PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

SOLVING A MARKOV DECISION PROCESS MULTIDIMENSIONAL PROBLEM WITH TENSOR DECOMPOSITION

DANIELA KUINCHTNER

Master Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Afonso Henrique Correa de Sales
Co-Advisor: Prof. Dr. Felipe Rech Meneguzzi

Porto Alegre
2021

ACKNOWLEDGMENTS

Primeiramente, gostaria de agradecer ao meu orientador, Afonso Henrique Correa de Sales, e ao meu coorientador, Felipe Rech Meneguzzi, por todo o conhecimento fundamental desta pesquisa, tanto para a minha formação acadêmica, quanto para a minha formação pessoal.

Agradeço à CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) pela oportunidade de uma bolsa de estudos para a realização do Mestrado.

Agradeço aos meus pais, Nadira e Élio, por todo o apoio emocional e financeiro que me deram, vocês significam tudo para mim. Agradeço ao meu namorado e amigo, Felipe, pela compreensão das vezes que faltei com você. Saibam que os amo e os admiro muito.

E agradeço aos meus amigos, Anielle Lisboa, Laura Tomaz, Karina Kohl, Gabriel Paludo e Cássio Trindade, por todo incentivo e ajuda que foram de grande importância nesses dois anos.

SOLUCIONANDO UM PROBLEMA MULTIDIMENSIONAL DE PROCESSO DE DECISÃO DE MARKOV UTILIZANDO DECOMPOSIÇÃO DE TENSORES

RESUMO

Processo de Decisão de Markov (MDP) é um modelo usado para planejamento de tomada de decisão de agentes em ambientes estocásticos e completamente observáveis. Embora, muita pesquisa se concentra na solução de problemas de MDPs atômicos em formas tabulares ou MDPs com representações fatoradas, nenhuma se baseia em métodos de decomposição de tensores. Resolver MDPs usando álgebra tensorial oferece a perspectiva de alavancar avanços em cálculos baseados em tensor para aumentar a eficiência de solucionadores de MDP. Nesta pesquisa, primeiro, é formalizado problemas multidimensionais de MDP usando álgebra tensorial. Segundo, é desenvolvido um solucionador de MDP usando o método de decomposição de tensor CANDECOMP-PARAFAC para compactar as matrizes de transição de estados. O solucionador utiliza os algoritmos de *iteração de valor* e *iteração de política* para computar a solução de forma compacta. Então, os algoritmos compactos são avaliados de forma empírica em comparação com métodos tabulares. Como resultados, é mostrado que a abordagem tensorial pode computar problemas maiores usando substancialmente menos memória, abrindo novas possibilidades para métodos baseadas em tensores para o planejamento estocástico.

Palavras-Chave: inteligência artificial, decomposição CANDECOMP-PARAFAC, processo de decisão de markov, álgebra tensorial, decomposição de tensores.

SOLVING A MARKOV DECISION PROCESS MULTIDIMENSIONAL PROBLEM WITH TENSOR DECOMPOSITION

ABSTRACT

Markov Decision Process (MDP) is a model used for planning decision-making of agents in stochastic and completely observable environments. Although much research is focused on solving atomic MDP problems in tabular forms or MDP problems with factored representations, none is based on tensor decomposition methods. Solving MDPs using tensor algebra offers the prospect of leveraging advances in tensor-based calculations to increase MDP solvers' efficiency. In this research, first, we formalize MDP multidimensional problems using tensor algebra. Second, we develop an MDP solver using the CANDECOMP-PARAFAC tensor decomposition method to compact state transition matrices. The solver uses the value iteration and policy iteration algorithms to compute the solution compactly. Then, we empirically evaluate the compact algorithms compared to tabular methods. As a result, we show that the tensor approach can compute larger problems using substantially less memory, opening up new possibilities for tensor-based methods for stochastic planning.

