
- a set $P$ of predicates,
- a set $F$ of functions,
- a set $AS$ of action schemata.

An example domain is presented in Figure 2.3, named *antivirus-domain*. The elements regarding PDDL is explained in the following subsections. For detailed specifications of PDDL, including possibles values for *requirements* tag which is also being presented in the example, refer to [9, 10].

**Figure 2.3** Definition of a typical domain in PDDL : *antivirus-domain*

```
(define (domain antivirus-domain)
 (:requirements :typing :equality :conditional-effects  :fluents
    :negative-preconditions)
 (:types tool robot bench)
 (:constants t7 - robot)
 (:predicates (onBench ?x - robot) (holding ?x - tool) (hasVirus ?x - robot))
 (:functions  (fuel-level ?x - robot) (robot-count ?x - bench))
 (:action mountRobotOnWorkbench
        :parameters (?x - robot ?y - bench)
        :precondition (< (robot-count ?y) 1)
        :effect (and (onBench ?x) (increase (robot-count ?y) 1)))

 (:action dismountRobotFromWorkbench
        :parameters (?x - robot)
        :precondition (onBench ?x)
        :effect (and (not (onBench ?x)) (decrease (robot-count ?y) 1)))


 (:action program
        :parameters (?x - robot ?y - tool)
        :precondition (and (holding ?y) (onBench ?x))
        :effect (when (hasVirus ?x)
                    (not (hasVirus ?x))) )

 (:action refuel
         :parameters (?x - robot ?y - tool)
         :precondition (and (holding ?y) (< (fuel-level ?x) 10))
         :effect (assign (fuel-level ?x) 1) )

 (:action getTool
        :parameters (?x - tool)
        :precondition (not (holding ?X))
        :effect (and (forall (?y - tool)
                     (when (holding ?y)
                         (not (holding ?y))))
                   (holding ?x))) )
```

### 2.5.1 Types

In PDDL, types are for representing the objects and variables which all have some type $t \in T$. The domain definition in Figure 2.3 has *tool, robot, bench* as types which are called *simple types*. All of the types declared in a given domain are $subtype$ of the built-in type $object$ in PDDL. The subtyping relation is *reflexive* and *transitive*. Also, PDDL allows to define a type, $t$, such that $t = \bigcup_{i=1}^{n} \tau_i$ where $n \in Z^+ \wedge n > 1$ and each of the $\tau_i$ is a simple type, which is called a *union type*.

### 2.5.2 Predicates and Functions

In PDDL, predicates are a mapping from PDDL objects to boolean state variables, whereas functions have numeric state variables as range. It is possible to restrict the domain of a function or predicate. Provided example in the Figure 2.3, has the following predicate *onBench* which restricts the domain to the type *robot*.

### 2.5.3 Actions

Actions in PDDL are for determining state transitions, i.e. altering the assignments to a subset of state variables. An action is comprised of preconditions which dictates an assignment to a subset of state variables for making the action applicable and an effect which updates the state variables. The action schemas $refuel$ and $getTool$, from the *antivirus-domain* domain, in the Figure 2.4, has preconditions $precondition\ (and\ (holding\ ?y)\ (<\ \ (fuel\text{-}level\ ?x)\ 10))$ and $precondition\ (not\ (\ holding\ ?X))$ respectively. The $refuel$ action has the effect that $fuel - level$ is settled to 1, whereas $getTool$ action has the effect that all of the $tool$s except the one given as argument are not being hold anymore.

### 2.5.4 Problems

A typical problem in PDDL consists of ;

- a set $V$ of state variables,

**Figure 2.4** $refuel$ and $getTool$ actions from the *antivirus-domain* domain

```
(:action refuel
        :parameters (?x - robot ?y - tool)
        :precondition (and (holding ?y) (< (fuel-level ?x) 10))
        :effect (assign (fuel-level ?x) 1) )

(:action getTool
       :parameters (?x - tool)
       :precondition (not (holding ?X))
       :effect (and (forall (?y - tool)
                    (when (holding ?y)
                        (not (holding ?y))))
                   (holding ?x))) )
```

- a set $A$ of actions,
- an initial state, $s_{initial}$,
- a goal state, $\phi$,
- an optimization metric, $f$.

Every problem is associated with a domain definition in PDDL. The problem definition presents the specific objects, $O$, to be used in the domain. The set of state variables $V$ are gathered from $O$, $C$, $P$ and $F$. Figure 2.5 illustrates a typical problem in PDDL associated with the domain presented in the Figure 2.3. And the set $V$ of variables is presented in the Figure 2.6.

### 2.5.5 Probabilistic Extension of PDDL : PPDDL

In 2004, an important extension to PDDL is proposed, namely *Probabilistic Planning Domain Decision Language : PPDDL*, which allowed to specify MDPs [26]. In this extension, *probabilistic effects* are introduced, in addition *rewards* are implemented by the existing PDDL feature : *fluents* [9].

**Figure 2.5** Definition of a problem, named *test-antivirus*, associated with the *antivirus-domain* domain

```
(define (problem test-antivirus)
    (:domain antivirus-domain)
    (:objects optimus - robot megatron - robot workbench - bench
        fuelprobe - tool programmingport - tool)
    (:init (holding programmingport)
        (hasVirus megatron)
        (= (fuel-level optimus) 0)
        (= (fuel-level megatron) 0)
        (= (robot-count workbench) 0))
    (:goal (and (not (hasVirus optimus)) (not (hasVirus megatron))
            (>= (fuel-level optimus) 9) (>= (fuel-level megatron) 9))) )
```

**Figure 2.6** State variables for the problem *test-antivirus*

| Name | Type | Init |
|---|---|---|
| onBench {optimus} | boolean | false |
| onBench {megatron} | boolean | false |
| holding {programmingport} | boolean | true |
| holding {fuelprobe} | boolean | false |
| hasVirus {megatron} | boolean | true |
| hasVirus {optimus} | boolean | false |
| fuel-level {optimus} | numeric | 0 |
| fuel-level {megatron} | numeric | 0 |
| robot-count {workbench} | numeric | 0 |

### 2.5.6 Probabilistic Effects

The syntax for the probabilistic effects is given below:

$$(probabilistic\ p_1\ e_1\ p_2\ e_2\ ...\ p_k\ e_k), \tag{2.9}$$

where $e_i$ denotes the effects whereas $p_i$ denotes the proabilities of the effects where $\forall i, p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$. Any probabilistic effect should state a set of an exhaustive set of probability-weighted outcomes. However, the probabilistic effects can be also expressed in the following form $(probabilistic\ p_1\ e_1\ p_2\ e_2\ ...\ p_l\ e_l)$, where

$\forall i, p_i \geq 0$ and $\sum_{i=1}^{l} p_i < 1$ which happens to mean that effect $e_i$ may happen with probability $p_i$ and no effect occurs (state remains unchanged) with probability $q = 1 - \sum_{i=1}^{l} p_i$. Figure 2.7 and 2.8 illustrate the action modified with probabilistic effects for the *antivirus-domain* domain given in the Figure 2.3 and the problem associated with this domain being extended with the newly generated probabilistic action, respectively. Note that, the requirements flag $:probabilistic-effects$ states that probabilistic effects are used in the domain definition. For full specifications, refer to [26].

**Figure 2.7** PPDDL encoding of the *program* action extended with probabilistic effect

```
(:action program
        :parameters (?x - robot ?y - tool)
        :precondition (and (holding ?y) (onBench ?x))
        :effect (when (hasVirus ?x)
                      (probabilistic 0.9 (not (hasVirus ?x)))) )
```

**Figure 2.8** PPDDL encoding of the initial conditions *test-antivirus* problem extended with probabilistic initial state

```
   (:init (holding programmingport)
          (probabilistic 0.5 (hasVirus megatron)
                         0.5 (not (hasVirus megatron)))
          (= (fuel-level optimus) 0)
          (= (fuel-level megatron) 0)
          (= (robot-count workbench) 0))
```

In the Figure 2.7, the probabilistic effect $(probabilistic\ 0.9\ (not\ (hasVirus\ ?x)))$ states that the virus will be cleaned with 90% probability and the state will remain unchanged with 10% probability.

