A CONTEXT-AWARE MODEL FOR STOCHASTIC PLANNING IN ENVIRONMENTS WITH HIDDEN STATES

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 DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

A CONTEXT-AWARE MODEL FOR STOCHASTIC PLANNING IN ENVIRONMENTS WITH HIDDEN STATES

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

Prof. Dr. Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

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

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu
Electric and Electronics Engineering, METU _____

Prof. Dr. Faruk Polat
Computer Engineering, METU _____

Prof. Dr. Ahmet Coşar
Computer Engineering, Çankaya Üniversitesi _____

Assoc. Prof. Dr. Erol Şahin
Computer Engineering, METU _____

Assoc. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU _____

10.09.2021:

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, Surname: ÖMER EKMEKCİ

Signature :

ABSTRACT

A CONTEXT-AWARE MODEL FOR STOCHASTIC PLANNING IN ENVIRONMENTS WITH HIDDEN STATES

EKMEKCİ, ÖMER
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Faruk Polat

September 2021, 69 pages

Partially Observable Markov Decision Processes (POMDP) have been used extensively to formalize representations for decision-theoretic planning problems to be solved under uncertainty. In this setting, autonomous agents do not have the perfect state information. Thus, agents need to store a memory for keeping track of which state it is in depending on the observations. In domains with huge state spaces, policy generation becomes costly. In order to overcome this problem, compact representations using propositional logic and/or grammar-based models are needed. These representations benefit from the underlying state-action relationship in a given problem setting. However, plain POMDPs do not encode these relationships. Existing exact solution algorithms for POMDP planning are inefficient at determining a useful policy in task with huge state space. Based on this motivation, in this thesis, we take our inspiration from an earlier work and propose a new grammar-based model called Context-Aware POMDP (CA-POMDP) for the purpose of representing Markovian sequential decision making problems in a more structured manner in partially observable environments. CA-POMDP changes and augments POMDP facilities by integrating causal relationships between states, actions and observations thereby en-

abling structural, compact if possible, representation of the tasks. To show the expressive power of CA-POMDP, we give the theoretical bounds for complexity of conversion between POMDP and CA-POMDP. Second, we enhance a policy generation algorithm for fully observable domains to reveal the way for solution procedures for partially observable domains which uses the local relationships for improved performance. We give theoretical definition and analysis of our solution algorithm then present our conducted experiments on numerous problems. Results show that incorporation of context dependent information to solver algorithm significantly improved policy generation.

Keywords: Markov process, stochastic planning, decision making, partial observability, context-aware

ÖZ

SAKLI DURUMLARI OLAN ORTAMLARDA STOKASTİK PLANLAMA İÇİN BAĞLAM-FARKINDALIĞI OLAN MODEL

EKMEKCİ, ÖMER
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Faruk Polat

Eylül 2021 , 69 sayfa

Kısmi Gözlemlenebilir Markov Karar Süreçleri (KGMKS) belirsizlik altında çözülmesi gereken stokastik planlama problemlerinin gösterimlerinin formalize edilmesi için yaygın bir biçimde kullanılmaktadır. Bu düzende, otonom etmenler durum bilgisine kusursuz bir biçimde sahip değildir. Bunun için, etmenler gözlemlerine bağlı olarak hangi durumda olduğunun bilgisini saptamak için bellek tutma gereksinimi duyarlar. Durum uzayının büyük olduğu alanlarda plan oluşturmak çok masraflı hale gelebilir. Bu problemin üstesinden gelmek için, önermesel mantık ve/veya gramer-tabanlı modeller kullanılarak elde edilen kompakt gösterimler oldukça faydalıdır. Bu gösterimler, verilen bir problemde bulunan durum-aksiyon ilişkilerinden faydalanır. Yine de, KGMKS'ler bu ilişkileri kodlamazlar. KGMKS planlaması için var olan kesin çözüm algoritmaları çok büyük durum uzayına sahip bir görev için işe yarar bir poliçe bulma konusunda verimsizlerdir. Buradan hareketle, bu tezde, daha önceki bir çalışmadan ilham alarak kısmi gözlemlenebilir ortamlarda Markov ardışık karar verme problemlerini daha yapısal bir biçimde temsil etmek için Bağlam-Farkında KGMKS (BF-KGMKS) isimli yeni gramer-tabanlı bir model sunduk. BF-KGMKS, KGMKS

yapılarını durum, aksiyon ve gözlemler arasındaki nedensel ilişkileri entegre ederek değiştirerek geliştirir ve böylece görevlerin yapısal, mümkünse kompakt, gösterimini sağlar. BF-KGMKS'nin ifadesel avantajını göstermek için, KGMKS ve BF-KGMKS arasındaki dönüşümün karmaşıklığının kuramsal sınırlarını çizdik. İkinci olarak, tam gözlemlenebilir alanlar için, daha iyi bir performans adına yerel ilişkileri de kullanan ve kısmi gözlemlenebilir ortamlar için olabilecek çözüm yöntemine de yol göstermesi için bir plan üretme algoritması sunduk. Çözüm algoritmamızın kuramsal tanımını ve analizini yapmış olup farklı problemler ile gerçekleştirdiğimiz deneylerimizi sunduk. Sonuçlar, içerik bilgisinin çözüm algoritmasında kullanılmasının poliçe oluşturulmasını önemli ölçüde geliştirdiğini göstermiştir.

Anahtar Kelimeler: Markov süreci, stokastik planlama, karar verme, kısmi gözlemlenebilme, bağlam-farkında

Remember this : Time is of the essence

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Dr. Faruk Polat for his support and guidance. It was a great honor to work under supervision of him during my PhD studies. I also thank my thesis committee members for their valuable comments and guidance. I am also highly obliged in taking the opportunity to sincerely thanks to all staff members of my department, espcially Fatos T. Yarman Vural and İsmail Hakkı Toroslu, for their guidance, generous attitude and friendly behavior. Morever, being sincerely very grateful to countless friends of mine that were highly supportive during my studies, I would like to thank my dear friend Abdullah Al-chihabi the most for his friendship and support.

At last but not the least, I am very grateful to my family which I owe my everyhing to. I wouldn't be here if it weren't for them. I will succeed no matter what to make them proud.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiii

LIST OF FIGURES xiv

LIST OF ABBREVIATIONS xvi

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation and Problem Definition 1

1.2 Proposed Methods and Models 3

1.3 The Outline of the Thesis 4

2 BACKGROUND AND SURVEY 5

2.1 Reinforcement Learning and Planning 5

2.1.1 Markov Decision Processes 6

2.1.1.1 Evaluating MDPs and Finding an Optimal Policy 8

2.1.2 Partially Observable Markov Decision Processes 10

2.1.2.1 Evaluating POMDPs and Finding an Optimal Policy . . 11

2.2 Factorization of MDPs & POMDPs 13
2.2.1 Graphical Models for Compact Representation of MDPs/POMDPs 14
2.3 Abstraction & Hierarchical Structures 16
2.4 Non-Stochastic & Stochastic Planning 18
2.4.1 Planning Domain Definition Language (PDDL) 19
2.4.2 Probabilistic Planning Domain Definition Language (PPDDL) 19
3 CONTEXT-AWARE PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES 23
3.1 Context-Aware Markov Decision Processes 24
3.2 Context-Aware Partially Observable Markov Decision Processes . . . 25
3.2.0.1 Belief State Representations & Updates 27
3.2.1 Problem 28
3.2.2 Families 30
3.2.3 Conversion: POMDP $\longleftrightarrow$ CA-POMDP 31
4 A SOLUTION METHOD FOR FULLY OBSERVABLE ENVIRONMENT . 35
4.1 Solving a CA-MDP : Actual State Representation and Conditionals . 35
4.1.1 CA-MDP Value Iteration (CA-MDP VI) 37
4.1.2 Experiments 39
5 CONCLUSION AND FUTURE WORK 59
5.1 Conclusion and Future Work 59
REFERENCES 63

LIST OF TABLES

TABLES

Table 4.1 Running times of solver algorithms with the given formulations for the given number of epochs (in msecs) 42

Table 4.2 Policy generated for $Small$ 42

Table 4.3 Running times of solver algorithms with the given formulations for the given number of epochs (in msecs) 47

Table 4.4 Policy generated for $Hypercube_{3,2,dense}$, actions are given with their initials 48

Table 4.5 Running times of solver algorithms with the given formulations for the given number of epochs (in msecs) 50

Table 4.6 Running times of solver algorithms with the given formulations for the given number of epochs (in msecs) 53

Table 4.7 Comparison of average running times of CA-MDP VI algorithm with factorized and non-factorized formulations (in secs) 57

LIST OF FIGURES

FIGURES

Figure 2.1 *coffee* domain 20

Figure 2.2 PPDDL declaration of initial state assignments: *coffee* task . . . 21

Figure 4.1 State-action transition graph of $Small$ 41

Figure 4.2 Average accumulated rewards for formulation $Small$. 43

Figure 4.3 State-action transition graph of $Hypercube$. Cell legend; white: initial, pink: hazardous, purple: dead-end, green: safe, blue: cypher, yellow: exit. Each diagonal transition occurs with 0.5 probability, rest is with 1 including $(cypher = 0) \rightarrow (cypher = 1)$ transition (not shown). 44

Figure 4.4 Average accumulated rewards for formulation $Hypercube_{3,2,dense}$. 47

Figure 4.5 Average accumulated rewards for formulation $Coffee$. 50

Figure 4.6 Taxi problem 52

Figure 4.7 Bidirectional topology (left), conditional probabilities when the action is *noop* (right). When a machine is in failure condition it remains in the same condition. 53

Figure 4.8 Average running times for SysAdmin problems with various number of machines. All have bidirectional ring topology. 55

Figure 4.9 Average running times for SysAdmin problems with various number of machines. All have bidirectional ring topology. 56

LIST OF ABBREVIATIONS

RL	Reinforcement Learning
MDP	Markov Decision Process
DBN	Dynamic Bayesian Network
POMDP	Partially Observable Markov Decision Process
CA-MDP	Context-Aware Markov Decision Process
CA-POMDP	Context-Aware Partially Observable Markov Decision Process
DP	Dynamic Programming
CA-MDP VI	Context-Aware Markov Decision Process Value Iteration

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Representing and solving stochastic and deterministic decision making problems require specific models and algorithms. *Markov Decision Processes (MDPs)* [1] are commonly used as formal framework for such problems. MDPs use states, actions, transitions and rewards to represent a task, and with this representation at hand dynamic programming (DP) oriented algorithms are employed to produce an optimal policy. These algorithms guarantee generating optimal/near-optimal policies for a sequential decision problem given enough resources of time and space. However, one critical downside is that they disregard the structural information available which leads to a rather unprocessed representation and inefficient generation of optimal policies when the state space complexity grows large which is the case in real world applications. Hence, in reinforcement learning, notions of factorization [2, 3, 4, 5, 6] and abstraction [7, 8] including hierarchical decomposition [9, 10] and tree-based abstraction [11] are vital. Many methods like state abstraction [12], causal influence networks [3] and hierarchical decomposition [13] are proposed to overcome this problem by defining sub problems in some way, namely sub-MDPs, like the divide and conquer paradigm, then after solving those sub-problems independently, the results are merged to constitute the policy. In this study, we chose to take another path and not only exploit state relationships but also integrate state-action dependencies to enable factorization, a bottom-up approach, in contrast to first defining problem then engineer certain aspects to get factorization which is a top-down approach. Note that, our approach is also widely open to further top down engineering which is beyond the scope of this study, yet its representation elements can most definitely alleviate the

computations to divide the problem into sub independent problems. However, it is important to remark that MDP is originally proposed for modeling general stochastic problems, not only for AI specifically. AI problems are mostly real world problems having lots of crucial information that is useful for efficient representation and solution. Without a hint of that available context and integrating it a priori, like certain dependencies especially between states and actions, the true nature of the task becomes harder to be revealed thereby leading to inefficiencies and computational burden in both representation and solution.

Problems modeled as MDPs assume full observability of the environment. At every time step, agent knows exactly what state it is in. Most of the problems in the real world does not have full observability. For example, sensors of an agent may collect information with some probability from the environment and agent needs to take this into consideration in its computations hence leading to uncertainty about the state it is in. *Partially Observable Markov Decision Processes (POMDPs)* are suited for problem settings having partial observability. POMDP formulation expresses the imperfect/partial state information and allows reasoning about which possible action to take under uncertainty. The aim is again to find a method to choose actions to maximize rewards. However, in contrast with MDPs, to keep track of which possible state that the agent might be in, a probability distribution over the states needs to be stored. This distribution, called *belief state*, is generated by successively being updated every time agent makes an observation. Numerous POMDP studies have been reported in AI community to address the planning and decision making under uncertainty [14, 15, 16, 17, 18, 19].

The set of belief states has infinite cardinality making it impossible to iterate over belief states to find optimal value function. Various algorithms yielding a representation of a policy that maps actions to all belief states by using piece-wise linear representations of value functions have been proposed [20, 21, 14, 15, 16]. algorithms generating exact solutions have double-exponential complexity in the worst case in the horizon and in many cases undecidable [22]. In addition to this curse of state dimensionality, problems of MDP mentioned in first paragraph, are also applicable to partially observable settings, hence making the formulation in need of factored representations to alleviate the costs of finding an optimal policy. Algorithms for factored

representations of POMDPs are detailed in the next section.

The motivation behind this study is based on the scalability problem which forbids efficient generation of policies for both fully and partially observable settings.

1.2 Proposed Methods and Models

In this thesis, we first proposed a grammar-based model for partially observable settings which benefits from factored representation. Second, we proposed a dynamic programming (DP) based solution algorithm for fully observable settings. We leave finding an efficient solution method for our model for partially observable environment as future work.

Our proposed model called, *Context-Aware Partially Observable Markov Decision Processes (CA-POMDP)* is a factored representation of a regular POMDP which also expresses of relationships between states, actions, observations and beliefs. In this model, states are represented as a conjunction of state variables whereas transitions, observations, rewards and beliefs are represented in a grammar-based form called *effects* as in [23, 24, 25]. In addition, a new element which is preconditions of actions is introduced which indicates the prerequisite values of state variables before executing an action. With this setting, a formulation can be encoded much more efficiently and expressively which guides solution algorithms to generate policy improving the time cost. Another major benefit is, given a task having localized relationships between states, actions and observations, which is generally the case for real world problems, with our formulation, these are revealed more easily than a regular POMDP enabling use of hierarchical models efficiently. Next, belief updates are also factorized to reduce the complexity of update computation which is an important bottleneck in generation of optimal policy in partially observable setting. Hence, our model is more compact and expressive than POMDP and it will be highly beneficial for sequential decision making problems in environments with partial observability.

Finally, in a fully observable setting in order to get the benefits of those elements in generating policies, we proposed an extension to regular value iteration (VI) algorithm namely CA-MDP Value-Iteration (CA-MDP VI). CA-MDP VI applies similar

updates as VI however taking into account of the task structure supplied with its problem formulation thus eliminating redundant computations and converge to optimal policy in a faster manner.

1.3 The Outline of the Thesis

The flow of this thesis is organized into five main titles with a conclusion section in the end which the next section presents the background of stochastic planning and Markov Processes and relevant studies. Next, we introduce Context-Aware POMDP formulation with an example for clarification. In addition, conversions between a regular POMDP and CA-POMDP is given to demonstrate expressive power. In the next section a value iteration algorithm for fully observable environments, CA-MDP VI is introduced with its theoretical background. Also, experiments are carried out in various domains. In those trials, we managed to justify our theoretical claims empirically and give detailed discussions about the results. We finish our paper with a conclusion section in the end giving an overall discussion and remarks with future work blended in.

CHAPTER 2

BACKGROUND AND SURVEY

2.1 Reinforcement Learning and Planning

Reinforcement learning (RL) is a type of learning which focuses on techniques and algorithms for solving sequential decision making problems. In this setting, autonomous agents try to learn useful aspects of the given problem using certain signals to make sequence of decisions conditioned on the history. These signals, called *rewards*, behave as a supervision for the agent enabling learning the best action map given a configuration of the problem state space, i.e. *policy*. Hence, the main foundations of the RL paradigm can be summarized as [26]:

- **model**, is a representation for the problem environment. It can come in various flavors however key point is that agent may or may not have knowledge all of the model hence now knowing which state is in (e.g. sensor error) and it can be forced to act according to its belief distribution over states.