Figure 2.8 shows that the initial conditions, in addition to effects, can be probabilistic, which in this particular case two possible initial states with equal probability (0.5) of being true is defined for any given execution. The state variables and their values in the possible initial states for the PPDDL encoding of the *test-antivirus* problem is listed in 2.9.

**Figure 2.9** State variables and their possible initial values for the definition in figure 2.8

| Name | Type | Init1 | Init2 |
|---|---|---|---|
| onBench {optimus} | boolean | false | false |
| onBench {megatron} | boolean | false | false |
| holding {programmingport} | boolean | true | true |
| holding {fuelprobe} | boolean | false | false |
| hasVirus {megatron} | boolean | true | false |
| hasVirus {optimus} | boolean | false | false |
| fuel-level {optimus} | numeric | 0 | 0 |
| fuel-level {megatron} | numeric | 0 | 0 |
| robot-count {workbench} | numeric | 0 | 0 |

### 2.5.7 Rewards and Optimization Metrics

In PPDDL, *rewards*, being associated with state transitions and updated according to rules in action effects, are encoded using *fluents*. The fluent *reward*, presented as $reward$ or $(reward)$ in PPDDL description, is reserved to represent the accumulated reward from the beginning of the execution. The syntax for the reward fluent is as follows :

$$(\langle additive\text{-}op \rangle \ \langle reward\text{-} - fluent \rangle \ \langle f\text{-}exp \rangle), \tag{2.10}$$

where $\langle additive - op \rangle$ is $increase$ or $decrease$ and $\langle f - exp \rangle$ is numeric value stating the amount of reward. It is important to remark that accumulated reward isn't considered as part of the state space, hence preconditions and effect-conditions cannot refer to reward fluent.

For domains requiring rewards, $:rewards$ flag should be added in the requirements list. If a domain requires both probabilistic effects and rewards, $:mdp$ requirements flag can be added which implies both $:probabilistic\text{-}effects$ and $:rewards$ flags.

A goal statement $:goal\ \phi$ in PPDDL encodes the objective that should be maximized

**Figure 2.10** Reward structure in PPDDL

```
(:action program
        :parameters (?x - robot ?y - tool)
        :precondition (and (holding ?y) (onBench ?x))
        :effect (and (when (hasVirus ?x)
                        (probabilistic 0.9 (not (hasVirus ?x))))
                     (when (not (hasVirus ?x))
                        (increase (reward) 2))) )
```

for the solution of the problem, unless an optimization metric is specified explicitly. For problems associated with domains having the $:rewards$ property declared, plan objective is to maximize the expected reward by default.

Figure 2.10 demonstrates the reward structure in PPDDL for the *antivirus-domain* domain extended with actions contaning probabilistic effects. A reward of 2 is gained if the virus is cleaned after the *program* action is taken.

# CHAPTER 3

# CONTEXT-AWARE MARKOV DECISION PROCESS (CA-MDP)

The most important downside of MDP is disregarding of the available information in the environment of a given task. This information, as mentioned in the previous chapter which will make scaling and decomposition avaliable, is never inherited by some elements of previous models like MDPs. However, reinforcement learning models without information processing will fail to generate cross-domain solvers, solve problems with very high dimensionalities and even simple problems which turn into tasks with very large number of states when not mining the information. More importantly these models will fail converge in shorter times for real-world problems. Hence, information processing has vital importance, and for this purpose in this thesis a new model is proposed for representing and solving tasks in AI.

In this chapter, we present the model, Context-Aware Markov Decision Processes (CA-MDP), which is not solely based on actions, states and rewards, unlike MDP. CA-MDP will inherit the state-action relationships for a given task for more realistically modeling an environment in a given AI problem, thereby making it more suitable model for AI. Besides, the internal structure of the state-action tuple has the core information for scaling and decomposition. If there exists localized or independent structures, they will be revealed and the computational complexity of solving the problem will be alleviated significantly by enabling decomposition of the problem into smaller problems. Next, the description of the proposed state machine will be presented.

A CA-MDP is a tuple ;

$$M =< \Gamma, S, A, C, E, I > \tag{3.1}$$

where ;

- $\Gamma$ is a finite set of finite sets which define the domains for each of the state variables, $\Gamma = \{\Gamma_0, \Gamma_1, ..., \Gamma_{|S|-1}\}$ and $\forall \Gamma_i \in \Gamma, |\Gamma_i| \geq 2$,
- $S$ is a finite set of independent state variables, each from a domain $\Gamma_i$ with discrete values ,
- $A$ is a finite set of actions ,
- $C$ is a finite set of boolean formulas stating which state variables should be in which set of possible values to apply the corresponding action ,
- $E$ is the set of the effects of actions ,
- $I$ is the initial assignments to state variables .

To elaborate the details further, starting with the first element, $\Gamma$ is a set being comprised of sets, $\Gamma = \{\Gamma_0, \Gamma_1, ..., \Gamma_{|S|-1}\}$ which each $\Gamma_i$ is a finite set of discrete values specifying the domain for each state variables individually. Continuing with the set $S$, the description of MDP contains the set of actual states whereas CA-MDP contains set of state variables which the actual states are the conjunction of these variables. Each of the state variable $s_i \in S$ has a finite domain $\Gamma_i$ and the assignment $(s_i = v_{ij})$, where $v_{ij} \in \Gamma_i$, expresses that the variable $s_i$ has the value $v_{ij}$. To sum up, an actual state is : $\bigwedge_{i=0}^{|S|-1}(s_i = v_{ij})$ where $v_{ij} \in \Gamma_i$. Finally, $I$ is the initial assignments to these state variables, i.e. the tuple $((s_0 = v_0) \wedge (s_1 = v_1)... \wedge (s_{|S|-1} = v_{|S|-1}))$ where $v_i \in \Gamma_i, s_i \in S$. For problems having more than one possible initial states, $I$ can be in disjunctive normal form, see definitions after the example.

The set $A$ consists of actions and $C$ is a set of boolean formulas, having cardinality $|A|$, specifying the sufficient conditions to execute actions in *disjunctive normal form*.

Literals in the boolean formulas are assignments to states such that; $(s = v)$ where $s \in S, v \in \Gamma_i$ and $\Gamma_i \in \Gamma$ is the domain of $s$.

The domain of the set of effects of actions $E$ has the following grammar for parsing and inferring transition probabilities and rewards:

---

$E_{action} = (\wedge \quad E_0) \mid \epsilon$

$E_0 = (\vee \quad E_1\ R)\ E_0 \mid (\vee \quad E_1)\ E_0 \mid (if\ (< conditional >)\ E_0)\ E_0 \mid \epsilon$

$E_1 = (T)\ E_1 \mid \epsilon$

$T = ((\wedge \quad V)\ < probability >)\ T \mid \epsilon$

$V = (< state\_variable >=< value >)\ V \mid \top \mid \epsilon$

$R = reward\ < numeric >$

---

The $< conditional >$ part of the $if$ structure should contain a boolean formula in disjunctive normal form stating which state variables should have which value to carry out the sub-effects stated in the body of the conditional. Some very important constraints for the effects are ;

- for an effect body $T$, $(\wedge \quad (s_0 = v_0)\ (s_1 = v_1)\ ...\ (s_n = v_n) \quad p)$, where $s_i \in S,\ v_i \in \Gamma_i,\ \Gamma_i \in \Gamma,\ n \geq 0$ then $\forall ij,\ i \neq j,\ s_i \neq s_j$,
- for the probabilistic effects in the form $(\vee \quad (\wedge \quad (s_{00} = v_{00})\ (s_{01} = v_{01})\ ...\ (s_{0n_0} = v_{0n_0}) \quad p_0)...(\wedge \quad (s_{k0} = v_{k0})\ (s_{k1} = v_{k1})\ ...\ (s_{kn_k} = v_{kn_k}) \quad p_k)\ reward\ R)$, where $s_{ij} \in S$, $v_{ij}$ is a value from the domain of $s_{ij}$, $0 \leq n_i < |S|$ and $k \geq 0,\ \forall p_i \geq 0,\ \sum_{i=0}^{k} p_i = 1$,
- for an effect in the form $(\wedge \quad E_0\ E_0\ ...\ E_0) \vdash (\wedge \quad < effect_1 > ... < effect_n >)$; the changes to values of state variables should be consistent. For example, if $< effect_i >$ makes changes in the value of state variable $s$, let it change $s$'s value to $v_1$, and the $< effect_j >, i \neq j$ changes the value to $v_2$

hence leading to an ambiguity. By defining the subsets of state variables in the body of the effects *disjoint*, ambiguity is prevented.