- **reward**, is a real-valued supervision signal to guide the agent by defining what is useful and not in the environment which is typically gathered at each time step by the agent. The aim of the agent is to maximize the expected reward gathered over the long run in the given state-action space.

- **value function**, is a real-valued mapping from states to their assigned expected utilities. The expectations are computed by the agent from immediate and future rewards indicating the long term goodness of states when the expectation of immediate rewards of future possible states (that are available from the current one) are taken into account. The future sight, i.e. the amount how much of

the future rewards are taken into account, is determined by a a parameter called *discount value*.

- **policy**, is a mapping from states to actions which sufficiently defines the behavior of the agent in the environment it is in. They may also be stochastic, mapping states to probability distributions over certain actions, hence enabling uncertainty in the selection of actions in a state.

In a nutshell, an agent in a given problem setting: (i) interacts with the environment, (ii) gets rewards and observations (iii) generates value functions by evaluating states with available actions either with trial-error/experience or offline, $(iiii)$ generates an state-action policy according to the value function.

The environment model and rewards can be encoded in Markov Decision Processes [1, 27] and various dynamic programming (DP) based solution algorithms are proposed for generation of value functions and policies.

Here, we briefly sketch the general Markovian framework with discrete time steps which most of RL studies are employed on.

2.1.1 Markov Decision Processes

A Markov decision process (MDP) [1] is a tuple ;

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

where ;

- S is a set of states
- A is a set of actions
- $T : S \times A \times S \rightarrow [0, 1]$ is a mapping defining the transition such that $T(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability measure stating the probability that the current state changes from s to s' when action a is taken.

Note that, $T(s, a, .)$ defines a probability measure hence $\sum_{s' \in S} T(s, a, s') = 1 \forall s \in S$and $a \in A$. The environment is *deterministic* if the range is $0, 1$ rather than interval otherwise it is *stochastic*.

- $R : S \times A \rightarrow \mathbb{R}$ is the reward function defining the accumulated reward amount when being in state s and taking action a.

Given discrete time steps, $t \in \mathbb{N}$ denoting current time step, an agent being in state s_t interacts with the environment by choosing action a_t and observing next state s_{t+1} and getting reward r_t as follows:

$$s_{t+1} \sim T(s_t, a_t, .) = P(s_{t+1} \mid s_t = s, a_t = a), \forall s, s' \in S, a \in A$$
$$r_t = R(s_t, a_t)$$

The aim of the agent is to generate a policy $\pi : S \rightarrow A$ that maximizes the accumulated discounted reward in the long run.

To generate policy that maximizes the expected reward over states, a value function is defined $V^{\pi}(s)$ which reflects expected cumulative discounted reward for state s. In order to compute that for a state following a policy, we have;

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

where r_{t+1} is the immediate reward gained from taking action a_t in s_t with respect to π and $\gamma \in [0, 1]$ is the discount factor. If $\gamma = 0$, computation only considers immediate rewards whereas if $\gamma = 1$, all future values are taken into account equally as if they are gained at that time step. Mathematically, aim is to generate a policy π maximizing $V^{\pi}(s), \forall s \in S$.

2.1.1.1 Evaluating MDPs and Finding an Optimal Policy

In order to generate an optimal policy, the first thing we need to do is evaluating the value function given in equation 2.2.

$$V^{\pi}(s) = E_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t], \tag{2.3}$$

The transition probability function $P(s'|s,a)$ and $R(s,a)$ can be inserted into Equation 2.3 and it can be rearranged in a recursive form, therefore;

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

Equation 2.4 represents the Bellman equation. Similarly to the value function, action-value function can be defined, represented as $Q : S \times A \rightarrow \mathbb{B}$:

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

Q function represents the expected accumulated value of the state s when action a is taken.

In order to find the optimal policy, we need to compute the maximum values, i.e. the optimal value function $V^{\pi^*}(s)$, for every $s \in S$. The optimal value function is represented as V^* which is the solution of the Bellman equation [1],

$$V^*(s) = \max_{pi} V^{\pi} = \max_{a \in A} Q^*(s,a), \forall s \in S$$
$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s'), \forall s \in S, a \in A$$

Therefore, optimal policy π^* is the policy with value function: V^*. For an MDP, a deterministic optimal policy can always be generated.

DP-oriented algorithms are widely used to efficiently generate optimal policies. They store intermediate value results in the numerical approximation process and converge

to optimal value function. Optimal policy can be generated by taking the action that has the highest state-action value for each state.

Most important algorithm is *Value iteration (VI)* which stores a table containing values for every state-action pair, which is an estimate of V^* [1]. At each iteration, it updates the values as following;

$$V_{k+1}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s') \right]. \tag{2.6}$$

where it will continue until a predefined stopping criteria is satisfied. VI is detailed in Algorithm 1.

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 $|\delta| < \varepsilon$

Up to this point, we assumed agent has full observability of the environment dynamics such that agent can perceive all of the state-action (and reward) relationship, i.e. the probability distribution of next state depending on current state, and reward information, to generate desired optimal policy. However, if we make the setting more realistic by adding uncertainty to the state knowledge of the agent, i.e. sensor error etc., it is not fully known which the state agent is in with $\%100$ probability. Hence, for partially observable settings, a new model is needed for agents that operate in those environments.

2.1.2 Partially Observable Markov Decision Processes

In real world problems, there almost always exists uncertainty, even though small, about the environment. In order to address these realistic scenarios, *Partially Observable Markov Decision Process (POMDP)* has been developed. A POMDP is a tuple [28]:

$$M =< S, A, T, R, \Omega, O, b_0 > \tag{2.7}$$

where ;

- S, A, T, R, define the underlying MDP
- Ω, is the set of all possible observations
- O, represents the observation probability measure, $O(o, a, s') = Pr(o_{t+1}|a_t, s_{t+1})$, which is the probability of receiving observation o after taking action a at time-step t then reaching state s'
- b_0, is the initial assignment to belief.

Agents in partially-observable environments take actions according to two approaches. First, the agent can choose it's immediate action based on complete past action-observation history and current observation. However, this approach has exponential complexity in policy representation and generation in terms of $|O|, |A|$, i.e. total number of possible observations and actions at each stage: At a time step t, a given policy, π maps history to an action, i.e. $\pi : (|O|^t \times |A|^{t-1} \rightarrow A)$.

Second, they can implement a memory which stores information about which state they can be in depending on action-observation history, which is called *belief*. Belief defines a distribution over the state set S encoding the summary of agent's fuzzy knowledge of which state it might be in. With belief states, the process becomes Markovian from Non-Markovian, since belief state is a sufficient statistic of the past and all beliefs reside in $|S| - 1$ dimensional simplex. Intuitively, one can imagine this addition as conversion from POMDP over S to and MDP over belief space. Whenever

agent takes an action a and receives and observation o, current belief state is updated according to Bayes' rule:

$$b(s') = \frac{P(o|s',a)}{P(o|b,a)} \sum_{s \in S} P(s'|s,a)b(s) \quad \text{where,}$$
$$P(o|b,a) = \sum_{s' \in S} P(o|s,a) \sum_{s \in S} P(s'|s,a)b(s)$$

In this case, domain of policy does not grow as time step progresses. However, belief distribution comes with a great complexity cost too, since space of all probability distributions over states is huge and continuous (uncountable) even though states are discrete.

2.1.2.1 Evaluating POMDPs and Finding an Optimal Policy

As a result of using belief states instead of current state, state value function V, state-action value function Q and policy function π are redefined as follows:

- Policy function $\pi : \mathbf{B} \rightarrow \mathbf{A}$ maps a belief state to an action.
- State Value function $V : \mathbf{B} \rightarrow \mathbf{R}$ maps a belief state to its total expected reward.
- State-action Value function $Q : \mathbf{B} \times \mathbf{A} \rightarrow \mathbf{R}$ maps a belief state and an action pair to its total expected reward.

Solution methods for POMDP in the literature can be split into two different categories; the first one is exact-value iteration methods which includes *Witness algorithm* [15], *One pass algorithm* [28, 20], *Enumeration algorithm*[29] and *Incremental Pruning* [16].

The exact-value update of value function can be written as:

$$V'(s) = R(s, \pi_V(s)) + \gamma \sum_{s' \in S} T(s, \pi_V(s), s')V(s') \tag{2.8}$$

The overall complexity of a single iteration of exact value update is: $\mathcal{O}(|V| \times |A| \times |\Omega| \times |S|^2 + |A| \times |S| \times |V|^{|\Omega|})$. Given a value function V, a greedy action policy can be derived using the following equation:

$$\pi_V(S) = \arg\max_{a \in \mathbf{A}} R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \tag{2.9}$$

Since each value function V_t is piece-wise linear and convex [21], we can implicitly represent the value function as a set of linear functions. The linear functions that make up the value function are called α -vectors, $V = \alpha_1, ..., \alpha_n$. Thus, we can compute the value at belief state b as follows:

$$V(b) = \max_{\alpha \in V} b.\alpha \tag{2.10}$$

where the worst case complexity grows exponentially in observation space which makes it extremely prohibitive for practical uses. However, not all alpha vectors will contribute to the value function, only a small subset of non-extraneous alpha vectors will actually be used due to the max operation in equation 2.10. To address this issue, two different approaches have been proposed in the literature in order to reduce the computational complexity of the exact-value iteration solution methods: in the first approach, a large number of alpha vectors are generated as the value function is updated, then the extraneous alpha vectors are pruned. This approach was proposed in several studies [29, 30] and it works efficiently when the number of possible observations is small. However, in the second approach which is usually referred to as relaxed region approach, we start with a small set of non-spurious set of alpha vectors, then we add non-extraneous alpha vectors one by one until no more non-extraneous policy trees exist. This approach has been pursued in several works such as algorithms proposed in [15] and [28]. Another important study, [16], introduces an exact solution method to POMDP's called *Incremental Pruning*. In comparison to other famous exact solution methods based on DP such as *Witness* and *Exhaustive*. Results show that, Incremental Pruning outperforms the other methods on various problems with different sizes. The Incremental Pruning algorithm consists of two main steps which are repeatedly computed until optimal value function is computed, thus optimal policy is computed as well: *(i)* update the set of alpha vectors, *ii* prune the updated set to its

minimal size.

The other category of solution methods is called; approximate-value update methods which calculate approximate value-iterations instead of exact value-iterations. They scale very well with the state-action space. That is based on the fact that performing a large number of the computationally inexpensive approximate value-iterations yields a better value function compared to performing a few exact value-iterations. As an example, [31] presents an approximate solution method to POMDP's named *HSVI*. HSVI combines heuristic search and value iteration. It also maintains and updates both an upper and a lower bounds on the piecewise linear value function. The lower bound is represented as a set of alpha vectors while the upper bound is represented as a set of belief points. The main operation of the algorithm is performing local updates at belief points thus improving both the upper and lower bounds. The belief point to be updated is chosen using heuristic search methods that explores forward in the search tree.

Most important approximate solution methods are Point-based methods [19]. These algorithms exploit from factored representations of POMDPs and try to overcome the difficulties arose from belief state updates [32, 33]. Details of point-based studies are presented in the next sub-section.

At this point, we will turn our attention to complexity problems of PO/MDPs arose from problems with huge state spaces and solution proposals for scaling of such processes.

2.2 Factorization of MDPs & POMDPs

In this section, we described various approaches and studies to address scalability problem of MDP and POMDP formulations.

2.2.1 Graphical Models for Compact Representation of MDPs/POMDPs

In MDP, state space complexity of the given domain can grow prohibitively large for problems of practical applications. As a result, defining $O(|S|^2)$ sized transition matrices for each action and a $O(|S|)$ vector of expected rewards may be quite costly and unnecessary. Authors in study [2], built a Dynamic Bayesian Network (DBN) based model to factor the transition matrices into a stochastic temporal process [34] represented as graphs reflecting the relationships between state variables (rather than states) for compact representation. In this framework, a DBN is defined with state variables for each action which shows the effects of a given action to state variables. This network is a 2-slice structure due to Markovian assumption showing the transitions of state variables. Each node denotes a state variable with temporally succeeding state variables emitting a conditional probability distribution.

As an example, a problem called *coffee task* is formulated in [2]. Here is the list of state variables presented in the problem:

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

For this action, it can be observed that the value of HU is effected by L, HU and HR for the action $deliver\ coffee$ with no influence from the rest of the variables.

This approach is also extended for POMDPs which carries utmost importance since policy generation for POMDPs became infeasible after tens of states. In [35], authors proposed to use Bayesian networks and influence diagrams to compactly represent POMDPs by exploiting the structure of the problem and the independence between state-variables. Although that is a powerful way of compact modeling, our model

approaches in a different perspective enabling even fine-grained factorization using tools of propositional logic. To generate optimal policy authors, introduce Monahan's solution method to POMDP's which uses the explicit representation of states and make use of Sondik's finding that value-functions are piece-wise linear and convex to find optimal policies using α vectors [29, 28]. The algorithm iteratively generates sets of α vectors then prunes the dominated ones. It describes how the previous algorithm can be modified and applied to the proposed representation of POMDP's using Bayesian nets. The idea is to use *Algebraic Decision Diagrams (ADDs)* to represent value functions [36, 37]. They generate α trees rather than α vectors and apply a modified version of the Monahan's solution method.

Another theoretical study, proposes a new approximate representation for value functions of a factored POMDP as a linear combination of basis functions [38]. It then proceeds to explain how to use the proposed approximate value function with value-iteration and policy-iteration algorithms in order to solve factored POMDP.

There are other studies that uses ADDs as factorization [39, 40, 19]. In study [39], authors presented a DP based algorithm for generating policy for factorized POMDPs represented as ADDs. Their proposed algorithm is based on Incremental Pruning to update value functions which has two main steps. It first performs value iteration using one step back up then pruning on the set of piece-wise linear function to delete dominated ones. One other study proposed an improvement over HSVI algorithm ([31], see previous sub-section) that can exploit factorization by ADDs.

On the other hand, as stated, most important solution methods are point-based value iteration algorithms [41] which gave birth to a whole new category that were developed and improved over the years [42, 43, 18, 44]. The main advantage of point-based solution methods is that they work with a subset of the belief space, restricting their search domain to the reachable belief states.

[42] introduces new theoretical measure for convergence that combines both considerations of the curse of dimensionality and history called discounted reachability. Based on the proposed theoretical work, it introduces two modifications to the previously studied algorithm HSVI called *HSVI1* and *HSVI2*. It compares the proposed variations and experiments with their corresponding performance.

Another point-based method related study introduces a novel point-based approximate solution method to POMDP's called *Perseus* [45]. What makes Perseus more efficient than other point-based solvers is that it computes backups for only a subset of the belief points into consideration, given that the computed solution will be effective for the entire set of belief points. At each backup iteration, Perseus randomly chooses a belief point, performs back up update for it then adds the new alpha vector to the solution set if it is an upper bound to the point's previous alpha vector. After that, it eliminates all belief points whose values has been improved by the recent updates from the set of belief points. It then randomly samples another belief point from the set of belief points until the set is empty. This algorithm outperforms PBVI and HSVI solution methods [41, 31]. One issue with the proposed algorithm is that after performing n backups, the obtained solution might not be that of n steps into the future given how the algorithm operates, it could be n or less than n steps into the future.

2.3 Abstraction & Hierarchical Structures

Other types of solutions to address the high state space complexity problem are abstraction and function approximation [46] which the former one works as coarsening the structure of the problem into larger clusters in terms of states, actions and transitions whereas the latter one proposes ways to represent value function compactly rather than matrix form. We will briefly introduce the concepts for abstraction perspective since they are mostly orthogonal to our approach. However, our model eases up abstraction & hierarchical models if it is needed. Before getting into abstraction, *Semi-Markov Decision Processes* need to be introduced.

Semi-Markov Decision Processes : Semi-Markov Decision Processes (SMDPs) generalizes MDPs by allowing multi-step transitions which will be fundamentally useful for hierarchical methods. Transition functions is encoded as below:

$$T \ : \ S \times A \times S \times \mathbb{R}^+ \to [0,1]$$
$$T(s,a,s',\tau) = P(t_{k+1} - t_k = \tau, s_{k+1} = s' \mid s_k = s, a_k = a)$$

Semi-Markovness comes from the fact that at times the decision of the agent may not depend only on the current state but also on the time passed from the last transition. Also, over the multi-step transition, multiple rewards can be obtained. Hence, with the reward function $\rho \ : S \times A \times \mathbb{R}^+ \to \mathbb{R}$, we have the following Bellman equation:

$$V^*(s) = \sup_{a \in A} \left[\sum_{\tau=0}^{\infty} \sum_{t=0}^{\tau} \gamma^t \rho(s,a,t+1) + \sum_{s' \in S} \gamma^{\tau+1} T(s,a,s,\tau) V^*(s') \right], \forall s \in S \tag{2.11}$$

Important portion of abstraction studies define sub-task routines that will enable decomposition of a task. The policies generated will be used over and over again wherever the same sub-task appears in the problem. With this higher level abstraction, the aim is allow agents to solve the original task more efficiently. There are several approaches in RL for defining and handling sub-task decomposition including Hierarchical Abstract Machines (HAMs) [47], Options [48], MAXQ [13] and MAXQ-OP [10].

The notion of *hierarchical decomposition* refers to the identification and execution of sub-tasks and *state abstraction*. Although, re-execution of those subroutines wherever needed is straightforward whereas the identification is a whole different issue. Those models try to address the problem of large state-action spaces by discovering structures within a given task, unlike our model which constructs a given task in a factored manner from ground up which in turn simplifies further engineering of abstraction. There are numerous landmark studies proposed to identify useful sub-tasks [8, 49, 7]. On the other hand, state abstraction is the process of structuring of state space by identifying them into certain clusters based on similarity where these groups can be treated as a single abstract state. Therefore, the computational cost for generating of a policy is decreased, whereas the optimal point for a policy can still be reached under certain conditions [12]. One study based on state abstraction notion approach

[50], proposed a new algorithm by finding partial state-symmetry and stacking states in abstract form getting the benefit of both compact representation and symmetrical abstraction [51, 52].

One other perspective of MDP decomposition process is to incorporate graph partitioning algorithms which mostly contrasts our work. In this approach, state transition graph of a given problem is partitioned via different approaches into sub-graphs to be solved independently. One of the important studies done in this perspective proposed the algorithm Variable Influence Structure Analysis (VISA)[3]. VISA algorithm is a hierarchical decomposition algorithm for factored MDPs being structurally represented with Dynamic Bayesian Networks (DBN) [2]. In a nutshell, like our objective, they first use a compact model in order to decompose it further in a more efficient manner. It uses causal information between states gathered from the DBN model to reveal strongly connected components in the state structure for constructing a graph with higher level abstraction. After, it uses exits [53] discovering the state-state relationships. The obtained causality graph is then used to define the necessary abstraction and subroutines to solve the sub-task defined. Some other studies use a different approach to partition the state-action transition graph which can be applied after building the proper model. In the study [54], authors incorporated a number of algorithms based on spectral theory and a complex network measure, eigenvector centrality. They proposed two algorithms to determine the sub-goals for a given task by revealing the underlying graph. After this procedure, they generated a number of options for each neighboring clusters to define macro actions. In another study [55], data mining algorithms are demonstrated for automated discovery of structural relationships emerged from the underlying state-action graph of a given task. State-action history is compiled as a data set for data-driven algorithms, then using state clustering and action trajectory mining macro actions/options are determined.

2.4 Non-Stochastic & Stochastic Planning

Our model is inspired from the important planning languages proposed. In the next sub-sections, will briefly introduce the concepts.

2.4.1 Planning Domain Definition Language (PDDL)

PDDL is a formal language for representing non-stochastic planning [23, 24]. A PDDL planning domain consists of various elements including a set P of predicates and a set AS of action schemata.

Predicates are mappings from PDDL objects to boolean variables whereas functions are mappings from PDDL objects to numeric state variables. Our example domain in Figure 2.1 has the predicates W, U, R, L, HU, HR defined in the coffee task before.

Actions in PDDL defines state transitions, i.e. changing values of a certain number of state variables. An action definition contains *preconditions* which dictates certain values of some state variables for enabling the action to be executed and an *effect* section which updates values of certain state variables. These elements of actions, although constructed in a probabilistic fashion, are two of the crucial elements for our proposed model.

For detailed specifications of PDDL, including problem definitions and possible values for *requirements* tag, refer to [23, 24].

2.4.2 Probabilistic Planning Domain Definition Language (PPDDL)

Probabilistic Planning Domain Decision Language (PPDDL) is proposed as a probabilistic extension to PDDL, which can represent MDPs [25]. In this model, *probabilistic effects* are defined. Moreover, *rewards* can be declared via the existing PDDL element named *fluents* [24].

The syntax for the probabilistic effects is as follows:

$$(probabilistic\ p_1\ e_1\ p_2\ e_2\ ...\ p_k\ e_k), \tag{2.12}$$

where e_i denotes the effects whereas p_i denotes the probabilities of the effects where $\forall i, p_i \geq 0$ and $\sum_{i=1}^{k} p_i = 1$. Any probabilistic effect must declare a set of probability-weighted outcomes under the restrictions. Figure 2.1 illustrates the action with probabilistic effects. Note that, the requirements flag $:probabilistic-effects$ signals

```
(define (domain coffee)
(:requirements :probabilistic-effects
:negative-preconditions
:disjunctive-preconditions
:conditional-effects :mdp)
(:predicates (w) (u) (r) (l) (hu) (hr))
(:action deliver-coffee
:precondition (when (hr))
:effect (and (when (l))
(probabilistic 0.8 (hu)))
(when (user-has-coffee)
(increase (reward) 0.9))
(when (not (is-wet))
(increase (reward) 0.1))))
...)
```

Figure 2.1: *coffee* domain

that probabilistic effects are used in the represented problem. Anyone seeking full descriptions and details can refer to [25].

On the other hand, *rewards* are encoded using *fluents*. The syntax of the reward fluent is given as:

$$(\langle additive\text{-}op \rangle \ \langle reward\text{-}fluent \rangle \ \langle f\text{-}exp \rangle) \tag{2.13}$$

where $\langle additive - op \rangle$ denotes an $increase$ or a $decrease$ and $\langle f - exp \rangle$ is a numeric value denoting the amount of reward taken. Figure 2.1 illustrates the rewards in PPDDL for the *coffee* task extended with probabilistic effects. For tasks from domains having the $:rewards$ property stated in the goal statement, default aim of the plan is to maximize the expected reward.

Figure 2.2 shows that the initial conditions can be probabilistic. In coffee example, two possible initial states is defined with equal probability (0.5) of being true. Every state variable is assigned to $false$ except l to $true$ and r to equal chance of $true$ and

```
1        (:init (probabilistic 0.5 (r)
2        0.5 (not (r)))
3        (l))
```

Figure 2.2: PPDDL declaration of initial state assignments: *coffee* task

$false$. For initial states, variables and their corresponding possible values should also be determined in the beginning.

In our model, we integrated a factorized model of actions and rewards inspired from PPDDL, although, some of the facilities of PPDDL are not so trivial to be converted to a regular MDP to generate an optimal policy with an efficient DP-based algorithm. Partly due to this reason, PPDDL is not evolved and benchmark solvers aren't proposed over the years. In the light of these, we started somewhat a more compact and theoretical automaton with a corresponding benchmark solver and allow other extensions that can be engineered within the model in the following studies.

CHAPTER 3

CONTEXT-AWARE PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

Our attempt to address the problems occurred due to huge state spaces is to take the state-action relationships into account while formulating the task. In important portion of the real world applications, changes occurred by state-action transformations generally reside on the local portion of the state space. By capturing these local regularities, a much more compact representation can be obtained. This will also help reflecting the structural relationships which in turn makes further engineering aspects (e.g. hierarchical models) easily applicable for a given task to reduce the computational burden to generate an optimal policy. Hence, localized or independent structures residing on a state-action space of a given task will be emitting by our model which will lead the way for efficient learning of policies. Furthermore, it can make top-down decomposition easier to employ for alleviating the computational complexity of obtaining solutions.

This complexity shows itself much more heavily in partially observable environments. As stated in earlier section, due partial information, agents need to maintain a belief state to keep track of possible states it might be in in the form a probability distribution. If belief state distribution emits high entropy then most of the engineering perspectives fail and even near optimal policy becomes infeasible when state space grows moderately large. Keeping this in mind, we try our best to keep partially observable formulation in a structured form reflecting relationships to keep transitions as localized as possible as long as task permits.

In the following sections, we present MDP based context-aware models for stochastic planning.

3.1 Context-Aware Markov Decision Processes

In this section, we point out some specifics of an earlier model, Context-Aware Markov Decision Processes (CA-MDP), defined as the following [56]:

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

where ;

- Γ, is a set containing domains of state variables,
- S, is a set of state variables,
- A, is a set of actions,
- C, is a set of boolean statements defining proper conditions to execute actions,
- E, is a set of effects of actions,
- I, is the initial assignment to state variables.

For a problem formulated as a CA-MDP, an actual state is defined as the conjunction of all state variables with a value assigned to each of them from their respective domains. Actions are defined in the set A and C defines the preconditions for execution of each action. A state must satisfy those conditions for agent to take the desired action in that state. Set of effects, E, contains action specific state variable transitions and rewards is defined in a grammar form. Finally, I defines the initial state for the problem. Note that there can be more than one possible initial state. For more specific details and illuminating examples check [56].

Now, we can turn our attention to the partially observable case which is the core formulation of this thesis.

3.2 Context-Aware Partially Observable Markov Decision Processes

Inspired by the proposed grammar in CA-MDP, and motivated by the problems due to space and time complexities of current models; we propose a new formulation to describe partially observable tasks, *Context-Aware Partially Observable Markov Decision Processes (CA-POMDP)*. The proposed CA-POMDP is described by the following tuple $M =< \Gamma, S, A, C, O, E, I, B >$.

- Γ, is a finite set of the finite domains of each state variable,
- S, is a finite set of state variables each with a restricted domain from Γ,
- A, is a finite set of all possible actions,
- C, is a finite set of conditions formulated as boolean statements,
- O, is a finite set of all possible observations and related classes,
- E, is a finite set of effects of actions,
- I, is an initial assignment of the problem which constitutes the initial belief state,
- B, is defined as a set of distributions over state variables representing belief of agent in factorized form.

Elements except O and B constitute the underlying CA-MDP for a given task. Again, actual states are conjunctions of state variables assigned with some value from its domain, i.e. $\bigwedge_{i=0}^{|S|-1}(s_i = v_i)$ where $s_i \in S,\ v_i \in \Gamma_i$ and $\Gamma_i \in \Gamma$. This piece-wise representation is highly beneficial when the problem designer needs to exploit the state-action dependencies.

Set C with $|C| = |A|$, consisting of boolean statements in disjunctive normal form, expresses the proper conditions, i.e. values of state variables, in order to execute the corresponding action. Each literal in the formula is an assignment to a state variable, i.e. $(s = v)$ where $s \in S, v \in \Gamma_i$ and $\Gamma_i \in \Gamma$. Conditionals are still valuable when uncertainty is low.

O is described as the set of all observations that the agent can get from the environment. In regular POMDP definition, observation probabilities are given as $O(o, a, s') =$

$Pr(o_{t+1}|a_t, s_{t+1})$ whereas in CA-POMDP definition, it is described in factorized form. In addition, it is structured in various classes to allow joint but mutually exclusive observations that can be obtained from one source, i.e. multiple valued observations rather than binary valued. This structure decreases certain probability update computations. As an example, if an agent can get one of the observation from the set $\{red, yellow, green\}$, we can bundle them together as a class named $light$ and define probabilities accordingly. The grammar is defined as follows:

$O_{action} = (\wedge \quad O_a)$

$O_a = \ (if\ (C)\ P)\ O_{action} \mid (if\ (C)\ O_{action}) \mid (if\ (C)\ P)$

$C = < conditional >$

$P = \ OP\ P \mid OP \mid \top$

$OP = \ (< obs\_value > = < probability >)$

The $<\ conditional\ >$ part of the if structure comprises of a boolean formula in disjunctive normal form stating the conditions to get the corresponding observation hence implementing the conditional observation probability. There are some important constraints on defining observational classes in this grammar form:

- for an observational probability in the form $((ov_0 = p_0)\ (ov_1 = p_1)\ ...\ (ov_n = p_n))$ where ov_i with $0 \leq i \leq n$ is a possible observational value for the given class and $\forall p_i \geq 0,\ \sum_{i=0}^{n} p_i = 1$ and if $\forall i, j \leq n, i \neq j \iff ov_i \neq ov_j$.
- for multiple valued observations that have positive probability for non-emission, e.g. $\{hungry\ /\ non-hungry\}$, $\{red\ /\ yellow\ /\ green\ /\ no-light\}$, with some probability $1 - \sum_{i=0}^{n-1} p_i$, this value and corresponding probability can be easily expressed with the $\top$ symbol in the grammar.
- this grammatical structure does not define any constraints on $<\ conditional\ >$s

to prevent ambiguity which are designer's responsibility to avoid incoherencies like defining nested if declarations that are not logical or constructing if structures that has the potential to state multiple probabilities for one observational value.

Effects set E has the same structure as in CA-MDP [56] which reflects the transitions and rewards of the underlying fully observable process. I is the initial assignments to state variables reflecting the real initial state of the problem, i.e. the tuple $((s_0 = v_0) \wedge (s_1 = v_1) ... \wedge (s_{|S|-1} = v_{|S|-1}))$ where $v_i \in \Gamma_i, s_i \in S$. If a problem has more than one possible initial states, I can be in disjunctive normal form. Then, initial belief state assignment can be done accordingly by choosing one of the initial states before problem beginning.

Finally, we need to construct beliefs over state variables rather than states in a factorized approach which is detailed in the following subsection.

3.2.0.1 Belief State Representations & Updates

In CA-POMDP formulations, beliefs are represented as distributions over possible values of state variables, hence actual belief distribution over states is the joint distribution of the variables. Hence, set B, with $|B| = |S|$, consists of beliefs b_i with $i \in \{0, 1, ..., |B| - 1\}$ where each $b_i \in B$ is a discrete probability distribution defined over the domain $\Gamma_i \in \Gamma$ of a state variable $s_i \in S$. Here is the Bayesian update for the belief of a state variable $s \in S$ with a value $v \in \Gamma_s$ in its respective domain when action a is taken and observation o is observed:

$$b^{t+1}(s = v) = P(s = v | o, a, b^t(\mathbf{s}))$$
$$b^{t+1}(s = v) \propto \sum_{\mathbf{s}'} P(o \mid \mathbf{s}'|_{s=v}, a) \sum_{\mathbf{s}} P(\mathbf{s}'|_{s=v} \mid \mathbf{s}, a) b^t(\mathbf{s})$$

where $\mathbf{s} = \bigwedge_{i=0}^{|S|-1}(s_i = v_i)$ and $\mathbf{s}' = \bigwedge_{i=0}^{|S|-1}(s_i = v_i')$ are actual states and the normalization factor is:

$$\sum_{u \in \Gamma_s} \sum_{\mathbf{s}'} P(o \mid \mathbf{s}'|_{s=u}, a) \sum_{\mathbf{s}} P(\mathbf{s}'|_{s=u} \mid \mathbf{s}, a) b^t(\mathbf{s})$$

However, this update necessitates state enumeration and results in computation of lots of irrelevant probabilities. So, in order to fully gain the power of factorization for update computation, grammars of observations, effects and conditionals need to be used. To illuminate the concept further, we present an example here by formulating a problem as a CA-POMDP.

3.2.1 Problem

This rather theoretical problem illustrates the power of CA-POMDP representation when the problem enables factorization. This task is declared with 4 binary state variables, 4 actions and 3 binary and 1 ternary observation with values $\{red, green, yellow\}$, and the goal is $s_3 = 1$.