If the effect of a particular action in a specific situation does not make any changes to any of the state variables, i.e. the actual state does not change, with some probability $q = 1 - \sum_{i=0}^{k-1} p_i$, this is expressed with the $\top$ symbol in the body of $V$ node in the grammar.

Note that the set of effects $E$ of actions also allows nested $if$ structures which should also be *consistent* such that the third constraint is not violated hence resulting no ambiguity.

The $< numeric >$ part in the $R$ node of the grammar, contains a positive or negative numerical value which defines the appropriate reward for the effect. If no reward, i.e. 0, is gained from the execution of an action, then reward needs not necessarily be declared. The $E_0$ node without the reward node $R$ in the body of the node $E_1$ represents the 0-reward effect. If a body of a effect declares more than one rewards, then they should be summed up, since the reward terminal declares either an increase(positive values) or decrease(negative values) in the accumlated reward. For example, the effect in the form $(\wedge \quad (\vee \quad E_1\ R)\ (\vee \quad E_1\ R))$, the rewards declared in the body should be summed up.

Finally, $\epsilon$ defines the empty string. If $(\wedge < effect >_i)$ or $(\vee < effect >_j)$ are generated from the grammar, the operators can be disregarded hence getting only $< effect >_i$ and $< effect >_j$ respectively.

To illustrate the notion of CA-MDP, here is an example.

**Example 3.1.** *Suppose we have an agent locked in a room with a computer on the table. The agent has to escape from the room, however, the exit door has an electronic protection, if it is not unlocked properly the software calls security. To properly unlock it, the agent needs to find the right string. To get this string, agent needs to solve a puzzle on the computer. Without any solution the agent cannot go to the door and unlock it. There are 4 outcomes of the puzzle namely 0, 1, 2, 3. If the agent solves the puzzle and if ;*

- *gets 1 (with probability* $50\%$*) then the provided string will make the software call the security with* $70\%$ *probability,*
- *gets 2 (with probability* $30\%$*) then the provided string will make the software call the security with* $30\%$ *probability,*
- *gets 3 (with probability* $10\%$*) then the provided string will make the software call the security with* $0\%$ *probability.*

*There is also a* $10\%$ *probability that the agent cannot solve the puzzle getting 0, then it needs to retry to solve the puzzle get a string. Also, there is a hidden cypher in the room which makes it easier to solve the puzzle with the best outcome. If the agent gets the cypher, and tries to solve the puzzle then the probabilities of getting 1,2 and 3 becomes* $10\%$*,* $30\%$*,* $50\%$ *respectively. The goal for agent is to unlock the door which it gets extra credit if the software doesn't call the security, otherwise it will get negative credit, penalty.*

Let us represent the above example with CA-MDP, let $M =< \Gamma, S, A, C, E, I >$ be CA-MDP where ;

- $\Gamma = \{\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\}$ where $\Gamma_0 = \{inTheRoom, nearComputer, nearDoor\}$, $\Gamma_1 = \{cypher, noCypher\}$, $\Gamma_2 = \{0, 1, 2, 3\}$, $\Gamma_3 = \{locked, unlocked\}$, $\Gamma_4 = \{notCalled, called\}$
- $S = \{location, hasCypher, hasCode, doorLock, security\}$
- $A = \{findCypher, solvePuzzle, unlockDoor, goNearComputer, goNearDoor\}$
- $I = ((location = inTheRoom) \wedge (hasCypher = noCypher) \wedge (hasCode = 0) \wedge (doorLock = locked) \wedge (security = notCalled))$
- $C = \{(hasCypher = noCypher), (location = nearComputer), (location = nearDoor), \top, (hasCode = 1) \vee (hasCode = 2) \vee (hasCode = 3)\}$
- $E = \{\boldsymbol{E_{findCypher}} = (\wedge \quad (hasCypher = cypher)\,(location = inTheRoom)\,1)$, $\boldsymbol{E_{solvePuzzle}} = (\wedge \quad (if\,(hasCypher = noCypher)\,(\vee \quad ((hasCode = 1)\,0.5)$ $((hasCode = 2)\,0.3) \quad ((hasCode = 3)\,0.1))$

$(if\ (hasCypher = cypher)\ (\vee \quad ((hasCode = 1)\ 0.1) \quad ((hasCode = 2)\ 0.3)$

$((hasCode = 3)\ 0.5))),$

$\boldsymbol{E_{unlockDoor}} = (\wedge \quad (if\ (hasCode = 1)\ (\vee \quad ((\wedge \quad (doorLocked = unlocked)$

$(security = called))\ 0.7)$

$((\wedge \quad (doorLocked = unlocked)\ (security = noCalled))\ 0.3))\ reward\ 8)$

$(if\ (hasCode = 2)\ (\vee \quad ((\wedge \quad (doorLocked = unlocked)\ (security = called))$

$0.3)$

$((\wedge \quad (doorLocked = unlocked)\ (security = noCalled))\ 0.7))\ reward\ 13)$

$(if\ (hasCode = 3)\ (\wedge \quad (doorLocked = unlocked)\ (security = noCalled)\ 1\ reward\ 15))),$

$\boldsymbol{E_{goNearComputer}} = ((location = nearComputer)1),$

$\boldsymbol{E_{goNearDoor}} = ((location = nearDoor)1)\}$

After this illustration, now we are ready to give some definitions related to CA-MDP.

**Definition 3.2.** *Let $M =< \Gamma, S, A, C, E, I >$ be a CA-MDP with $\forall \Gamma' \in \Gamma$, $|\Gamma'| = 2$ , then $M$ is called a **binary** Context-Aware Markov Decision Process.*

**Definition 3.3.** *Let $M =< \Gamma, S, A, C, E, I >$ be a CA-MDP and if $\forall i, c_i \in C$ contains no disjunctive, i.e. only conjuctive formulas, then $M$ is called a **uniconditional** Context-Aware Markov Decision Process. For uniconditional CA-MDPs, the set $C$ can be converted into a matrix with dimensions $|A| \times |S|$ where rows are actions and columns are states. Each cell $c_{ij}$ in $C$ has a value $v_{ij}$ stating the necessary values of state variables in order to execute the action, where $v_{ij} \in \Gamma'$ which $\Gamma' \in \Gamma$ is the domain of the state variable being associated with the column $j$ of matrix $C$.*

**Definition 3.4.** *Let $M =< \Gamma, S, A, C, E, I >$ be a CA-MDP with $C = \emptyset$ , then $M$ is called **unconditional** Context-Aware Markov Decision Process. Any CA-MDP can be converted into an unconditional CA-MDP by carrying the conditions into the body of effects.*

**Definition 3.5.** *Let $M =< \Gamma, S, A, C, E, I >$ be a CA-MDP where $I$ contains more than one state, being represented in disjunctive normal form, then $M$ is called **compound** Context-Aware Markov Decision Process. Note that in this case, the starting*

*state of the agent is not probabilistic. The agent starts in one of the possible states with certainty which has to be determined at the moment of the execution.*

**Definition 3.6.** *Let $M =< \Gamma, S, A, C, E, I >$ be a CA-MDP having one and only one actual state, i.e. $|S| = 1$, thereby $|\Gamma| = 1$, then $M$ is called **singular** Context-Aware Markov Decision Process.*

Any CA-MDP which does not fall into one of these categories can be regarded as ***regular*** Context-Aware Markov Decision Process.