- $\Gamma = \{\{0,1\},\{0,1\},\{0,1\},\{0,1\}\}$
- $S = \{s_0, s_1, s_2, s_3\}$
- $A = \{a_0, a_1, a_2, a_3\}$
- $C = \emptyset$
- $O = \{\mathbf{o1_{a_0}} = (\wedge \quad (if(s_0 = 0 \wedge s_1 = 0) \quad (true = 0) \quad \top)$
  $(if(s_0 = 1 \wedge s_1 = 0) \quad (true = 0.1) \quad \top)$
  $(if(s_0 = 0 \wedge s_1 = 1) \quad (true = 0.2) \quad \top)$
  $(if(s_0 = 1 \wedge s_1 = 1) \quad (true = 0.3) \quad \top)),$
  $\mathbf{o2_{a_1}} = (\wedge \quad (if(s_1 = 0) \quad (true = 0.2) \quad \top)$
  $(if(s_1 = 1) \quad (true = 0.8) \quad \top)),$
  $\mathbf{o3_{a_1}} = (\wedge \quad (if(s_1 = 0 \wedge s_2 = 0) \quad (true = 0) \quad \top)$
  $(if(s_1 = 1 \wedge s_1 = 0) \quad (true = 0.4) \quad \top)$
  $(if(s_2 = 1) \quad (true = 0.2) \quad \top)),$
  $\mathbf{o4_{a_2}} = (\wedge \quad (if(s_2 = 0) \quad (red = 0.8) \quad (green = 0.1) \quad (yellow = 0.1))$
  $(if(s_2 = 1) \quad (red = 0.1) \quad (green = 0.8) \quad (yellow = 0.1)))\}$

- $E = \{\mathbf{E_{a_0}} = (\wedge\, (if(s_0 = 0 \wedge s_1 = 0) \quad (\vee\, ((s_0 = 1)\, 0.6) \quad \top))$
  $(if(s_0 = 0 \wedge s_1 = 1) \quad (\vee\, ((s_0 = 1)\, 0.5) \quad \top))$

$(if(s_0 = 1 \wedge s_1 = 0) \quad (\vee\,((s_0 = 1)\,0.6) \quad \top))$
$(if(s_0 = 1 \wedge s_1 = 1) \quad (\vee\,((s_0 = 1)\,0.3) \quad \top))),$
$\mathbf{E_{a_1}} = (\wedge\,(if(s_1 = 0 \wedge s_2 = 0) \quad (\vee\,((s_1 = 1)\,0.5) \quad \top))$
$(if(s_1 = 0 \wedge s_2 = 1) \quad (\vee\,((s_1 = 1)\,0) \quad \top))$
$(if(s_1 = 1 \wedge s_2 = 0) \quad (\vee\,((s_1 = 1)\,0.5) \quad \top))$
$(if(s_1 = 1 \wedge s_2 = 1) \quad (\vee\,((s_1 = 1)\,1))),$
$\mathbf{E_{a_2}} = (\wedge\,(if(s_0 = 0 \wedge s_2 = 0) \quad (\vee\,((s_2 = 1)\,0.2) \quad \top))$
$(if(s_0 = 0 \wedge s_2 = 1) \quad (\vee\,((s_2 = 1)\,0) \quad \top))$
$(if(s_0 = 1 \wedge s_2 = 0) \quad (\vee\,((s_2 = 1)\,0.9) \quad \top))$
$(if(s_0 = 1 \wedge s_2 = 1) \quad (\vee\,((s_2 = 1)\,1) \quad \top)))\}$
$\mathbf{E_{a_3}} = (\wedge\,(if(s_2 = 0 \wedge s_3 = 0) \quad (\vee\,((s_2 = 1)\,0) \quad \top))$
$(if(s_2 = 0 \wedge s_3 = 1) \quad (\vee\,((s_2 = 1)\,1) \quad \top))$
$(if(s_2 = 1 \wedge s_3 = 0) \quad (\vee\,((s_2 = 1)\,0.8\,reward\,10) \quad \top))$
$(if(s_2 = 1 \wedge s_3 = 1) \quad (\vee\,((s_2 = 1)\,1) \quad \top)))\}$

- $I = \{s_0 = 0 \wedge s_1 = 0 \wedge s_2 = 0 \wedge s_3 = 0\}$
- $B = \{P(s_0 = 1) = 0, P(s_1 = 1) = 0.5, P(s_2 = 1) = 0, P(s_3 = 1) = 0\}$

In this formulation, B symbol summarizes the initial belief state, b^0, compliant with the initial state variable assignment, although the agent isn't fully sure which value is assigned to s_1 variable. Here are the computations for belief updates of some of the state variables when :

1. action $\mathbf{a_0}$ is executed and observation $\mathbf{o_1} = 1$ is taken

$$
\begin{aligned}
b^{t+1}(s_0 = 1) = P(s_0^{t+1} = 1 | o_1 = 1, a_0, b^t(s_0, s_1, s_2, s_3)) \\
&\propto \sum_{v' \in \{0,1\}} P(o_1 = 1 | s_0^{t+1} = 1, s_1^{t+1} = v', a_0) \quad \times \\
&\quad \sum_{u,v \in \{0,1\}} P(s_0^{t+1} = 1 | s_0^t = u, s_1^t = v, a_0) P(s_1^{t+1} = v' | s_1^t = v, a_0) \quad \times \\
&\quad b^t(s_0 = u) b^t(s_1 = v)
\end{aligned}
\tag{3.2}
$$

   The factorization of observation and transitions can be gathered from o_1 and E_{a_0} which the observation outcome if only effected by s_0 and s_1 state variables and in turn they are temporally effected from s_0, s_1 and s_1 respectively.

2. action a_1 is executed and observation $o_2 = 1$ is taken (hence $o_3 = 0$ is taken),

$$
\begin{aligned}
b^{t+1}(s_1 = 1) &= P(s_1^{t+1} = 1 | o_2 = 1, o_3 = 0, a_1, b^t(s_0, s_1, s_2, s_3)) \\
&\propto (P(o_2 = 1 | s_1^{t+1} = 1, a_1) \sum_{v,y \in \{0,1\}} P(s_1^{t+1} = 1 | s_1^t = v, s_2^t = y, a_1) \times \\
& b^t(s_1 = v) b^t(s_2 = y)) \quad \times \\
& (\sum_{y' \in \{0,1\}} P(o_3 = 0 | s_1^{t+1} = 1, s_2^{t+1} = y', a_1) \quad \times \\
& \sum_{v,y \in \{0,1\}} P(s_1^{t+1} = 1 | s_1^t = v, s_2^t = y, a_1) P(s_2^{t+1} = y' | s_2^t = y, a_1) \quad \times \\
& b^t(s_1 = v) b^t(s_2 = y))
\end{aligned}
\tag{3.3}
$$

To make things clear in this case, the multiple observation probability $P(o_2, o_3 | s_0, s_1, s_2, s_3, a_2)$ can be factorized like the following:

$$
\begin{aligned}
P(o_2, o_3 | s_0, s_1, s_2, , s_3, a_2) &= P(o_2 | s_0, s_1, s_2, s_3, a_2).P(o_3 | s_0, s_1, s_2, , s_3, a_2) \\
&= P(o_2 | s_1, a_2).P(o_3 | s_1, s_2, a_2)
\end{aligned}
\tag{3.4}
$$

3. action a_2 is executed and observation $o4 = green$ is taken,

$$
\begin{aligned}
b^{t+1}(s_0 = 0) &= P(s_0^{t+1} = 0 | o_4 = green, a_2, b^t(s_0, s_1, s_2)) \\
&\propto P(o_4 = green | s_0^{t+1} = 0, a_2) \sum_{u,y \in \{0,1\}} P(s_0^{t+1} = 0 | s_0^t = u, s_2^t = y, a_2) \times \\
& b^t(s_0 = u) b^t(s_2 = y)
\end{aligned}
\tag{3.5}
$$

Before proceeding to next section, one can check next section for demonstration of actual state generation and conditionals in policy generation.

3.2.2 Families

Here, we give definitions for CA-POMDP families that we consider as important:

Definition: Let $M =< \Gamma, S, A, C, O, E, I, B >$ be a CA-POMDP, M is a member of **binary** CA-POMDPs if $\forall \Gamma_i \in \Gamma, |\Gamma_i| = 2$.

Definition: Let $M =< \Gamma, S, A, C, O, E, I, B >$ be a CA-POMDP, is a member of **uniconditional** CA-POMDPs if all the conditions in C are in conjunctive form i.e. $\forall i, c_i \in C$ has no disjunctive formulas. C can be represented as an $|A| \times |S|$ matrix as well.

Definition: Let $M =< \Gamma, S, A, C, O, E, I, B >$ be a CA-POMDP, M is a member of **unconditional** CA-POMDPs if $C = \emptyset$.

Definition: Let $M =< \Gamma, S, A, C, O, E, I, B >$ be a CA-POMDP, M is a member of **compound** CA-POMDP if there is more than one possible initial state denoted with disjunctive normal form in I. Only one of the possible initial states should be determined before execution.

Definition: Let $M =< \Gamma, S, A, C, O, E, I, B >$ be a CA-POMDP, M is a member of **singular** CA-POMDPs if $|S| = 1$.

CA-POMDP formulations which do not satisfy the definitions of any of the families mentioned can be referred as a **regular** CA-POMDP.

3.2.3 Conversion: POMDP $\longleftrightarrow$ CA-POMDP

To demonstrate that the representative power of CA-POMDP by means of compactness, we will detail transformation from POMDP to CA-POMDP or vice versa in the following subsections.

A POMDP formulation is composed of an MDP with a set of observations, thus the conversion between POMDP $\longleftrightarrow$ CA-POMDP is similar to the conversion between MDP $\longleftrightarrow$ CA-MDP presented in [56]. In a nutshell, states can be converted into state variables, state transitions and rewards can be encoded into effects grammar or vice versa. State variables of CA-POMDP can be converted to states by defining a state in POMDP for each of the possible actual states which can be generated by taking tensor product of all state variables with their all possible values. Reverse direction can be done in various ways one of which is constructing a binary state variable for each

state with a 1 value to indicate the agent is in that state and vice versa. It is designer's responsibility to make state mutual exclusive. Another way is to define a single state variable with a number of values equal to state number. Each value can indicate a state since states are mutual exclusive in POMDP. The effects can be converted to transitions by generating the state-action graph according to the grammar and taking tensor products of state variable transition probabilities to generate actual transitions between states with respect to given actions. Reverse direction can be done by generating effects from transitions probabilities which is a straight-forward transformation, i.e. define a probability from one state variable to another in effects since they need to be mutual exclusive. Thus, to convert a POMDP into a CA-POMDP, the underlying MDP is converted to a CA-MDP, then the observations are converted in linear-time in terms of $|O|$. The same process applies when converting a CA-POMDP into a CA-MDP as well.

Theorem 3.2.1. *Complexity for transformation : POMDP $\longrightarrow$ CA-POMDP*

There exists a transformation algorithm transforming POMDP $P =< S, A, T, R, O >$ to a binary unconditional or singular unconditional CA-POMDP $C =< \Gamma', S', A', C', E', O', I' >$ with polynomial time complexity in terms of $|S|$, $|A|$ and $|O|$ [1].

Proof of theorem 3.2.1. The algorithm with the desired complexity can be constructed as following:

1. Transform underlying MDP M of P to a binary or singular unconditional CA-MDP C
2. Transform O into O'

The first transformation is shown to be done in polynomial time in terms of S and A. Conversion for observation set can be done in O($|O|$) time, completing the proof.

□

[1] $|O|$ refers to the cardinality of probability distribution table entries

Theorem 3.2.2. *Complexity for transformation : POMDP ⟵ CA-POMDP*

There exists a conversion algorithm transforming CA-POMDP $C =< \Gamma, S, A, C, E, O, I >$ *to a POMDP with* $P =< S', A', T', R', O' >$ *polynomial to exponential time computational complexity in terms of* $|\Gamma|$, $|S|$, $|A|$, $|O|$.

Proof of theorem 3.2.2. The algorithm with the desired complexity can be constructed as following:

1. Transform underlying CA-MDP M' of C to MDP M
2. Transform O into O'

If the CA-POMDP is a compound CA-POMDP, i.e. multiple possible states, then it can be done in the same way in the new regular POMDP formulation. The first transformation is shown to be done in linear to exponential time in terms of $|S|$ and $|A|$. Similar to first conversion, second conversion for observation set can also be done in linear time in terms of $|O|$ [2] and up to exponential time in terms of $|S|$, since body of an observation class can be exponential in terms of $|S|$, completing the proof.

□

[2] $|O|$ refers to the cardinality of set of observation classes

CHAPTER 4

A SOLUTION METHOD FOR FULLY OBSERVABLE ENVIRONMENT

In this chapter, we proposed a solution method for CA-MDP improving the one proposed in earlier work [56] and gave a broader theoretical perspective and analysis. The solution mechanism benefits from the factored state form and opens the way for more engineering-wise enhancements like hierarchical approaches.

4.1 Solving a CA-MDP : Actual State Representation and Conditionals

For CA-MDP formulations except singular CA-MDPs, since we could not directly infer the transition probabilities of actual states given by transition probabilities of individual state variables, embedded in effects, generation of transition probabilities of actual states can be accomplished via tensor products. For a given CA-MDP formulation, the transitions can be represented as tensors. For a given an actual state comprised of m state variables, equivalent vector notation of logical conjunctions;

$$\begin{aligned} \boldsymbol{s} &= ((s_1 = v_1) \ \wedge (s_2 = v_2) \ \ldots \wedge (s_m = v_m)) \\ &= [(s_1 = v_1) \quad (s_2 = v_2) \ \ldots \ (s_m = v_m)], \end{aligned} \tag{4.1}$$

with an effect of action declares leading to T different number of states, when the action is taken in that state, i.e. satisfying conditional c of action and some if clause(s) if any, which the probability distribution of transition of each of the individual state variable can be represented as random (column) vectors $\boldsymbol{t_1}, \boldsymbol{t_2}, ...\boldsymbol{t_n}$ whose probability entries being corresponded to some instantiation of state variables $\boldsymbol{s_1}, \boldsymbol{s_2}, ...\boldsymbol{s_n}$, with obviously $0 \leq n \leq m$, then we have the following transition probability distribution

:

$$\boldsymbol{P}(\boldsymbol{S'}|\boldsymbol{s},a) = \bigotimes_{i=1}^{n} \boldsymbol{t_i} \tag{4.2}$$

where ;

$$\begin{aligned} \boldsymbol{S'} &= \begin{bmatrix} \boldsymbol{s}'_1 \\ \boldsymbol{s}'_2 \\ \vdots \\ \boldsymbol{s}'_T \end{bmatrix} \\ \boldsymbol{s}'_{\boldsymbol{i}} &= \boldsymbol{s} \odot (\mathbf{I} - \mathbf{1}_{\boldsymbol{a}}) + \boldsymbol{s}_{\boldsymbol{i}}^{\otimes}\, \mathbf{D}_{\boldsymbol{a}} \\ \boldsymbol{s}^{\otimes} &= \bigotimes_{i=1}^{n} \boldsymbol{s_i} \end{aligned} \tag{4.3}$$

where the matrix $\mathbf{D}_a$ is a $n \times m$ design matrix being comprised of 0s and 1s which an entry $(\mathbf{D}_{\boldsymbol{a}})_{ij}$ is 1 if ith state variable in entries of $\boldsymbol{s}^{\otimes}$ is in jth entry in the vector $\boldsymbol{s}$. The mere function of this matrix is to match the dimensions of change of state variables given in entries of $\boldsymbol{s}^{\otimes}$ matrix to the actual state s. On the other hand, $\mathbf{1}_{\boldsymbol{a}}$ is an indicator vector which an entry is either 0 or 1 based on whether any change occurs in that state variable given the action: if a change occurs that entry is 1, 0 otherwise, which is exactly the sums of rows of design matrix, $\sum_{i=1}^{n}(\mathbf{D}_{\boldsymbol{a}})_i$. Also, $\mathbf{I}$ is the ones vector and the tensor product of two state variables are defined as:

$$\begin{aligned} \left(\boldsymbol{s_i} = \begin{bmatrix} (s_i)_1 = (v_i)_1 \\ (s_i)_2 = (v_i)_2 \\ \vdots \\ (s_i)_k = (v_i)_k \end{bmatrix}\right) \otimes \left(\boldsymbol{s_j} = \begin{bmatrix} (s_j)_1 = (v_j)_1 \\ (s_j)_2 = (v_j)_2 \\ \vdots \\ (s_j)_l = (v_j)_l \end{bmatrix}\right) \\ = \begin{bmatrix} ((s_i)_1 = (v_i)_1) \wedge ((s_j)_1 = (v_j)_1 \\ (s_i)_1 = (v_i)_1) \wedge ((s_j)_2 = (v_j)_2) \\ \vdots \\ ((s_i)_k = (v_i)_k) \wedge ((s_j)_l = (v_j)_l) \end{bmatrix} \end{aligned} \tag{4.4}$$

The generation of the states and transition probabilities can be online or offline. If offline generation is preferred, the actual states and corresponding transitions are stored in an appropriate data structure later to be used in the execution of policy generation algorithm, whereas if the other way is preferred, then every time the transitions are generated on the run, and desired value function updates are executed. Thus, this trade-off between space and time should be taken into account, depending on the engineering of the problem and the nature of solver algorithm.