## 3.1 Conversion : MDP $\longleftrightarrow$ CA-MDP

It is stated earlier in this chapter that CA-MDP has more expressive power over MDP. The possible two-way conversion between MDP and CA-MDP will illustrate the power of CA-MDP. The details of these conversion processes are in the subsections below.

### 3.1.1 MDP $\longrightarrow$ CA-MDP

A uniconditional MDP $M =< S, A, T, R >$, where $S$ is the set of states, $A$ is the set of actions, $T$ is the transition probabilities, i.e. $P(s'|s, a)$, and $R$ is the expected reward, i.e. $R(s, a)$, can be easily converted to a **binary unconditional** CA-MDP $M' =< \Gamma', S', A', C', E', I' >$ in *linear time*.

- $|\Gamma'| = |S|$ and $\forall \Gamma \in \Gamma', \Gamma = \{0, 1\}$, [1]
- set of state variables $S'$ would be comprised of the actual states in $S$, i.e. $S' = S$,
- the set of possible actions would be the same, i.e. $A' = A$,
- MDPs do not have conditionals so $C' = \emptyset$,
- $I'$ would be the same with the problem which the MDP is defined for.

[1] the sets $\Gamma \in \Gamma'$ can be constructed from any two different values, the set $\{0, 1\}$ is presented here for convenience

- $E'$ is constructed from the $P(s'|s,a)$ such that the effect of an action is defined as $(if(s=1)(\vee \quad ((\wedge \quad (s'_0=1)\ (s=0))\ P(s'_0|s,a))\ ...\ ((\wedge \quad (s'_n=1)\ (s=0))\ P(s'_n|s,a))\ R(s,a)))$ where $s, s'_i \in S, a \in A$ and the states are gathered from all of the possible transitions $P(s'|s,a)$ for the action $a$ defined in the MDP. The whole set $E'$ is generated by performing the above conversion for all of the transitions in $T$.

Furthermore, goal state is the same with MDP problem and note that the actual state $s$ in CA-MDP would have one and only one state variable whose value is true, the rest should be false, i.e. $s = ((s_1 = v_1) \wedge (s_2 = v_2)\ ...\ \wedge (s_{|S|} = v_{|S|}))$ where $\forall i \exists! j, i \neq j, s_i = 0$ and $s_j = 1$ [2]. Other actual states are unreachable, hence irrelevant with the problem.

As an alternative to the above procedure, the conversion of MDP $M =< S, A, T, R >$ into a **binary unconditional** CA-MDP $M' =< \Gamma', S', A', C', E', I' >$ can be done by defining $log(\lceil |S| \rceil)$ number of state variables in S' where states in MDP are being mapped to actual states in CA-MDP such that $\forall s \in S, s \mapsto \bigwedge_{i=0}^{log(\lceil |S| \rceil)-1}(s'_i = v'_i)$ where $s'_i \in S'$, $v'_i \in \Gamma = \{0,1\}, \Gamma \in \Gamma'$. This mapping can be represented as $f$ which $f : S \rightarrow \wedge_{S'}$ where $\wedge_{S'}$ is the set of all possible actual states of the target CA-MDP. Note that $f$ must be a $\mathbf{1-1}$ **mapping**, otherwise more than one state in MDP can be mapped to the same state in CA-MDP which will result in ambiguity. The conversion of the rest of the components is the same whereas construction of $E'$ can be modified as : $(if(f(s)=1)(\vee \quad ((\wedge \quad (f(s'_0)=1)\ (f(s)=0))\ P(s'_0|s,a))$ $...\ ((\wedge \quad (f(s'_n)=1)\ (f(s)=0))\ P(s'_n|s,a))\ R(s,a)))$ where $s, s'_i \in S, a \in A$.

A better way is to convert a uniconditional MDP $M =< S, A, T, R >$ into a **singular unconditional** CA-MDP $M' =< \Gamma', S', A', C', E', I' >$ by defining only one state variable having a domain $\Gamma_0 \in \Gamma', \Gamma_0 = S$. In this case, the CA-MDP behaves like one state variable having multiple states, hence modeling the actual states defined in the MDP by emitting the property of mutual exclusiveness. The new conversion formula of the set of effects $E'$ is $(if(s'=s)(\vee \quad ((s'=s'_0)\ P(s'_0|s,a))\ ...\ ((s'=s'_n)\ P(s'_n|s,a))\ R(s,a)))$, where $s' \in S'$, $s, s'_i \in S, a \in A$, which is again needed to be performed for all of the actions in the action set $A$ of MDP. Conversion of the rest

[2] Remark : $\exists!$ is the quantification for uniqueness

of the elements of the CA-MDP can be done in the similar way as above except state transitions defined in the effects of actions would be changed as transitions between different values of one state.

In the light of the conversion we can state the following theorem.

**Theorem 3.7.** *Complexity of Conversion : MDP $\longrightarrow$ CA-MDP*

*The conversion of MDP $M =< S, A, T, R >$ to a binary unconditional or singular unconditional CA-MDP $M' =< \Gamma', S', A', C', E', I' >$ has linear time computational complexity in terms of $|S|$, $|A|$ and $|T|$.*

*Proof of theorem 3.7.* **For binary unconditional CA-MDP:** Every step of conversion, i.e. conversion of individual elements, is done in linear time such that the construction of $\Gamma'$ and $S'$ is done in $\Theta(|S|)$ time, $A'$ is done in $\Theta(|A|)$ time, $C$ and $I$ is done in $\Theta(1)$ time and $E'$ is done in O(|T|) time . Adding up would give; $2 \times \Theta(|S|) + \Theta(|A|) + 2 \times \Theta(1) + O(|T|) = O(|S|) + \Theta(|A|) + O(|T|)$, thereby showing the conversion is done in linear time.

*(Note that, for the alternative method presented above, the construction of $S'$ is done in logarithmic time, i.e. $\Theta(log(\lceil |S| \rceil))$, however since a mapping needs to be defined, the complexity is the same: the mapping process takes $\Theta(|S|)$ time.)*

**For singular unconditional CA-MDP:** In this case, construction of $\Gamma'$ is done in $\Theta(|S|)$ whereas $S$ is done in $\Theta(1)$, since we have only one state variable. The rest would be the same therefore giving a linear time computational complexity; $\Theta(|S|) + \Theta(|A|) + O(|T|)$.

For both of the cases, the conversion is done in linear time in terms of $|S|$, $|A|$ and $|T|$, hence the proof is complete. □

As having demonstrated in a detailed manner, the conversion MDP $\longrightarrow$ CA-MDP is rather pretty straight forward and computationally inexpensive. The reverse of this

process is a little bit trickier.

### 3.1.2 MDP ⟵ CA-MDP

A CA-MDP $M =< \Gamma, S, A, C, E, I >$, where $\Gamma$ is the set of domains of states, $S$ is the set of state variables, $A$ is the set of actions, $C$ is the set of boolean formulas representing conditions for actions to be executed, $E$ is the effects of actions and $I$ is the starting (initial) state, can be converted into an MDP $M' =< S', A', T', R' >$ in variable time complexity *linear time* to *exponential time* which will again be proven after.

- set $S'$ can be constructed from the cartesian products of all values of state variables in their defined domains, i.e. $S' = \times_{i=0}^{|S|-1}\Gamma_i$
- set of possible actions would be the same, i.e. $A' = A$,
- $T'$ can be constructed from the effects $E$, for every state $s \in S$, execute all the possible actions, i.e. satisfying conditionals, and store the probabilities of the transitions,
- $R'$, the expected reward can be parsed from the effects, $E$.

The goal state and the initial state are the same with CA-MDP problem. If the CA-MDP is a compound CA-MDP, since the initial state needs to be determined before execution, it should be done in the same way in the constructed MDP.

The construction of set $S'$ an $T'$ using the cartesian product and exhaustive search is called **the naive method**. Since all of the possible actual states are generated, there can exist many unreachable states which these irrelevant states can be pruned after, for example with a solver. Rather than generating all of the states in the state space and then pruning the unreachable ones, only the relevant states can be generated by starting from a chosen initial state from the possible states and visiting all other states being reachable from the effects of set of actions, which is called **the search method**. The algorithmi based on *breadth-first search*, is given in algorithm 2.