4.1.1 CA-MDP Value Iteration (CA-MDP VI)

CA-MDP model, due to having conditionals, is highly suitable for stating the state-action relationships. With this task-related information at hand, a new algorithm named CA-MDP Value Iteration (CA-MDP VI) is developed for learning an optimal policy which is an improvement over the regular Value Iteration (VI) algorithm (given in Algorithm 1) in terms of efficiency by redefining updates.

In order to calculate $V(\boldsymbol{s})$ for a particular state $\boldsymbol{s}$, all of the transition probabilities for all of the actions for that state should be generated regardless of the problem structure. In the light of this fact, it can be inferred that VI algorithm is not *scalable* with respect to task's internal state-action space. Conversely, CA-MDP VI is a scalable one in this regard. CA-MDP elements helps the algorithm eliminating all inefficiencies in the computations stated earlier, hence making it a *scalable* algorithm. Higher level outline of CA-MDP VI is given in Algorithm 2.

In line 8 of the algorithm 2, the value function, $\boldsymbol{V}()$ is actually a (column) vector containing values of next possible states. One other important note is that, in some problems rather than $R(s, a)$, $R(s, a, s')$ formulation is given, so the update rule should be changed as;

$$Q(\boldsymbol{s}, a) := \boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, a)^T \left(\boldsymbol{R}(\boldsymbol{s}, a, \boldsymbol{s'}) + \gamma \boldsymbol{V}^k(\boldsymbol{s'})\right) \tag{4.5}$$

which $\boldsymbol{R}$ is vector containing rewards for corresponding state-action pairs defined in

Algorithm 2 Outline of CA-MDP Value Iteration algorithm

Given a CA-MDP, $M =< \Gamma, S, A, C, E, I >$
Generate actual states S_{actual}
Initialize value table, V^0
repeat
  $\delta := 0$
  for all $\boldsymbol{s} \in S_{actual}$ **do**
    for all $a \in A$ and a is permissible in s **do**
      $Q(\boldsymbol{s}, a) := R(\boldsymbol{s}, a) + \gamma(\boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, a)^T \ \boldsymbol{V}^k(\boldsymbol{s'}))$
      $V^{k+1}(\boldsymbol{s}) := \max(Q(\boldsymbol{s}, a), V^k(\boldsymbol{s}))$
    end for
    $\pi(\boldsymbol{s}) := argmax_a Q(\boldsymbol{s}, a)$
    $\delta := \max(\delta, |V^{k+1}(\boldsymbol{s}) - V^k(\boldsymbol{s})|)$
  end for
until $(|\delta| < \varepsilon)$

the problem.

Mathematically, inner for loop of of CA-MDP VI algorithm can be re-written as:

$$\begin{aligned} &\forall \boldsymbol{s} \in S_{actual}, \ \forall a \in A, \\ &\quad Q(\boldsymbol{s}, a) := \mathbf{1}_{\boldsymbol{s}}(c_a)[R(\boldsymbol{s}, a) + \gamma(\boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s}, a)^T \ \boldsymbol{V}^k(\boldsymbol{s'}))] \end{aligned} \tag{4.6}$$

where c_a is the conditional of the current action. Therefore, the expected number of updates of the Q-function of equation 4.6 given as;

$$\begin{aligned} E[\mathbf{1}_{\boldsymbol{s}}(c_a)|S_{actual}||A|] &= E[\mathbf{1}_{c_a}(\boldsymbol{s})](|S_{actual}||A|) \\ &\leq (|S_{actual}||A|) \end{aligned} \tag{4.7}$$

which shows that it is bounded by the number of iterations in VI algorithm.

Intuitively, CA-MDP VI algorithm gets rid of the inefficiencies in the loop of the regular VI algorithm due not exploiting the structural dependencies in a given task. The

loop of the VI algorithm iterates for every state-action pair regardless of the connectivity structure of the problem, on the other hand CA-MDP VI iterates only on the state-actions satisfying defined conditionals. The sparseness of the state-action graph of a task determines the time complexity of CA-MDP VI, whereas in VI rather than connectivity, number of states (nodes) and actions play the major role in determining complexity. If a state-action graph is sparsely connected or contains clusters i.e. clusters with small cut to distinguish one another, the number of redundant computations grow large if the connectivity is not taken into account. VI algorithm runs as regarding the state-action graph of a problem is fully connected, however CA-MDP VI considers the inherent relational structure of the task which the time complexity depends on the sparseness of the graph hence making it a scalable policy generator. As a result, the complexity of CA-MDP VI is much better than VI yet it increases and approaches to the complexity of VI as the structure of the problem got denser.

4.1.2 Experiments

In this section, a set of comparative experiments are conducted to demonstrate engineering of CA-MDPs and execution of CA-MDP VI. In the experiments, besides some of the most common problems in the reinforcement learning literature, in the light of the theoretical foundations above, we defined a couple of problems in order to reveal the gains over defining and solving problems using regular MDP and running VI. The CA-MDP formulations and/or state graphs of those problems can be found below. All of the corresponding equivalent MDP formulations are encoded in plain arrays and matrices in order to give a fair comparison of running times.

The experiments are performed in 5 tasks: A small sparse task with low number of states and actions [56], two hypercube tasks having sparse and dense transitions of the same state space, the coffee task [2], the taxi task [13] and the SysAdmin task [**?**]. Small task is an abstract problem with small state-action space having 7 actual states and 4 actions. In this problem, the agent needs to reach the goal state from the initial state defined. For the purpose of defining two different variations of the same problem to empirically show the growth rates of solver algorithms in a comparative manner, we define a new task called as hypercube. In hypercube task, there are n

m-ary state variables for representing coordinates in a hypercube environment and an additional variable for indicating whether the agent has certain cypher or not. Goal is defined as exiting the cube by first getting cypher(s) from specified node(s) then finding its way to exit node, which is just a regular safe node, getting a positive reward, however in the cubes, some cells are safe whereas some cells are hazardous for the agent, resulting in penalty if stopped by. In addition to those dangerous nodes, there may be dead-end nodes which the agent should stay away from. In this study, we have $3 + 1$ binary state variables. The agent has 6 actions for teleportation from one cell to another. To accomplish its task, agent should get a particular cypher in one of the cells to exit the hypercube when the exit cell is reached. In the next coffee problem, agent needs to deliver coffee to the user. There are 6 binary state variables and 4 actions defined. In order to accomplish its task, it has to go to coffee shop and bring it back to user in office and as a side task it has to pick up the umbrella if it is a rainy day. In taxi task, the goal is the pickup passengers in certain locations and deliver to predefined destinations. It has 6 state variables and actions. The agent gets positive reward if it picks up and delivers from passenger location to destination, however there is a penalty involved in the case of wrong pick up or put down. In the SysAdmin problem, there are predetermined number of connected machines along their connectivity topology. Machines can be in two states: either running or failure state. Each running machine can fail in the next time step with a given probability. These failure probabilities significantly increase if other machines being connected also in failure state for the given time step. Purpose of SysAdmin is to decide which machine to reboot. Hence, action space is comprised of rebooting one of the machines or no operation (*noop*). SysAdmin gets predetermined amount of reward for each running machine in the system.

The small sparse problem, visualized in Figure 4.1 for understanding possible transitions better, is for demonstrating performance gains even if the problem size is small if structural information is exploited. We defined encoded as singular CA-MDP, formulated as; $Small = < \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0\}, \Gamma_0 = \{0, 1, 2, 3, 4, 5, 6\}$,
- $S = \{s\}$,

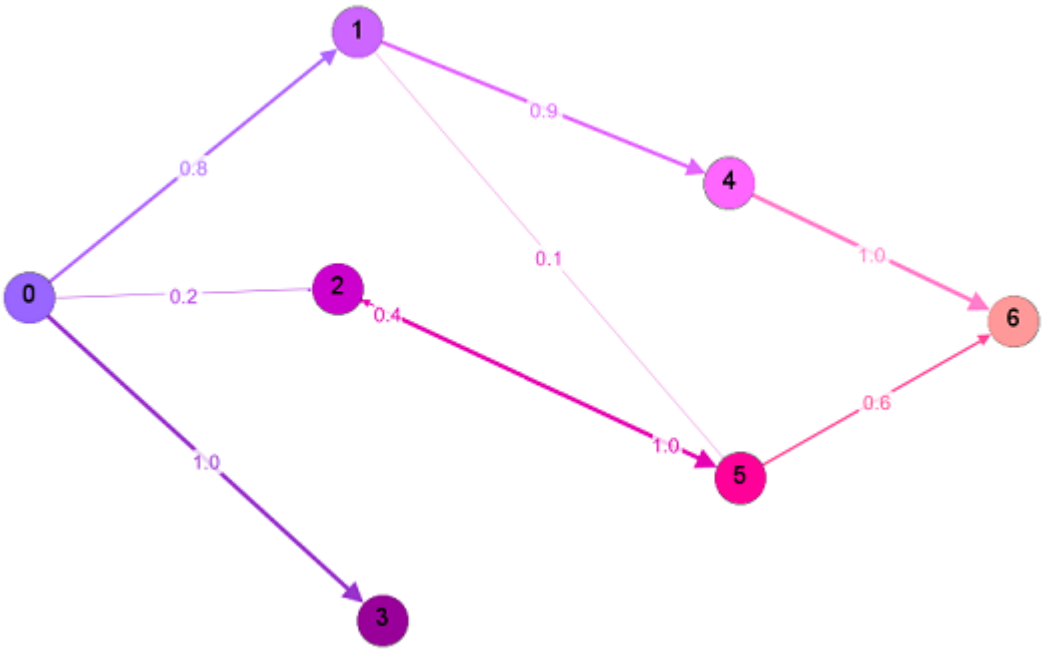


Figure 4.1: State-action transition graph of $Small$

- 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 = $\{\boldsymbol{E_{a_0}} = (\vee \quad ((s = 1)\ 0.8)\ ((s = 2)\ 0.2)\ reward\ 20)$,

  $\boldsymbol{E_{a_1}} = (((s = 3)\ 1)\ reward\ -100)$,

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

  $\boldsymbol{E_{a_3}} = (\wedge \quad (if\ (s = 5)$
  $(\vee \quad ((s = 6)\ 0.6)\ ((s = 2)\ 0.4)\ reward\ 100))$
  $(if\ (s = 4)$
  $(\vee \quad ((s = 6)\ 1)\ reward\ 100)))\}$

The goal state is defined as $(s = 6)$. Even though, the state space is relatively small, by exploiting the internal state-action space structure it is empirically shown that the agent gets significantly faster reward accumulation compared to regular MDP formulation giving justifying evidence to verify the theoretical results. Of course, the time performance gap of getting optimal policy would have degraded if the state-action space was much more denser which is evident in the experiments such that since regular MDP has a dense formulation, VI regards this density like lots of 0 probability connections are actual connections, i.e. edges existing between nodes even though probabilities are 0, and iterates over all of those non-existent transitions.

Table 4.1: Running times of solver algorithms with the given formulations for the given number of epochs (in msecs)

	$Small$	$Small_{MDP}$
1000 epochs	0.149 ± 0.0007	1.909 ± 0.003
10000 epochs	1.45 ± 0.02	17.87 ± 0.09

Table 4.1 details the experimental results on problem $Small$ justifying the claims. The practical minimum time to be achieved is $\sim \frac{1}{18}$ of MDP running time due to connectivity ratio of $Small$ to fully connected graph (well, except goal node). Although the state-action space is rather small, exploiting task specific context with conditionals and effects enhanced the running time significantly, achieving a factor of about $12 - 13$ due to branching computational overhead. It can be observed from the formulation and the running times that these branching facilities provide important factorization, and yield the same average accumulated reward in considerably lesser amount of time. Reward accumulation can be observed from Figure 4.2 with discount factor $\gamma = 0.95$. Policy generated for this task is demonstrated in Table 4.2.

Table 4.2: Policy generated for $Small$

states	0	1	2	4	5
$\pi(s)$	a_0	a_2	a_2	a_3	a_3

Hypercube tasks given in Figure 4.3 are the same problem having different number

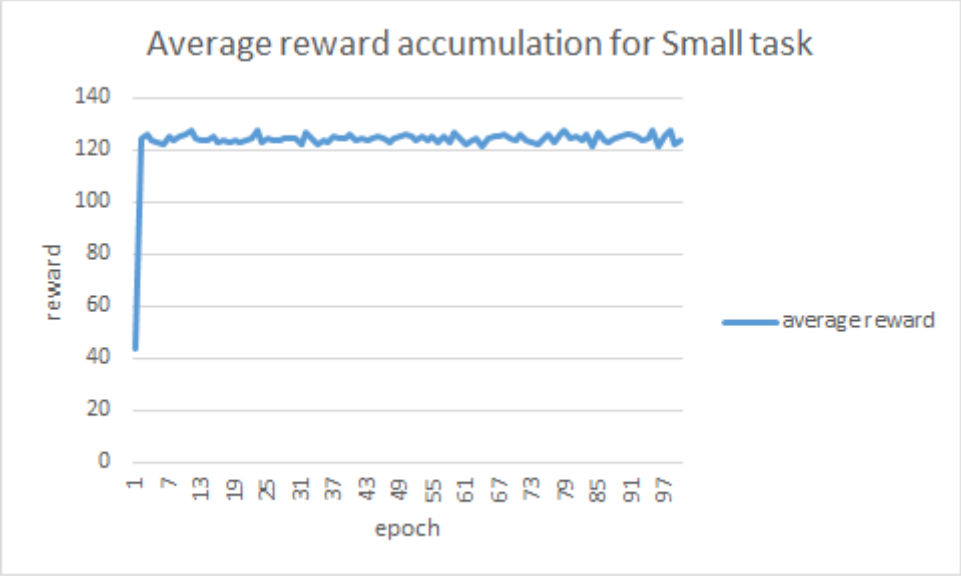


Figure 4.2: Average accumulated rewards for formulation $Small$.

of transitions. These tasks have approximately $2 \times 20 + 1 - 3$ and $2 \times 12 + 1 - 1$ transitions in total, where n is the number of connections in cube and $+1$ is for the $(cypher = 0) \rightarrow (cypher = 1)$ transition from the cypher cell and negative ones are for erasing the transitions from goal state. Dense cube has 20, sparse one has 12 actual state transitions per cube. In those particular versions of hypercube problems, 6 actions are allowed each results in 1 unit of displacement in positive or negative direction in a certain dimension. Diagonal transitions occur with some probability as a byproduct of regular actions due to the problems in the teleportation system. These tasks are formulated as uniconditional binary CA-MDPs which the denser one is encoded with $R(s, a, s')$ formulation as $Hypercube_{3,2,dense} = < \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3\}, \forall i, \Gamma_i = \{0, 1\}$,
- $S = \{x_0, x_1, x_2, cypher\}$,
- $A = \{Left, Right, Down, Up, Backward, Forward, Exit\}$,
- $I = ((x_0 = 0) \wedge (x_1 = 0) \wedge (x_2 = 0))$,
- $C = \{(x_0 = 1) \wedge (x_2 = 0), (x_0 = 0), (x_1 = 1), (x_1 = 0), (x_2 = 1), (x_2 = 0)\}$,
- $E = \{\boldsymbol{E_{Left}} = \{(\wedge \quad ((x_0 = 0)\ 1)$
  $(if\ (x_1 = 1)$

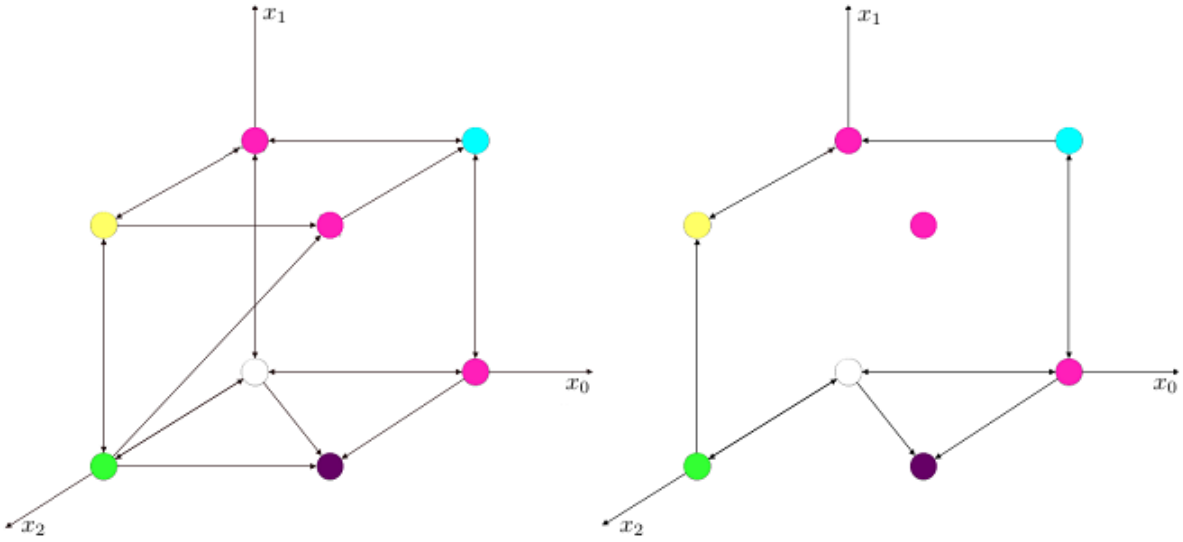


Figure 4.3: State-action transition graph of $Hypercube$. Cell legend; white: initial, pink: hazardous, purple: dead-end, green: safe, blue: cypher, yellow: exit. Each diagonal transition occurs with 0.5 probability, rest is with 1 including $(cypher = 0) \rightarrow (cypher = 1)$ transition (not shown).

$(1\ reward\ -10))),$

$\boldsymbol{E_{Right}} = (\wedge \quad ((x_0 = 1)\ 1)$
$(if\ (((x_1 = 0) \wedge (x_2 = 0)) \vee ((x_1 = 1) \wedge (x_2 = 1)))$
$(1\ reward\ -10))$
$(if\ ((x_1 = 1) \wedge (x_2 = 0) \wedge (cypher = 0))$
$((cypher = 1)\ 1\ reward\ 20))$
$(if\ ((x_1 = 0) \wedge (x_2 = 1))$
$(1\ reward\ -100))),$

$\boldsymbol{E_{Down}} = (\wedge \quad (if\ ((x_0 = 0))$
$((x_1 = 0)\ 1))$
$(if\ ((x_0 = 1) \wedge (x_2 = 0))$
$((x_1 = 0)\ 1\ reward\ -10))),$

$\boldsymbol{E_{Up}} = (\wedge \quad (if\ (\neg((x_0 = 1) \wedge (x_2 = 1)))$
$((x_1 = 1)\ 1)$
$(if\ ((x_0 = 1) \wedge (x_2 = 0) \wedge (cypher = 0))$

$((cypher = 1)\ 1\ reward\ 20)))$
$(if\ ((x_0 = 0) \wedge (x_2 = 1))$
$(if\ (cypher = 1)$
$(\vee \quad ((x_0 = 1)\ 0.5\ reward\ -10)\ (0.5\ reward\ 100)))$
$(if\ (cypher = 0)$
$(\vee \quad ((x_0 = 1)\ 0.5\ reward\ -10)\ 0.5)))$
$(if\ ((x_0 = 0) \wedge (x_2 = 0))$
$(1\ reward\ -10)))),$

$\boldsymbol{E_{Backward}} = (\wedge \quad (if\ ((x_0 = 1) \wedge (x_1 = 1))$
$((x_2 = 0)\ 1)$
$(if\ (cypher = 0)$
$((cypher = 1)\ 1\ reward\ 20)))$
$(if\ (x_0 = 0)$
$((x_2 = 0)\ 1)$
$(if\ (x_1 = 1)$
$(1\ reward\ -10)))),$

$\boldsymbol{E_{Forward}} = (\wedge \quad (if\ ((x_0 = 0) \wedge (x_1 = 1))$
$(if\ (cypher = 1)$
$((x_2 = 1)\ 1\ reward\ 100))$
$(if\ (cypher = 0)$
$((x_2 = 1)\ 1)))$
$(if\ ((x_0 = 0) \wedge (x_1 = 0))$
$(\vee \quad ((x_2 = 1)\ 0.5)\ ((\wedge \quad (x_2 = 1)\ (x_0 = 1))\ 0.5\ reward\ -100)))$
$(if\ ((x_0 = 1) \wedge (x_1 = 0))$
$((x_2 = 1)\ 1\ reward\ -100)))\},$

Hypercube problems, due to their nature, give very important insights about factorization. In $Hypercube_{3,2,dense}$ formulation above, first, in $Left$ action effect declarations you will notice that while generating the language, the ϵ terminal is used in transition node V embedded in T only stating the probability. This (abstract) empty transition is considered as a hypothetical unit deterministic transition which gives us the factorization in the following form: $(\vee \quad ((s = v)\ p)\ ((\wedge \quad (s = v)\ (t = w))\ 1 - p)) =$

$(\wedge \quad ((s = v)\ 1)\ (\vee \quad (p)\ ((t = w)\ 1 - p))$, for arbitrary state variables s, t having arbitrary domains. This type of factorized declaration is used in many of the effects including $Right, Up, Backward$. If $Left$ action is admissible in some state and if $(x_1 = 1)$ condition is satisfied then the whole effect is $((x_0 = 0)\ 1\ reward\ -10)$ otherwise the transition occurs yet no reward is gained, whereas if action Up is taken in state $((x_0 = 0) \wedge (x_1 = 0) \wedge (x_2 = 1) \wedge (cypher = 1))$, then the resulting effect is $(\vee \quad ((\wedge \quad (x_1 = 1)\ (x_0 = 1))\ 0.5\ reward\ -10)\ ((x_1 = 1)\ 0.5\ reward\ 100))$. We deliberately did not use the same form in $Forward$ to reveal the difference, which $(x_2 = 1)$ can be factorized out, since each forward operation always results in 1 unit displacement in that axis. Second, as we just saw in the $Left$ and Up action, we observe that body of an effect is a natural reward factorizer, which otherwise reward declaration can be a quite complicated task when encoding problems with big state-action spaces without any factorization. In order to use the factorized encoding for rewards also, reward being declared in the topmost factor should be simply added, when generating actual transitions with the other factors. Third, in Up action, most general if clause can be easily moved up to conditionals, yet it would have taken the formulation out of uniconditional family and resulting a burden in representations in conditionals due to worse factorization. It is programmer's responsibility to employ such tricks to maintain a balance between data structures and computational efficiency with respect to the constraints. In hypercube problems, we represent states with half of a byte of storage, and used two 8-bit masks, one for getting the desired bits and other one is for checking, per conditional and if clause which proved itself significantly beneficial in terms of both space and time.

Running times for hypercube tasks are detailed in Table 4.3. Again, due to the nature of transitions in the problem, i.e. at most 2 transitions are defined per state-action pair, there is no overhead of tensor products, hence time differences between sparse and dense task can be attributed to number of transitions, the structural representation and branching overhead. Denser problem is about 1.5 times more connectivity than the sparser one, in our experiments, this bound is almost hit with a factor of $1.3 - 1.4$. In addition, we observed that as the problem got denser the performance gap drops, approaching to regular MDP performance justifying our earlier analyses. Average reward accumulation can be seen in Figure 4.4 for the formulation $Hypercube_{3,2,dense}$

with $\gamma = 0.9$. Generated policy is given in Table 4.4. Note that actions in the optimal policy have deterministic effects. For absorbing and non-existent states, no action is assigned by the policy function.

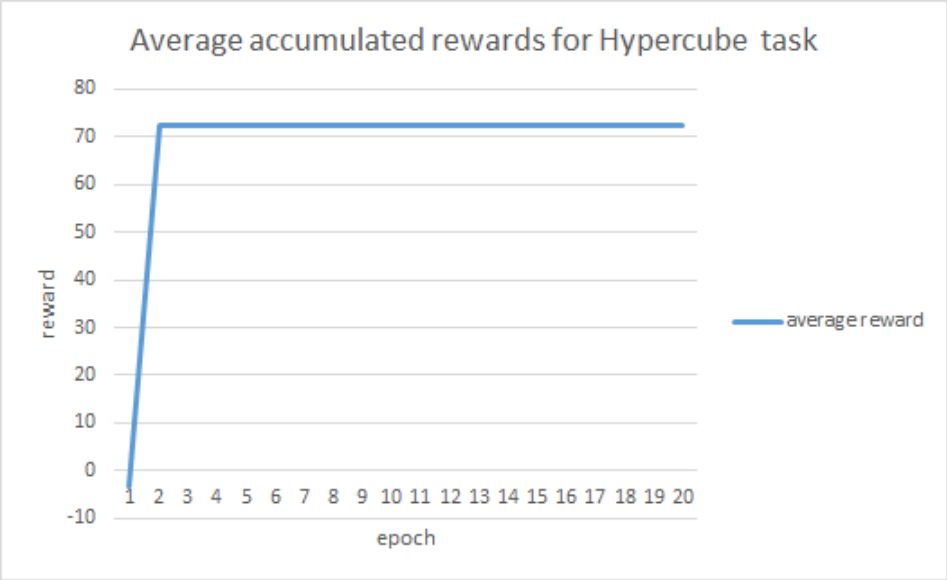


Figure 4.4: Average accumulated rewards for formulation $Hypercube_{3,2,dense}$.

Table 4.3: Running times of solver algorithms with the given formulations for the given number of epochs (in msecs)

	$Hypercube_{3,2,sparse}$	$Hypercube_{3,2,s-MDP}$	$Hypercube_{3,2,dense}$	$Hypercube_{3,2,d-MDP}$
1000 epochs	0.99 ± 0.11	28.27 ± 3.83	1.39 ± 0.01	30.25 ± 0.24
10000 epochs	10.42 ± 0.12	188.96 ± 1.44	13.55 ± 0.12	188.62 ± 3.99

The coffee task, on the other hand, is formulated as both uniconditional and unconditional compound binary CA-MDP. The conditional encoding is given below as $Coffee = < \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5\}, \forall i, \Gamma_i = \{0, 1\}$,
- $S = \{W, U, R, L, HUC, HRC\}$,
- $A = \{Go, BuyCoffee, DeliverCoffee, GetUmbrella\}$,

Table 4.4: Policy generated for $Hypercube_{3,2,dense}$, actions are given with their initials

states	0	1	2	4	6	7	8	9	10	11	12	15
$\pi(s)$	R	U	R	U	R	B	U	L	F	L	B	B

- I = $(((HUC = 0) \wedge (HRC = 0) \wedge (W = 0) \wedge (R = 0) \wedge (U = 0) \wedge (L = 1)) \vee ((HUC = 0) \wedge (HRC = 0) \wedge (W = 0) \wedge (R = 1) \wedge (U = 0) \wedge (L = 1)))$,
- C = $\{\top, ((HRC = 0) \wedge (L = 0)), ((HUC = 0) \wedge (HRC = 1) \wedge (L = 1)), ((U = 0) \wedge (L = 1))\}$,
- $E = \{\boldsymbol{E_{Go}} = (\wedge \quad (if\ ((W = 0) \wedge (R = 1) \wedge (U = 1))$
  $(\vee \quad ((W = 1)\ 0.1)\ \top))$
  $(if\ ((W = 0) \wedge (R = 0) \wedge (U = 0))$
  $(\vee \quad ((W = 1)\ 0.9)\ \top))$
  $(if\ (L = 0)$
  $(\vee \quad ((L = 1)\ 0.9)\ \top))$
  $(if\ (L = 1)$
  $(\vee \quad ((L = 0)\ 0.9)\ \top))),$

  $\boldsymbol{E_{BuyCoffee}} = (\wedge \quad \boldsymbol{(if\ ((HRC = 0) \wedge (L = 0))}$
  $(\vee \quad ((HRC = 1)\ 0.9)\ \top))),$

  $\boldsymbol{E_{DeliverCoffee}} = (\wedge \quad \boldsymbol{(if\ ((HUC = 0) \wedge (HRC = 1) \wedge (L = 1))}$
  $(\vee \quad ((\wedge \quad (HUC = 1)\ (HRC = 0))\ 0.8\ reward\ 0.9)\ \top))$
  $(if\ (W = 0)$
  $(1\ reward\ 0.1))),$

  $\boldsymbol{E_{GetUmbrella}} = (\wedge \quad \boldsymbol{(if\ ((U = 0) \wedge (L = 1))}$
  $(\vee \quad ((U = 1)\ 0.9)\ \top)))\}$

The state variables are defined in section 2.2.1 and the robot accomplishes its task when the user has coffee, i.e. $(HUC = 1)$, getting a reward of 0.9 and an additional 0.1 if the robot is not wet, $(W = 0)$. The unconditional version of coffee problem, $Coffee_u$, has the same formulation except no preconditions are defined whereas con-

ditional version can be obtained by removing the if clauses given in bold font. Also, note that, unlike in the problem $Small$, this formulation has $R(s, a, s')$ formulation, hence the given extended formulation, defined in equation **??**, is used for the task encoding.

Unlike in the previous problems, in coffee task, we used tensor products extensively. To give an example, for the action Go, we have the following product from an state satisfying the first and third if condition:

$$\begin{bmatrix} W=0 \\ W=1 \end{bmatrix} \otimes \begin{bmatrix} L=0 \\ L=1 \end{bmatrix} = \begin{bmatrix} (W=0) \wedge (L=0) \\ (W=0) \wedge (L=1) \\ (W=1) \wedge (L=0) \\ (W=1) \wedge (L=1) \end{bmatrix} \tag{4.8}$$

The probabilities of resulting transitions are calculated in a similar manner. The results of trials are detailed in Table 4.5. Note that, the results are summation of individual running times of problem with different initial states. In our trials, we used online generation of transitions, rather than generating once and storing them in memory. Even though, there is huge amount of product overhead, still, our proposed algorithm performed significantly better than regular VI. This is of course due to the nature of the problem such that actions do not have any effect for most the states which in turn gives factorization power to the programmer. The gap clearly would have been wider, and the execution times can be closer to practical minimum run time, if we performed off-line generation of states which can be preferable for tasks with relatively small state-action connectivity space. The results are summation of separate running times of problem with different initial states with the time cost of offline generation of transition matrix for MDP from state variables. Moreover, the gains from declaring conditionals can be understood clearly by comparing two different CA-MDP formulations of the coffee problem. Accumulated rewards over iterations can be found in Figure 4.5 with discount factor $\gamma = 0.95$.