**Algorithm 2** Search Method

1: $S_{MDP} := \emptyset$, $T_{MDP} := \emptyset$, $R_{MDP} := \emptyset$
2: search queue $Q := NULL$, visited map $visited := NULL$
3: Determine the initial state, $I_{current}$, $Q.push(I_{current})$
4: **while** $Q \neq NULL$ **do**
5: $currentState = Q.pop()$
6: **if** $currentState \notin visited$ **then**
7: $i := 0$
8: **while** $i \neq |A|$ **do**
9: $action := A[i]$
10: $precondition := C[i]$
11: **if** $(currentState \wedge precondition) = currentState)$ **then**
12: Generate all transitions (next states, probabilities and rewards) from the effects of the $action$ and store it in $transitionList$ and $reward$
13: **for all** $t \in transitionList$ **do**
14: $S_{MDP}.add(t.state)$
15: $T_{MDP}.add(currentState, action, t.state, t.probability)$
16: $Q.push(t.state)$
17: **end for**
18: $R_{MDP}.add(currentState, action, reward)$
19: **end if**
20: $i := (i + 1)$
21: **end while**
22: $visited.push(currentState)$
23: **end if**
24: **end while**

Note that, the line 11 in algorithm 2 is for checking whether conditions of actions are satisfied in the current state, note that conditionals can be in disjunctive normal form, any conjunctive part not satisfying the state will result in false and any conjunctive part satisfying the current state will give the current state formula, hence we got $currentState \vee false = currentState$.

**Theorem 3.8.** *Complexity of Conversion : MDP ⟵ CA-MDP*

*The conversion of CA-MDP* $M =< \Gamma, S, A, C, E, I >$ *to an MDP has linear to exponential time computational complexity in terms of* $|\Gamma|, |S|, |A|$*, the number of reachable states* $|S_{reachable}|$ *and the number of all transitions between actual states* $|T|$ *in the CA-MDP* $M$*.*

*Proof of theorem 3.8.* **Naive Method** : Since all of the actual states are generated, this computation will cost : $\prod_{i=0}^{|S|-1} |\Gamma_i|$, hence giving the lower bound $\Omega(|\Gamma|_{min}^{|S|})$, where $\Gamma_{min} = \min_{|i|} |\Gamma_i|$. The computational burden of conversion of actions is just $\Theta(|A|)$, whereas the computational complexity of generating transition probabilities and rewards are again $\Omega(|\Gamma|_{min}^{|S|})$, since we have to traverse all of the possible states. Hence, with this method we have exponential complexity bounded below. The given complexity also applies for all of the special CA-MDPs.

**Search Method** : The computational burden of conversion of actions is just $\Theta(|A|)$. To compute the complexities for states, transition probabilities and rewards the line numbers 8 - 21 of the Algorithm 2 should be analyzed. Since only the reachable states and the edges connecting them, i.e. transitions, are traversed once and for every possible transition the effects need to be parsed to generate the rest of the components of the target MDP, we have the following equation 3.2 for determining the running time of the algorithm:

$$\sum_{i=1}^{|S_{reachable}|} \sum_{j=1}^{|A|} \mathbf{1}_{S_{reachable}[i]}(C[j]) \times time_{parsing-effects}(S_{reachable}[i], A[j]) \qquad (3.2)$$

where $time_{parsing-effects}$ is the amount of time for parsing the effects for generating

information resulting from a state transition and $\mathbf{1}_B(x)$ is an indicator random variable defined as;

$$\mathbf{1}_B(x) = \begin{cases} 1 & x \wedge B = B \\ 0 & x \wedge B \neq B \end{cases} \tag{3.3}$$

where $B$ and $x$ are boolean formulas with conjunctives and in disjunctive normal form respectively.

It is important to remark that the $\mathbf{1}_{S_{reachable}[.]}(C[.]) \times time_{parsing-effects}(S_{reachable}[.], A[.])$ shows that the computation is dependent on the number of states lead by an action in a particular state. The more states that an action leads, the greater the body of the inner summation becomes. Thus, the inner summation gives the total number of connections from the current state, i.e. $S_{reachable}[i]$, to all other states made available by all of the actions possible.

In the light of these, the complexity for the whole summation in the equation 3.2 becomes $O(|S_{reachable}| + |T|)$, where $T$ is the set of the all connections, i.e. edges, between actual states.

Note that the upper bound for the body of the inner summation is $O(|S_{reachable}|)$ since from a state with a particular action the maximum number of possible transitions is $|S_{reachable}|$, if all of the actions are permitted in all of the states, then the upper bound for the whole summation becomes $O(|S_{reachable}| + |S_{reachable}| \times |A| \times |S_{reachable}|) = O(|S_{reachable}|^2 \times |A|)$.

For the special cases of CA-MDP like binary, uniconditional , unconditional and compound CA-MDP the time complexities are the same.

For singular CA-MDP, the upper bound has the complexity $O(|\Gamma_0| + |\Gamma_0| \times |A| \times |\Gamma_0|) = O(|\Gamma_0|^2 \times |A|)$, where $\Gamma_0 \in \Gamma$ and since $|\Gamma| = 1, |S| = 1$ and the number of reachable states is $|S_{reachable}| = |\Gamma_0|$.

In conclusion, if $|S_{reachable}|$ grows linearly with $|S|$ time complexity becomes linear whereas, if the growth is exponential the complexity becomes exponential, hence the

proof is complete.

□

# CHAPTER 4

# CONTEXT-AWARE VALUE ITERATION (CA-VI)

In the previous section, the notion of CA-MDP is theoretically formalized. It is shown with conversion process that the expressiveness power of CA-MDP is superior to MDP. Moreover, with the introduction of conditionals and effects, CA-MDP is more suitable for expressing the structural state-action relations. With the new information gathered from the environment, a new algorithm, Context-Aware Value Iteration (CA-VI) is presented for learning an optimal policy which improves the efficiency of the regular Value Iteration (VI) algorithm by redefining the following loop and the update rule in line 4 and 6 of algorithm 1 (given in chapter 2) respectively:

---

...

**for all** $s \in S$ **do**

...

. $V(s) = \max_{a \in A} \left[ R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s') \right]$

...

---

In VI algorithm, the inner loop iterates over all of the states not checking whether the value of state is going to change or not in that iteration step. If executing an action in a state leads to small number of states (compared to number of all), then the values, $V(s)$, of many states are not going to change thereby iterating the loop will give the same computations with the preceding discrete time step. As the number of possible

transitions decreases, the more redundant computations are performed, which is a crucial reason behind the time inefficiency of solving an MDP with large number of states. In the process of generating a policy for a CA-MDP problem, due to effects, only states, having values which are going to change in the succeeding time step, can be taken into consideration in the loop.

For the analysis of the update rule, to compute the $V(s)$ for a particular state, all of the transitions (probabilities) for all of the actions for that state should be fetched. Since, in an iteration, all of the values of the neighbor states for a state need not necessarily be updated, the update operation has some redundant computations. This can be eliminated by keeping track of states updated in an iteration. With the help of these stored states (values changed), the states and the actions, leading to those stored states, can be found out in the suceeding iteration. Only computing the state-action values and comparing them to the value function, $V(s)$, of the corresponding state should suffice, since unchanged state-action values will not have an impact to the $V(s)$, thereby improving the efficiency of the desired computation.

In the light of these facts, it can be safely concluded that VI algorithm is not *scalable* with respect to the given problem's internal state-action structure. Therefore, we define a new algorithm, CA-VI, based on VI, for solving CA-MDPs to generate an optimal policy for a given problem. The proposed algorithm eliminates all the inefficiencies in the computations mentioned above, thereby making it a *scalable* algorithm. Also, the same core idea behind the CA-VI can be easily adapted regular MDPs. The basic outline of CA-VI is demonstrated in algorithm 3.

The general idea of the CA-VI algorithm is to keep track of values of states being changed and eliminate computations that leads to same results as preceding iteration, as mentioned above. If the values of all of the states being in the immediate effects of a particular state are not being updated in an iteration then the value of that particular state needs not to be changed at all in the succeeding iteration. Based on this, CA-VI keeps track of states updated in the current iteration then to find the state-action pairs to update the corresponding $Q(s, a)$ value in the suceeding iteration, in result eliminating lots of redundant computations, alleviating the burden of the computational complexity vastly. A sample detailed implementation is given in algorithm 4 below.

**Algorithm 3** Outline of Context-Aware Value Iteration algorithm

Given a CA-MDP, $M =< \Gamma, S, A, C, E, I >$
Generate state transitions, compute $V^0(.)$ with rewards, and add updated states to $iterationList$
$\delta \leftarrow 0$
**repeat**
    Generate list $L$ containing state-actions having states in $iterationList$ in their effects, and actions leading to them
    **for all** $(s, a) \in L$ **do**
        $Q(s, a) := R(s, a) + \gamma[\sum_{s' \in S}(P(s'|a, s)\ V^k(s'))]$
        $V^{k+1}(s) := \max(Q(s, a), V^k(s))$ and update $\pi(s)$
        **if** $(|V^{k+1}(s) - V^k(s)| > 0)$ **then**
            Add $s$ to $iterationList$
        **end if**
        $\delta := \max(\delta, |V^{k+1}(s) - V^k(s)|)$
    **end for**
**until** $(|\delta| < \varepsilon)$

**Algorithm 4** A detailed implementation of CA-VI algorithm

Given a CA-MDP, $M =< \Gamma, S, A, C, E, I >$
Initialize a list of value functions $Q(.)$, $V(.)$ with 0s and policy $\pi(.)$ randomly
Initialize a list containing two hash-sets $H_{changed} := \{H1 = \emptyset, H2 = \emptyset\}$
Initialize a graph $G(.)$
Determine the initial state; $I_{current}$
Generate state transitions and compute $V^0(.)$ by executing the subroutine 5
$\delta \leftarrow 0$
**repeat**
  **repeat**
    $currentState = H_{changed}[0].pop()$
    **for all** $s \in G(currentState).ancestors$ **do**
      $a := G(currentState, s).action$
      $Q(s, a) := G(s, a).reward + \gamma[\sum_{s' \in S}(G(s, a, s').probability\ V(s'))]$
      $H_{changed}[1].add(s)$
    **end for**
  **until** $H_{changed}[0]$ is empty
  **repeat**
    $currentState = H_{changed}[1].pop()$
    **for all** $s \in H_{changed}[1]$ **do**
      $v := V(s)$
      $V(s) := \max_{a \in A} Q(s, a)$ and update $\pi(s)$
      $\delta := \max(\delta, |V(s) - v|)$
      **if** $(|V(s) - v| > 0)$ **then**
        $H_{changed}[0].add(s)$
      **end if**
    **end for**
  **until** $H_{changed}[1]$ is empty
**until** $(|\delta| < \varepsilon)$

**Algorithm 5** CA-VI Subroutine : Computation of $V^0(.)$

1: Initialize queue $Q_{states}.push(I_{current})$
2: **while** $Q_{states} \neq NULL$ **do**
3: $currentState = Q.pop()$
4: **if** $currentState$ is not visited **then**
5: $i := 0$
6: **while** $i \neq |A|$ **do**
7: $action := A[i]$
8: $precondition := C[i]$
9: **if** $(currentState \wedge precondition) = currentState)$ **then**
10: Generate all transitions (next states, probabilities and rewards) from the effects of the $action$ and store it in $transitionList$ and $reward$
11: **for all** $t \in transitionList$ **do**
12: $G.add(currentState, t.state, action, t.probability)$
13: $Q_{states}.push(t.state)$
14: **end for**
15: $G(currentState).add(action, reward)$
16: Update $V(currentState)$ and $\pi(currentState)$ according to $t.reward$ and $action$
17: **end if**
18: $i := i + 1$
19: **end while**
20: Mark $currentState$ as visited
21: **if** $V(currentState) > 0$ **then**
22: $H_{changed}[0].add(currentState)$
23: **end if**
24: **end if**
25: **end while**

The algorithm 4 starts by generating the transition graph, detailed in 5, and computing the $V^0(s), \forall s \in S$. While computing the $V^0(s)$ with immediate rewards (since value functions are initialized to 0 in the beginning), the states are stored in a hash-set, the first element of the hash-set list $H_{changed}$, if their values are changed. By storing them in a hash-set, the states whose values are going to change in the next iteration can be found out, hence only the values that are going to change can be computed by discarding the rest.

After computing zeroth step of the value iteration, by the new altered loop, rather than iteration of all of the states in the VI algorithm, the states in the current hash-set are fetched one by one to update the $Q(s, a)$ values of states who have the states in the set in their immediate effect area. Next, with the help of the second hash-set in the list $H_{changed}$, the value functions, $V(s)$, are updated according to the computed $Q(s, a)$ values and again in turn, the states are added which their values are updated into the first hash-set which are going to help to find the states to be updated in the next iteration. The values of every state in the hash-set, are updated and the algorithm ends if the values of the states in the two succeeding iteration results in difference in a pre-defined confidence interval, $\varepsilon$.

### 4.1 Example Execution of the Implementation

Consider the following tiny example of a problem modeled by a CA-MDP, $M =< \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0\}$ where $\Gamma_0 = \{0, 1, 2, 3, 4, 5, 6\}$,
- $S = \{s\}$,
- $A = \{a_0, a_1, a_2, a_3\}$,
- $I = ((s = 0))$,
- $C = \{(s = 0), (s = 0), (s = 1) \vee (s = 2), (s = 4) \vee (s = 5)\}$,
- $E = \{\boldsymbol{E_{a_0}} = (\vee \quad ((s = 1)\ 0.8) \quad ((s = 2)\ 0.2)\ reward\ 20)$,
  $\boldsymbol{E_{a_1}} = (((s = 3)\ 1)\ reward\ -100)$,

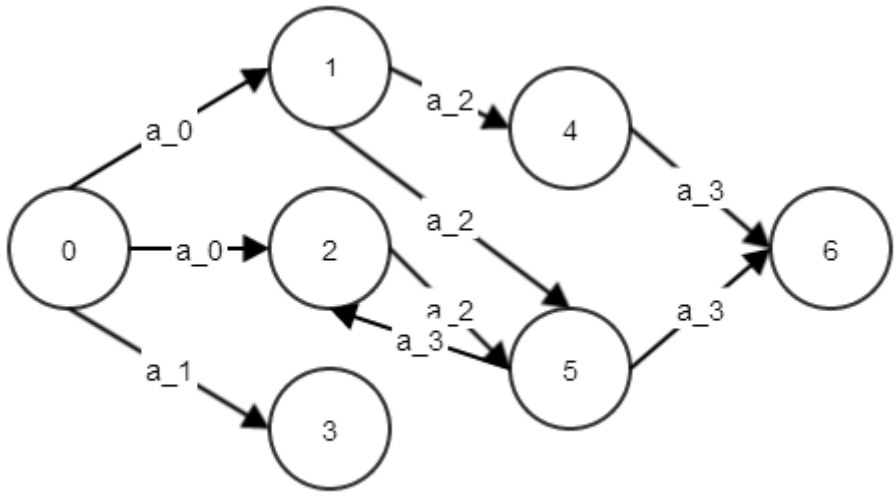
$\boldsymbol{E_{a_2}} = (\wedge \quad (if\ (s = 1)\ (\vee \quad ((s = 4)\ 0.9) \quad ((s = 5)\ 0.1)))\ (if\ (s = 2)\ ((s = 5)\ 1))),$

$\boldsymbol{E_{a_3}} = (\wedge \quad (if\ (s = 5)\ (\vee \quad ((s = 6)\ 0.6) \quad ((s = 2)\ 0.4)\ reward\ 100)))$

$(if\ (s = 4)\ (((s = 6)\ 1)\ reward\ 100))),$

Note that, the goal state is defined as $(s = 6)$. For visualization of the problem, observe the state transitions in the following figure 4.1.