The taxi task, given in Figure 4.6, containing 5×5 grid for this case, is formulated as

Table 4.5: Running times of solver algorithms with the given formulations for the given number of epochs (in msecs)

	$Coffee$	$Coffee_u$	$Coffee_{MDP}$
1000 epochs	7.86 ± 0.01	9.02 ± 0.09	104.02 ± 0.96
10000 epochs	66.57 ± 0.48	74.45 ± 0.41	553.7 ± 21.4

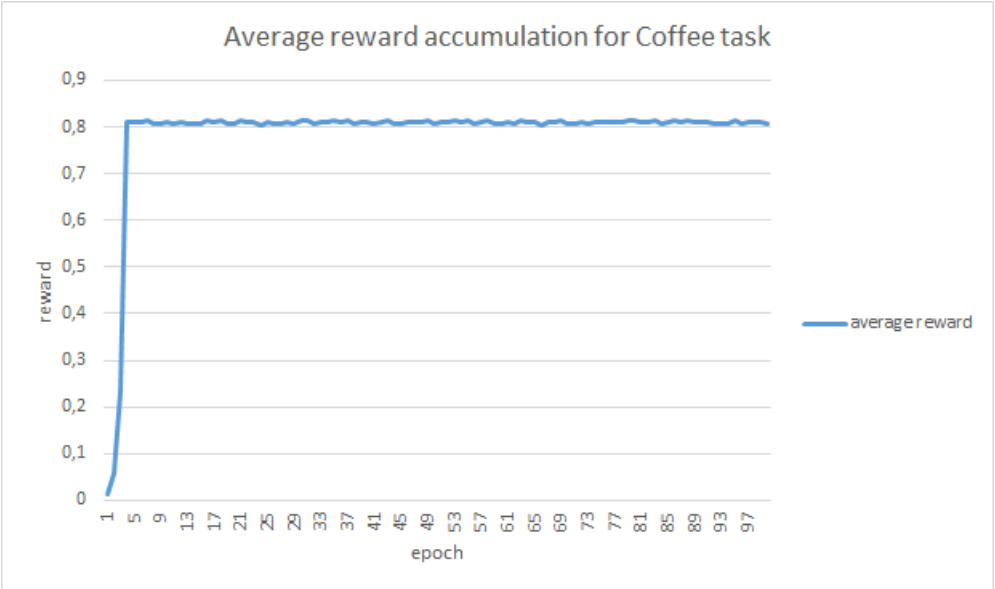


Figure 4.5: Average accumulated rewards for formulation $Coffee$.

regular CA-MDP having $R(s, a, s')$ formulation as $Taxi_{(5,5)} = < \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0, \Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4, \Gamma_5\}$,
  $\Gamma_0 = \Gamma_1 = \{0, 1, 2, 3, 4\}$,
  $\Gamma_2 = \{R, G, B, Y, Taxi\}$
  $\Gamma_3 = \{R, G, B, Y\}$
  $\Gamma_4 = \Gamma_5 = \{false, true\}$
- $S = \{X, Y, P, D, I, U\}$,
- $A = \{North, South, East, West, Pickup, Putdown\}$,
- $I = ((X = 3) \wedge (Y = 1) \wedge (P = G) \wedge (D = G) \wedge (I = false) \wedge (U = false))$,
- $C = \{\neg(Y = 4) \vee (U = true) \vee (I = true), \neg(Y = 0) \vee (U = true) \vee (I = true), \neg(X = 4) \vee (U = true) \vee (I = true), \neg(X = 0) \vee (U = true) \vee (I = true)\}$,

- $E = \{\boldsymbol{E_{North}} = (\wedge \quad (if\ (Y = 0)$
  $((Y = 1)\ 1))$
  $(if\ (Y = 1)$
  $((Y = 2)\ 1))$
  $(if\ (Y = 2)$
  $((Y = 3)\ 1))$
  $(if\ (Y = 3)$
  $((Y = 4)\ 1))$
  $((\wedge \quad (U = false)\ (I = false))\ 1)),$

  $\boldsymbol{E_{South}} = (\wedge \quad (if\ (Y = 1)$
  $((Y = 0)\ 1))$
  $(if\ (Y = 2)$
  $((Y = 1)\ 1))$
  $(if\ (Y = 3)$
  $((Y = 2)\ 1))$
  $(if\ (Y = 4)$
  $((Y = 3)\ 1))$
  $((\wedge\ (U = false)\ (I = false))\ 1)),$

  $\boldsymbol{E_{East}} = (\wedge \quad (if\ (X = 0)$
  $(if\ ((Y = 2) \vee (Y = 3) \vee (Y = 4))$
  $((X = 1)\ 1)))$
  $(if\ (X = 1)$
  $(if\ ((Y = 0) \vee (Y = 1) \vee (Y = 2))$
  $((X = 2)\ 1)))$
  $(if\ (X = 2)$
  $(if\ ((Y = 2) \vee (Y = 3) \vee (Y = 4))$
  $((X = 3)\ 1)))$
  $(if\ (X = 3)$
  $((X = 4)\ 1))$
  $((\wedge\ (U = false)\ (I = false))\ 1)),$

  $\boldsymbol{E_{West}} = (\wedge \quad (if\ (X = 1)$
  $(if\ ((Y = 2) \vee (Y = 3) \vee (Y = 4))$
  $((X = 0)\ 1)))$
  $(if\ (X = 2)$

$(if\ ((Y = 0) \vee (Y = 1) \vee (Y = 2))$
$((X = 1)\ 1)))$
$(if\ (X = 3)$
$(if\ ((Y = 2) \vee (Y = 3) \vee (Y = 4))$
$((X = 2)\ 1)))$
$(if\ (X = 4)$
$((X = 3)\ 1))$
$((\wedge\ (U = false)\ (I = false))\ 1)),$

$\boldsymbol{E_{Pickup}} = (\wedge \quad (if\ ((X = 4) \wedge (Y = 4))$
$((\wedge\ (P = Taxi)\ (D = R)\ (I = false))\ 1))$
$(if\ (\neg((X = 4) \wedge (Y = 4)))$
$((\wedge\ (P = G)\ (D = G)\ (I = true))\ 1\ reward\ -10))$
$((U = false)\ 1)),$

$\boldsymbol{E_{Putdown}} = (\wedge \quad (if\ ((P = Taxi) \wedge (X = 0) \wedge (Y = 4))$
$((\wedge\ (P = R)\ (U = false))\ 1\ reward\ 20))$
$(if\ (\neg((P = Taxi) \wedge (X = 0) \wedge (Y = 4)))$
$((U = true)\ 1\ reward\ -10))$
$((I = false)\ 1))\}$

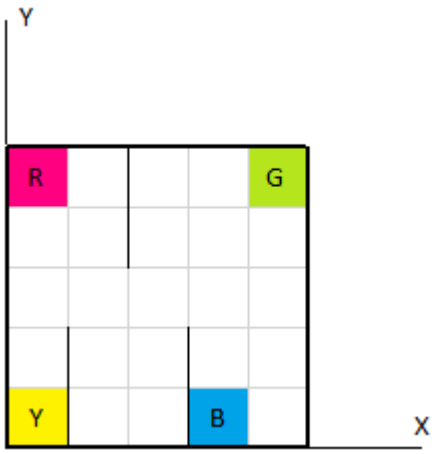


Figure 4.6: Taxi problem

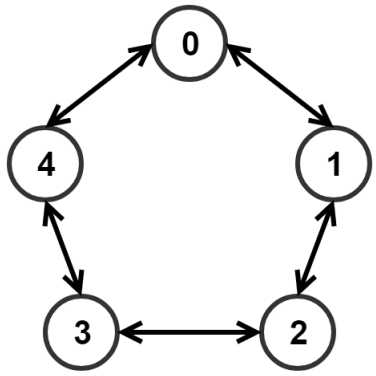
The state variables are the location of taxi in the grid, location of passenger, destination point and indicators of wrong pick up and put down. The goal is defined as

passenger is put down in location R, i.e. $(P = R)$. Like coffee and many other real world tasks, taxi problem is a sparse problem, however it has huge state-action space. That's the main reason why results in Table 4.6 demonstrate way more bigger gap than the other problems. Other important reason is that the transitions are all deterministic, hence no tensor product overhead arose again.

Table 4.6: Running times of solver algorithms with the given formulations for the given number of epochs (in msecs)

	$Taxi_{(5,5)}$	$Taxi_{(5,5,MDP)}$
100 epochs	29.74 ± 3.36	$> 15K$

Finally, different cases of the SysAdmin tasks are implemented having all bidirectional ring topology with varying number of connected machines. In bidirectional topology, each machine i is connected to two machines having ids $i \pm 1$. Therefore, the failure probability of a given machine is effected significantly from the states of the previous and successor machines in the network. 5 machine case with corresponding conditional probabilities occurring from the bidirectional ring connectivity, $P(\boldsymbol{X}_i^{t+1} \mid \boldsymbol{X}_i^t, \boldsymbol{X}_{i-1}^t, \boldsymbol{X}_{i+1}^t, \boldsymbol{A})$, is detailed in Figure 4.7, where $\boldsymbol{X}_i^t$ represents the state of machine i in time step t.



X_i^t	X_{i-1}^t	X_{i+1}^t	$P(X_i^{t+1} = 1 \mid X_i^t, X_{i-1}^t, X_{i+1}^t, A = noop)$
0	x	x	0
1	0	0	0.3
1	0	1	0.6
1	1	0	0.6
1	1	1	0.9

Figure 4.7: Bidirectional topology (left), conditional probabilities when the action is *noop* (right). When a machine is in failure condition it remains in the same condition.

When $reboot_i$ action is taken, machine i will be in running state in the next time step with 100% probability regardless of the states of neighbouring machines. SysAdmin will get a reward of 2 for each running machine and 0 for the failed ones. Here is an example formulation for 3 machine SysAdmin task as unconditional binary CA-MDP, $SysAdmin_3 = < \Gamma, S, A, C, E, I >$:

- $\Gamma = \{\Gamma_0, \Gamma_1, \Gamma_2\}$,
  $\Gamma_0 = \Gamma_1 = \Gamma_2 = \{0, 1\}$,
- $S = \{X_0, X_1, X_2\}$,
- $A = \{reboot_0, reboot_1, reboot_2, noop\}$,
- $I = ((X_0 = 0) \wedge (X_1 = 0) \wedge (X_2 = 0))$,
- $C = \{\top, \top, \top\}$,
- $$\begin{aligned} E = \{\boldsymbol{E_{reboot_0}} = (\wedge \quad & ((X_0 = 1)\ 1) \\ & (if\ ((X_2 = 1)\ \vee\ (X_1 = 1)) \\ & (if\ ((X_2 = 0)\ \wedge (X_0 = 0)) \\ & (\vee \quad ((X_1 = 1)\ 0.3)\ \top)) \\ & (if\ (((X_2 = 0)\ \wedge (X_0 = 1))\ \vee\ ((X_2 = 1)\ \wedge (X_0 = 0))) \\ & (\vee \quad ((X_1 = 1)\ 0.6)\ \top)) \\ & (if\ ((X_2 = 1)\ \wedge (X_0 = 1)) \\ & (\vee \quad ((X_1 = 1)\ 0.9)\ \top)) \\ & (if\ ((X_1 = 0)\ \wedge (X_0 = 0)) \\ & (\vee \quad ((X_2 = 1)\ 0.3)\ \top)) \\ & (if\ (((X_1 = 0)\ \wedge (X_0 = 1))\ \vee\ ((X_1 = 1)\ \wedge (X_0 = 0))) \\ & (\vee \quad ((X_2 = 1)\ 0.6)\ \top)) \\ & (if\ ((X_1 = 1)\ \wedge (X_0 = 1)) \\ & (\vee \quad ((X_2 = 1)\ 0.9)\ \top)))), \\ & ... \end{aligned}$$

For SysAdmin task, it can be observed that state space grows exponentially with the number of machines. In addition, running machines in a network can be in any of possible states in the state space in the next time step. Hence, for a given state, the size of its connectivity is defined from the number of running machines in that state which has again exponential growth rate. In order to address these difficulties and

analyse the run times and memory requirements we carried out two separate sets of experiments.

In the first part of our experiments, we compared results of our model and solver algorithm with regular MDP setting constructed from the bidirectional ring topology with various number of machines. Comparison chart for running times of problems having machines from 5 to 12 is given in Figures 4.8 and 4.9.

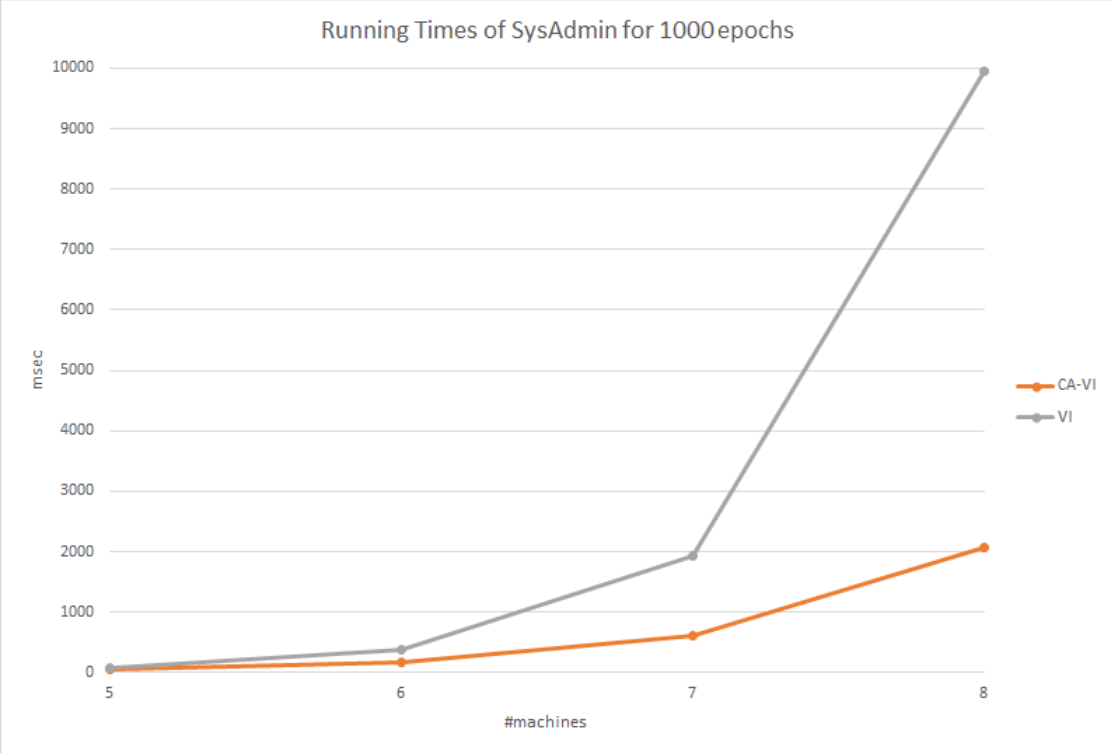


Figure 4.8: Average running times for SysAdmin problems with various number of machines. All have bidirectional ring topology.