**Figure 4.1** State transition graph of the problem




After the given illustration, we will demonstrate the running of CA-VI algorithm for a couple of steps.

**Step 1 :** A small number to, $0 < \varepsilon < 1$, and 1 to discount factor, $\gamma$, are assigned. Execution of the subroutine 5, i.e.generation of state transitions and computing $V^0(s), \forall s \in S$. For the visualization of the transition graph generated, refer to the figure 4.1.

After the subroutine is completed, the hash-set is $H_{changed}[0] = \{(s = 0), (s = 4), (s = 5)\}$, and the states of the variables are updated as follows :

Table 4.1: $Q(s, a)$, $V(s)$ and $\pi(s)$ after step 1

| Variable | Value | Variable | Value | Policy | Value |
|---|---|---|---|---|---|
| $Q(s = 0, a_0)$ | 0 | $V(s = 0)$ | 20 | $\pi(s = 0)$ | $a_0$ |
| $Q(s = 0, a_1)$ | 0 | | | | |
| $Q(s = 1, a_2)$ | 0 | $V(s = 1)$ | 0 | $\pi(s = 1)$ | $a_2$ |
| $Q(s = 2, a_2)$ | 0 | $V(s = 2)$ | 0 | $\pi(s = 2)$ | $a_2$ |
| | | $V(s = 3)$ | 0 | $\pi(s = 3)$ | $\emptyset$ |
| $Q(s = 4, a_3)$ | 0 | $V(s = 4)$ | 100 | $\pi(s = 4)$ | $a_3$ |
| $Q(s = 5, a_3)$ | 0 | $V(s = 5)$ | 100 | $\pi(s = 5)$ | $a_3$ |

**Step 2 :** For the second part of the algorithm, the loop between the lines 9-16 in the algorithm 4 is executed in a detailed manner shown as substeps below.

**Step 2.1 :** Pop an element from the hash-set, $H_{changed}[0] = \{(s = 0), (s = 4), (s = 5)\}$ ; $currentState = (s = 0)$, since it isn't in the immediate effect states of any node, the loop is not executed.

**Step 2.2 :** Pop an element from the hash-set, $H_{changed}[0] = \{(s = 4), (s = 5)\}$; $currentState = (s = 4)$, it is in the immediate effect list of (only) state $(s = 1)$. Then by computing the value will give; $Q((s = 1), a_2) := 0 + 1 * (0.9 * 100 + 0.1 * 100)$ which is equal to 100. $H_{changed}[1] = \{(s = 1)\}$ .

**Step 2.3 :** Pop an element from the hash-set, $H_{changed}[0] = \{(s = 5)\}$; $currentState = (s = 5)$, it is in the immediate effect list of states $(s = 1), (s = 2)$. Then by computing the value of $(s = 1)$ will give; $Q((s = 1), a_2) := 100$ and $Q((s = 2), a_2) := 100$. $H_{changed}[1] = \{(s = 1), (s = 2)\}$ .

Overall values of variables after Step 2 is given in table 4.5.

**Step 3 :** After the computations of $Q(.)$ values, $V(.)$ values are going be updated. The loop between the lines 17-27 in the algorithm 4 is executed in a detailed manner shown as substeps below.

**Step 3.1 :** Pop an element from the hash-set $H_{changed}[1] = \{(s = 1), (s = 2)\}$ ; $currentState = (s = 1)$. $V(s = 1) = 100$ and $\pi(s = 1) = a_2$. Push $(s = 1)$ to hash-set, thereby resulting, $H_{changed}[0] = \{(s = 1)\}$.

Table 4.2: $Q(s,a)$, $V(s)$ and $\pi(s)$ after step 2 (changes are bold)

| Variable | Value | Variable | Value | Policy | Value |
|---|---|---|---|---|---|
| $Q(s=0,a_0)$ | 20 | $V(s=0)$ | 20 | $\pi(s=0)$ | $a_0$ |
| $Q(s=0,a_1)$ | -100 | | | | |
| $Q(s=1,a_2)$ | **100** | $V(s=1)$ | 0 | $\pi(s=1)$ | $a_2$ |
| $Q(s=2,a_2)$ | **100** | $V(s=2)$ | 0 | $\pi(s=2)$ | $a_2$ |
| | | $V(s=3)$ | 0 | $\pi(s=3)$ | $\emptyset$ |
| $Q(s=4,a_3)$ | 100 | $V(s=4)$ | 100 | $\pi(s=4)$ | $a_3$ |
| $Q(s=5,a_3)$ | 100 | $V(s=5)$ | 100 | $\pi(s=5)$ | $a_3$ |

**Step 3.2 :** Pop an element from the hash-set $H_{changed}[1] = \{(s=2)\}$ ; $currentState = (s=2)$. Then, $V(s=2) = 100$ and $\pi(s=2) = a_2$. Push $(s=2)$ to hash-set, thereby resulting, $H_{changed}[0] = \{(s=1),(s=2)\}$.

Table 4.3: $Q(s,a)$, $V(s)$ and $\pi(s)$ after step 3 (changes are bold)

| Variable | Value | Variable | Value | Policy | Value |
|---|---|---|---|---|---|
| $Q(s=0,a_0)$ | 20 | $V(s=0)$ | 20 | $\pi(s=0)$ | $a_0$ |
| $Q(s=0,a_1)$ | -100 | | | | |
| $Q(s=1,a_2)$ | 100 | $V(s=1)$ | **100** | $\pi(s=1)$ | $a_2$ |
| $Q(s=2,a_2)$ | 100 | $V(s=2)$ | **100** | $\pi(s=2)$ | $a_2$ |
| | | $V(s=3)$ | 0 | $\pi(s=3)$ | $\emptyset$ |
| $Q(s=4,a_3)$ | 100 | $V(s=4)$ | 100 | $\pi(s=4)$ | $a_3$ |
| $Q(s=5,a_3)$ | 100 | $V(s=5)$ | 100 | $\pi(s=5)$ | $a_3$ |

**Step 4 :** The loop between the lines 9-16 should be executed again, since hash-set is not empty.

**Step 4.1 :** Pop a next element from the hash-set, $H_{changed}[0] = \{(s=1),(s=2)\}$ ; $currentState = (s=1)$, it is in the immediate effect list of (only) state $(s=0)$. Then by computing the value will give; $Q((s=0),a_0) := 20 + 1 * (0.8 * 100 + 0.2 * 100)$ which is equal to 120. $H_{changed}[1] = \{(s=0)\}$ .

**Step 4.2 :** Pop a next element from the hash-set, $H_{changed}[0] = \{(s=2)\}$; $currentState = (s=2)$, it is in the immediate effect list of (only) state $(s=0)$. Then by computing the value will give; $Q((s=0),a_0) := 20 + 1 * (0.8 * 100 + 0.2 * 100)$ which

is equal to 120. $H_{changed}[1] = \{(s = 0)\}$ .

Table 4.4: $Q(s, a)$, $V(s)$ and $\pi(s)$ after step 4 (changes are bold)

| Variable | Value | Variable | Value | Policy | Value |
|---|---|---|---|---|---|
| $Q(s = 0, a_0)$ | **120** | $V(s = 0)$ | 20 | $\pi(s = 0)$ | $a_0$ |
| $Q(s = 0, a_1)$ | -100 | | | | |
| $Q(s = 1, a_2)$ | 100 | $V(s = 1)$ | 100 | $\pi(s = 1)$ | $a_2$ |
| $Q(s = 2, a_2)$ | 100 | $V(s = 2)$ | 100 | $\pi(s = 2)$ | $a_2$ |
| | | $V(s = 3)$ | 0 | $\pi(s = 3)$ | $\emptyset$ |
| $Q(s = 4, a_3)$ | 100 | $V(s = 4)$ | 100 | $\pi(s = 4)$ | $a_3$ |
| $Q(s = 5, a_3)$ | 100 | $V(s = 5)$ | 100 | $\pi(s = 5)$ | $a_3$ |

**Step 5.1 :** After the computations of $Q(.)$ values, $V(.)$ values are going be updated again (lines 17- 27). Pop an element from the hash-set $H_{changed}[1] = \{(s = 0)\}$ ; $currentState = (s = 0)$, $V(s = 0) = 120$ and $\pi(s = 0) = a_0$. Push $(s = 0)$ to hash-set, thereby resulting, $H_{changed}[0] = \{(s = 0)\}$.

Table 4.5: $Q(s, a)$, $V(s)$ and $\pi(s)$ after step 5 (changes are bold)

| Variable | Value | Variable | Value | Policy | Value |
|---|---|---|---|---|---|
| $Q(s = 0, a_0)$ | 120 | $V(s = 0)$ | **120** | $\pi(s = 0)$ | $a_0$ |
| $Q(s = 0, a_1)$ | -100 | | | | |
| $Q(s = 1, a_2)$ | 100 | $V(s = 1)$ | 100 | $\pi(s = 1)$ | $a_2$ |
| $Q(s = 2, a_2)$ | 100 | $V(s = 2)$ | 100 | $\pi(s = 2)$ | $a_2$ |
| | | $V(s = 3)$ | 0 | $\pi(s = 3)$ | $\emptyset$ |
| $Q(s = 4, a_3)$ | 100 | $V(s = 4)$ | 100 | $\pi(s = 4)$ | $a_3$ |
| $Q(s = 5, a_3)$ | 100 | $V(s = 5)$ | 100 | $\pi(s = 5)$ | $a_3$ |

The loops will be executed once but since the state $(s = 0)$ isn't in the immediate effect list of any state, no changes will be made, and the algorithm is going to halt.

## 4.2 Analysis of CA-VI

As explained earlier, CA-VI algorithm eliminates the inefficiencies in the loop and update rule of the regular VI algorithm. The loop of the VI algorithm iterates for

every state not checking whether a value of a state is going to update or not, on the other hand CA-VI iterates only on the states whose value states are going to change.

The connectivity of the state graph of a task determines the complexity of CA-VI, whereas since VI iterates for all of the states, the number of nodes (state) is a crucial factor in the state-action transition graph. However, in reality to discard redundant updates, it is vital to take the connectivity into consideration. If a graph is sparsely connected, then a change in a state may change small number of states compared to all, in contrast in densely connected graphs a change in value of a state may affect many states. VI does not consider the structure of the connectivity graph of a task executing updates for all of the states, however CA-VI considers the structure of the graph which its running time highly depends on the sparsity of the graph therefore making it a scalable solver algorithm.

Combined with the inefficiency in the update rule, VI algorithm runs as regarding the state connectivity graph of a task is always fully connected. However, since CA-VI considers that structure, CA-VI has the same complexity with the VI algorithm, provided that the states of a given task is fully connected thereby forcing CA-VI algorithm to add all of the states to the lists (hash-sets provided in the implementation) to update them in the iterations.

# CHAPTER 5

# CONCLUSION

Although reinforcement learning provides important models and algorithms for autonomous agents to model a problem and generate an optimal policy, as detailed in previous chapters, the computational complexity of MDP model and VI algorithm, by means of space and time respectively, becomes disastrous as the state space of the given task grows large. Even though there are many proposed solutions for the scability problem of MDP (and VI), this work can be considered as an important and preliminary study aims to alleviate the burden of the complexities by integrating the information available in a given problem.

In this thesis, we present a new model, CA-MDP, based on regular MDP, having more expressive power than MDP which alleviates the space complexity vastly and a new algorithm, CA-VI, which is based on the Bellman update and computationally superior to the VI algorithm. CA-MDP changes the state definition from actual states of the problem to state variables, therefore actual states are defined as the conjunction of these state variables provided in the definition. In addition, CA-MDP introduces conditionals to define the admissibility of actions in particular situations. Moreover, inspired from PPDDL, rather than providing full description of transition probabilities, i.e. $P(s'|s,a)$, it defines effects of actions which is a more compact and programmer-friendly form of defining effects of actions in particular states. This structured definition, indeed, shows that CA-MDP is more suitable for integrating the structural information in a modeled problem, thereby making it more AI-oriented than the MDP model.

On the other hand, the proposed algorithm for solving CA-MDPs, CA-VI, takes ad-

vantage of the effects of actions in states and use this information for updating the value function, $V(s)$, in a significantly more efficient way. As shown in the previous chapter, CA-VI eliminates the redundant computations by discarding the updates which have no effect on value function in an iteration step, rather than updating the value function for all of the states without taking into account whether the value is going to change in an iteration or not. Even though CA-VI performs less compuations than VI, same overall changes to the value functions are made, leading to same result as VI, hence guaranteeing an optimal policy, as VI does. Besides the basic idea and outline of the algorithm, an implementation and a trace of the algorithm with a given basic and theoretical problem, is provided to perfectly give an intuitive understanding of the algorithm. It is important to remark that, the given implementation is not the fastest one among the many, some many other implementations can be derived also, by taking into account the time and space constraints which will lead to more efficient computations according to the constraints of the computing environment of the agent.

## 5.1 Future Improvements and Directions

As for future studies, the most important improvement will be defining factored representations for CA-MDPs. The very definition of the CA-MDP makes itself very suitable for factoring and decomposing the original CA-MDP defined for a problem into smaller CA-MDPs representing the localized sub-tasks. After, each of the CA-MDP can solved individually, and if not being causally connected they can be solved in a parallel fashion.

As for futher engineering to lessen the effect of computational complexity, the ideas like exits [11] or options [22] or hierarhical models for CA-MDPs can be proposed. These approaches can prove themselves useful with the powerful expresiveness structure of CA-MDP sheltering the state-state and state-action relationships from the very definition.

The partial observable extension for MDPs, POMDP, also suffers from the burden of the computational complexity, even more than MDPs. With the same motivation of CA-MDPs, using the information available in an environment given, extension to CA-

MDPs can be theorized, namely Partially Observable CA-MDPs, enabling the model for partially observable environments. Again, by the help of effects and conditionals, the representation and computations for solving problems in partially observable environments can be greatly enhanced. The new model also should be suitable for defining factored and/or decentralized form of the problem.

Finally, in this study we only consider the causal relationships between states and actions. By taking into consideration of the ultimate problem of AI, and for sake of taking a huge step towards it, the CA-MDP model can be enhanced such that it should enable using data processing approaches, i.e. pattern recognition algorithms, to mine the information presented in the problem, therefore restructuring the model for more efficient representation and improving the efficiency of the solver algorithms for the purpose of generating an optimal policy.

# REFERENCES


[1] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[3] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1104–1111, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 106–111. AAAI, 1997.

[6] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, December 1989.

[7] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *arXiv preprint cs/9905014*, 1999.

[8] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[9] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:2003, 2003.

[10] Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[11] Bernhard Hengst. Discovering hierarchy in reinforcement learning with hexq. In *In Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.

[12] Anders Jonsson and Andrew Barto. Causal graph based decomposition of factored mdps. *J. Mach. Learn. Res.*, 7:2259–2301, December 2006.

[13] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99 – 134, 1998.

[14] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *In Proceedings of the Twenty-First International Conference on Machine Learning*, pages 560–567. ACM Press, 2004.

[15] Amy Mcgovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the eighteenth international conference on machine learning*, pages 361–368. Morgan Kaufmann, 2001.

[16] Marvin Minsky. *The Society of Mind*. Simon & Schuster, Inc., New York, NY, USA, 1986.

[17] Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *IFIP Congress*, pages 256–264, 1959.

[18] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.

[19] Edwin P. D. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *KR*, pages 324–332, 1989.

[20] C. E. Shannon. XXII. Programming a computer for playing chess. *Philosophical Magazine (Series 7)*, 41(314):256–275, 1950.

[21] James Robert Slagle. A heuristic program that solves symbolic integration problems in freshman calculus, symbolic automatic integrator (saint). Technical report, DTIC Document, 1961.

[22] Richard S Sutton. Between mdps and semi-mdps: Learning, planning, and representing knowledge at multiple temporal scales. 1998.

[23] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[24] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[25] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pages 385–392. MIT Press, 1995.

[26] Håkan L. S. Younes and Michael L. Littman. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. 2004.