As expected, our algorithm performed better and scaled accordingly with respect to the state and connectivity space of the problem. Even though, when the number of machines in the network increases using regular MDP encoding becomes infeasible, online products of tensors also start to cause significant amount of overhead (although these products can be computed in a parallel fashion, however this is beyond our scope). This huge processing overhead is attributed to the very large size of state space and connectivity. Hence, putting all of the transitions in memory or generating all of the transitions from the state variables in an online fashion are both

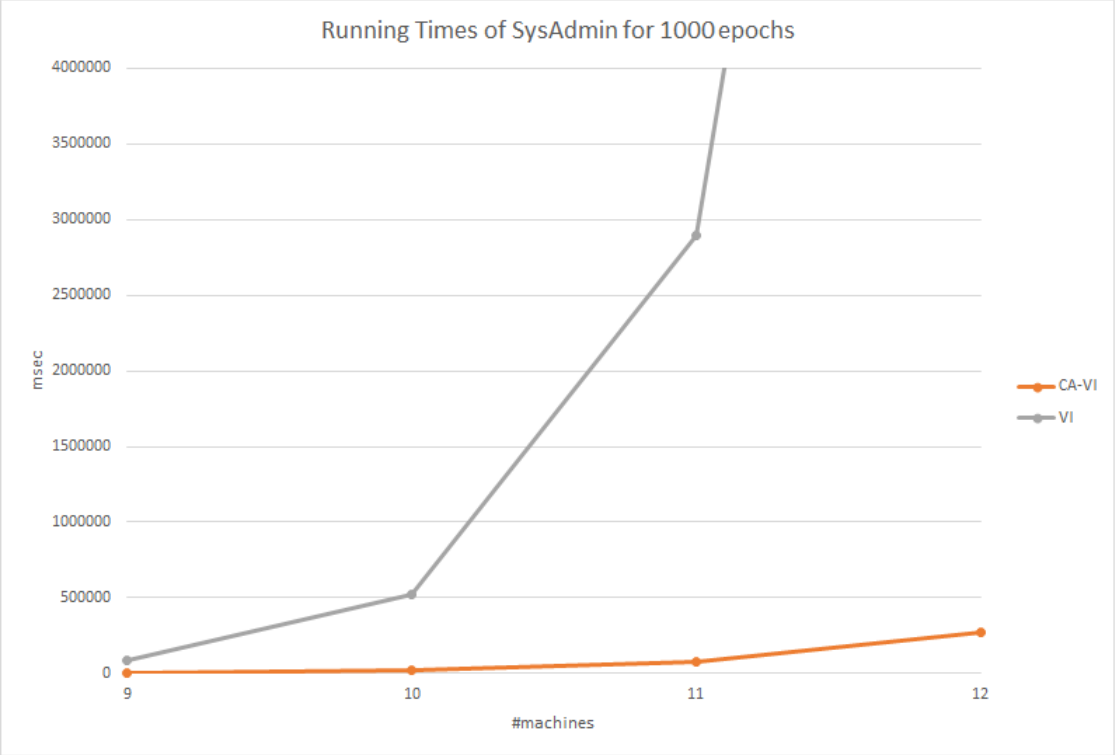


Figure 4.9: Average running times for SysAdmin problems with various number of machines. All have bidirectional ring topology.

infeasible in this computational setting. A middle ground can be established so that both complexity of memory and tensor processing time is alleviated.

In the second part of our experiments, to overcome this problem and further decrease the amount of time for generating an optimal policy, we factorized the problem into two separate set of state variables. For the first partition, all of the tensor products are processed offline and stored in memory. Transition probabilities are generated during execution of CA-MDP VI algorithm for the second partition. This, not only significantly decreased the running time, but also allowed CA-MDP VI to use the memory space available. Table 4.7 summarizes our experiments for tasks with very large state spaces.

Non-factorized formulations generate all of the transition probabilities ground up online. Factorized formulations divide state variables into two equal size groups and transitions are generated offline for one selected group. During execution of solver

Table 4.7: Comparison of average running times of CA-MDP VI algorithm with factorized and non-factorized formulations (in secs)

#machines	Factorized	Non-Factorized
14	5133	5305
16	56423	57896
18	616462	664272

algorithm, transition probabilities of state variables of the other group are generated. Then, tensor processing is carried out between these distributions with the precomputed ones in order to get the required actual transition probabilities to compute value function. As can be seen from the table, time gap between running times is increasing as the state space grows large. The gap can be wider with careful selection of number of factorized variables. However, finding that optimal number is beyond the scope of our study.

In summary, we tried to show the powerful aspects of CA-MDP VI with appropriate problems. Most of the tasks emit real-world problem behaviour i.e. localized admissibility and effects of actions, which enable us to factorize to some extent and solver to converge to practical minimum run time bound. SysAdmin and Coffee problem have higher factorization rate than the others which comes with tensor product cost. To overcome the difficulties of the tensor product overheads, an optimal point balancing between memory usage and tensor processing can be established to optimize run time. Nevertheless, in all of the problems, the internal structure is exploited hence enabling us to perfectly compress the representation of the problem as much as possible and almost hit the theoretical bounds. Not only from theory, but also from the experiments we can safely state that we successfully integrated problem specific information and declared effects and conditionals appropriately. These declarations in turn enabled CA-MDP VI to exploit the dependencies and prune some unnecessary and/or non-existent transitions, leading to faster convergence rate.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion and Future Work

Several models and corresponding solution algorithms are proposed for autonomous agents to represent a task in a partially observable setting which has two main issues: space complexity of states and uncertainty of the agent. Even though, there exists proposed solutions for the state space problem of POMDP formulations, this work may be regarded as an attempt to give a solution to get rid of the burden of the problems mentioned by integrating the structural information available a-priori using compact representation style of planning domain.

CA-POMDP uses state variables rather than states for the representation of a given problem. Actual states are defined as the conjunction of state variables, with the exception of singular CA-POMDP family. Combined with the grammar representing the relationship in a structured manner rather than explicitly enumerating transition matrix of the underlying environment, space complexity is considerably reduced. Exploiting relationships between states, actions and observations is essential for efficiently generating policies in AI. Also, we incorporated conditionals in the formulation which also helps with the factorization of the given problem. This structured representation shows that CA-POMDP is favorable for integrating the context dependent information in a given problem, hence easing up the computational burden of learning optimal policies. This factorization may be enough by itself in fully observable environments to improve complexity, however it is not in partially observable environments due to the need of tracking of states.

Uncertainty and efficiency are always in contrast each other. Our model, CA-POMDP

improves efficiency by representing states and the uncertainty of agent's in those states in a compact structure. Our proposed model CA-POMDP incorporates a grammar-based structure to express those relationships. In order to fully use the advantage of this representation, belief state updates need to be represented in the same manner which otherwise explicit enumeration of real states are needed to store the belief distribution and exponential number of updates need to be executed for distribution update. This representation is very crucial on generating optimal policy since the time complexity is alleviated by getting rid of unnecessary computations significantly.

On the other hand, the enhanced algorithm for solving problems in fully observable environments, CA-MDP VI, takes advantage of the locality, i.e. effects, to update the value function, $V(s)$, in a more efficient way. As shown in the previous chapter, CA-MDP VI eliminates the redundant computations using conditionals. The other portion of gains are coming from the structured effects of actions. Even though CA-MDP VI performs fewer computations than regular VI and guarantees an optimal policy, as VI does. In the experiments, we not only presented empirical evidence for the theoretical basis for analyses of algorithm by comparing our results to the practical lower bounds but also discussed the way formulating problems with various formulations.

As a conclusion, combining the strongest features of both worlds, namely compact representation and dynamic programming techniques, Markov Processes can be improved significantly for AI domain. CA-POMDP can compactly represent problems in partially observable settings using grammar based form which is also programmer friendly. The most important gain obtained by this formulation roots from defining structural dependencies of problems which enable exploitation of further sub-structures and localized dependencies. Moreover, our solution algorithm CA-VI performed well as suggested by the theoretical underpinnings. We presented our experimental setting in detail and justified theoretical analyses empirically. For future work, as mentioned in introduction, a new solution algorithm that can exploit CA-POMDP facilities can be developed. This algorithm should use the compact representation of states-actions-observations and factored belief updates. Moreover, one can define further top-down factorizations over CA-POMDP formulation, like generating sub-CA-POMDPs and factorizing conditionals over different state variables leading to even better structured representations hence in turn solution algorithms can be engineered

accordingly to take these improvements into account to get the approximate/exact optimal policy.

REFERENCES

[1] R. Bellman, "A Markovian Decision Process," *Journal of Mathematical Mechanics*, vol. 6, pp. 679–684, 1957.

[2] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting Structure in Policy Construction," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1104–1113, 1995.

[3] A. Jonsson and A. Barto, "Causal Graph Based Decomposition of Factored MDPs," *Journal of Machine Learning Research*, vol. 7, pp. 2259–2301, Dec. 2006.

[4] A. Raghavan, S. Joshi, A. Fern, P. Tadepalli, and R. Khardon, "Planning in Factored Action Spaces with Symbolic Dynamic Programming," in *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI'12, pp. 1802–1808, 2012.

[5] A. Raghavan, R. Khardon, A. Fern, and P. Tadepalli, "Symbolic Opportunistic Policy Iteration for factored-action MDPs," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, vol. 5 of *NIPS'13*, pp. 2499–2507, 2013.

[6] A. Raghavan, R. Khardon, P. Tadepalli, and A. Fern, "Memory-efficient Symbolic Online Planning for Factored MDPs," in *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, UAI'15, pp. 732–741, 2015.

[7] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic Abstraction in Reinforcement Learning via Clustering," in *Proceedings of the 21st International Conference on Machine Learning*, pp. 560–567, 2004.

[8] A. McGovern and A. G. Barto, "Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density," in *Proceedings of the 18th International Conference on Machine Learning*, pp. 361–368, 2001.

[9] T. R. Colin, T. Belpaeme, A. Cangelosi, and N. Hemion, "Hierarchical Reinforcement Learning as Creative Problem Solving," *Robotics and Autonomous Systems*, vol. 86, pp. 196 – 206, 2016.

[10] A. Bai, F. Wu, and X. Chen, "Online Planning for Large Markov Decision Processes with Hierarchical Decomposition," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, pp. 45:1–45:28, July 2015.

[11] S. Girgin, F. Polat, and R. Alhajj, "Improving Reinforcement Learning by Using Sequence Trees," *Machine Learning*, vol. 81, pp. 283–331, Dec 2010.

[12] T. Dean and R. Givan, "Model Minimization in Markov Decision Processes," in *Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 106–111, 1997.

[13] T. G. Dietterich, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, Nov. 2000.

[14] L. M., *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Providence, Rhode Island, 1996.

[15] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, no. 1-2, pp. 99–134, 1998.

[16] A. R. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for partially observable markov decision processes," in *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997* (D. Geiger and P. P. Shenoy, eds.), pp. 54–61, Morgan Kaufmann, 1997.

[17] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2* (W. J. Clancey and D. S. Weld, eds.), pp. 1168–1175, AAAI Press / The MIT Press, 1996.

[18] P. Poupart, K. Kim, and D. Kim, "Closing the gap: Improved bounds on optimal POMDP solutions," in *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011* (F. Bacchus, C. Domshlak, S. Edelkamp, and M. Helmert, eds.), AAAI, 2011.

[19] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *Auton. Agents Multi Agent Syst.*, vol. 27, no. 1, pp. 1–51, 2013.

[20] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[21] E. J. Sondik, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.

[22] O. Madani, S. Hanks, and A. Condon, "On the undecidability of probabilistic planning and related stochastic optimization problems," *Artif. Intell.*, vol. 147, no. 1-2, pp. 5–34, 2003.

[23] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "PDDL - the Planning Domain Definition Language," tech. rep., CMU-CS, 08 1998.

[24] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61 – 124, 2003.

[25] H. L. S. Younes and M. L. Littman, "PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects," tech. rep., CMU-CS, 2004.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[27] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming.* Belmont, MA: Athena Scientific, 1996.

[28] E. J. Sondik, "The optimal control of partially observable markov processes," Tech. Rep. AD0730503, Stanford Univ Calif Stanford Electronics Labs, 05 1971.

[29] G. E. Monahan, "State of the art—a survey of partially observable markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[30] J. N. Eagle, "The optimal search for a moving target when the search path is constrained," *Oper. Res.*, vol. 32, no. 5, pp. 1107–1115, 1984.

[31] T. Smith and R. G. Simmons, "Heuristic search value iteration for pomdps," in *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004* (D. M. Chickering and J. Y. Halpern, eds.), pp. 520–527, AUAI Press, 2004.

[32] D. A. McAllester and S. P. Singh, "Approximate planning for factored pomdps using belief state simplification," in *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999* (K. B. Laskey and H. Prade, eds.), pp. 409–416, Morgan Kaufmann, 1999.

[33] X. Boyen and D. Koller, "Tractable inference for complex stochastic processes," in *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998* (G. F. Cooper and S. Moral, eds.), pp. 33–42, Morgan Kaufmann, 1998.

[34] T. Dean and K. Kanazawa, "A Model for Reasoning About Persistence and Causation," *Computational Intelligence*, vol. 5, pp. 142–150, Dec. 1989.

[35] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2* (W. J. Clancey and D. S. Weld, eds.), pp. 1168–1175, AAAI Press / The MIT Press, 1996.

[36] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Formal Methods Syst. Des.*, vol. 10, no. 2/3, pp. 171–206, 1997.

[37] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier, "SPUDD: stochastic planning using decision diagrams," in *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999* (K. B. Laskey and H. Prade, eds.), pp. 279–288, Morgan Kaufmann, 1999.

[38] C. Guestrin, D. Koller, and R. Parr, "Solving factored pomdps with linear value functions," in *In IJCAI-01 workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[39] E. A. Hansen and Z. Feng, "Dynamic programming for pomdps using a factored state representation," in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000* (S. A. Chien, S. Kambhampati, and C. A. Knoblock, eds.), pp. 130–139, AAAI, 2000.

[40] H. S. Sim, K. Kim, J. H. Kim, D. Chang, and M. Koo, "Symbolic heuristic search value iteration for factored pomdps," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008* (D. Fox and C. P. Gomes, eds.), pp. 1088–1093, AAAI Press, 2008.

[41] J. Pineau, G. J. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003* (G. Gottlob and T. Walsh, eds.), pp. 1025–1032, Morgan Kaufmann, 2003.

[42] T. Smith and R. G. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pp. 542–547, AUAI Press, 2005.

[43] G. Shani, R. I. Brafman, and S. E. Shimony, "Prioritizing point-based POMDP

solvers," *IEEE Trans. Syst. Man Cybern. Part B*, vol. 38, no. 6, pp. 1592–1605, 2008.

[44] T. Veiga, M. T. J. Spaan, and P. U. Lima, "Point-based POMDP solving with factored value function approximation," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada* (C. E. Brodley and P. Stone, eds.), pp. 2513–2519, AAAI Press, 2014.

[45] M. T. J. Spaan and N. A. Vlassis, "Perseus: Randomized point-based value iteration for pomdps," *J. Artif. Intell. Res.*, vol. 24, pp. 195–220, 2005.

[46] B. L., B. R., D. S. B., and E. D., *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2010.

[47] R. Parr and S. Russell, "Reinforcement Learning with Hierarchies of Machines," in *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pp. 1043–1049, 1998.

[48] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181 – 211, 1999.

[49] S. Thrun and A. Schwartz, "Finding Structure in Reinforcement Learning," in *Advances in Neural Information Processing Systems 7*, pp. 385–392, 1995.

[50] M. Kamal and J. Murata, "Reinforcement Learning for Problems with Symmetrical Restricted States," *Robotics and Autonomous Systems*, vol. 56, pp. 717–727, 9 2008.

[51] B. Ravindran and A. G. Barto, "Smdp Homomorphisms: An Algebraic Approach to Abstraction in Semi-Markov Decision Processes," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pp. 1011–1016, 2003.

[52] M. Zinkevich and T. Balch, "Symmetry in Markov Decision Processes and its Implications for Single Agent and Multi Agent Learning," in *Proceedings of the 18th International Conference on Machine Learning*, pp. 632–640, 2001.

[53] B. Hengst, “Discovering Hierarchy in Reinforcement Learning with HEXQ,” in *Machine Learning: Proceedings of the 19th International Conference on Machine Learning*, pp. 243–250, 2002.

[54] N. Taghizadeh and H. Beigy, “A Novel Graphical Approach to Automatic Abstraction in Reinforcement Learning,” *Robotics and Autonomous Systems*, vol. 61, no. 8, pp. 821 – 835, 2013.

[55] G. Kheradmandian and M. Rahmati, “Automatic Abstraction in Reinforcement Learning Using Data Mining Techniques,” *Robotics and Autonomous Systems*, vol. 57, no. 11, pp. 1119 – 1128, 2009.

[56] O. Ekmekci, “Context-aware markov decision processes,” Master’s thesis, Middle East Technical University, Ankara, TURKEY, 2014.

[57] M. J. Kochenderfer and H. J. D. Reynolds, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.