Keywords: artificial intelligence, CANDECOMP-PARAFAC decomposition, Markov decision process, tensor algebra, tensor decomposition.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

MDP – Markov Decision Process

DBN – Dynamic Bayesian Network

CP – CANDECOMP-PARAFAC, or Canonical Polyadic Decomposition with Parallel Factors

CP-MDP – CANDECOMP-PARAFAC Markov Decision Process

AI – Artificial Intelligence

SVD – Singular Value Decomposition

PCA – Principal Component Analysis

CP-MDP-VI – compact value iteration using CANDECOMP-PARAFAC

CP-MDP-PI – compact policy iteration using CANDECOMP-PARAFAC

TABULAR-VI – tabular value iteration algorithm

TABULAR-PI – tabular policy iteration algorithm

MPI – Modified Policy Iteration

SPI – Structured Policy Iteration

KKT – Karush-Kuhn-Tucker

LIST OF SYMBOLS

CONTENTS

1. INTRODUCTION

Decision theory is concerned with the reasoning underlying an agent's choice, and it provides a framework for an agent's decisions made under uncertainty. These decisions occur when payoffs from actions are not immediate but instead result from several actions taken in sequence (Russell and Norvig, 2009, Ch. 1, p. 10). The work of Richard Bellman (Bellman, 1957b) formalizes a class of sequential decision problems called *Markov decision process* (MDP), which is an elegant mathematical formalism to model stochastic domains. Sequential planning applications, where uncertainty is crucial to account in the process, such as autonomous robots (Cassandra et al., 1996; Boger et al., 2006; Jónsson et al., 2000; Lamini et al., 2018), machine maintenance (Amari et al., 2006; Liu et al., 2018; Durazo-Cardenas et al., 2018), medical diagnosis (Schaefer et al., 2004; Iantovics, 2008; Berhili et al., 2020), satellite image analysis (Toomey and Mark, 1995; Sahe et al., 2004), and refinery controller systems (Joly et al., 2002; Tominac and Mahalec, 2017) are real-world problem examples. Such complexities are better captured by stochastic formalisms, like MDPs, because nontrivial problems have multiple dimensions/features and involve many state variables.

The solution to an MDP problem, given a stochastic state transition system, is an optimal policy that defines every state's optimal action in the domain. Most approaches that solve such problems using tabular representations, as Value Iteration and Policy Iteration algorithms (Sutton and Barton, 2018, Ch. 4), require a large number of mathematical operations and substantial memory. Such tabular approaches, while mathematically sound, have limited applicability because of the *curse of dimensionality*, when the required computational resources scale exponentially with the number of state variables (Bellman, 1957a).

By contrast, large MDPs can be modeled compactly if their structure is exploited in the representation (Guestrin et al., 2003) because interactions between states are fairly sparse. Subsequent research developed methods to factorize the transition model, such as Factored MDPs (Boutilier et al., 1995; Guestrin et al., 2003; Delgado et al., 2009), where the goal to factorize a problem is to decompose it into smaller objects. Factored MDPs produce compact representations of complex, uncertain systems allowing an exponential reduction in representation complexity. Such factored approaches represent states as factored states with internal structure and the state transition matrices as dynamic Bayesian networks (DBNs).

Increasing computing capacity has enabled tensor-based approaches to decompose multidimensional problems, leading to several applications in signal processing, computer vision, data mining, neuroscience, and machine learning (Kolda and Bader, 2009; Sidiropoulos et al., 2017). The term *multidimensional* is related to problems with several features. Assuming that a stochastic state transition system results from a result of a *tensor*

product, we represent MDP stochastic state transition matrices using tensor decomposition methods to compress them reduce the computational cost by improving solver runtime and memory usage of monolithic MDPs.

However, methods to solve factored DBN-based representations do not leverage advances in tensor decomposition methods to represent large monolithic/atomic MDPs. So, in this research, aiming to improve MDP solvers' efficiency, we develop an efficient tensor representation for *n*-dimensional problems. We characterize our method by a small number of components representing the state transition matrices, enabling solvers to scale up and mitigate the curse of dimensionality. We call the resulting approach CP-MDP, addressing the challenges of (i) reducing the necessary memory and the computational cost required by tabular methods to compute the solution, and (ii) leveraging advances in tensor processing to further increase solver efficiency of MDP solvers by representing the state transition matrices as tensor components. Our main contributions are (i) a formalization of MDP multi-dimensional problems using tensor algebra, (ii) an implementation of a GRIDWORLD problem using the CANDECOMP-PARAFAC (Carroll and Chang, 1970) decomposition method, (iii) implementation of compact value iteration and policy iteration algorithms, and (iv) a runtime, memory, and complexity analysis of CP-MDP method compared to tabular approaches.

This work is organized into six chapters. First, in Chapter 2, we provide background about decision theory, single-agent Markov Decision Process, factored Markov Decision Process, followed by tensor algebra and tensor decomposition. Second, in Chapter 3, we detail our proposed method. Then, we show the experimental results of our approach and compare the results to tabular approaches in Chapter 4. In Chapter 5, we contrast our approach to related work that has been developed for similar purposes. Finally, in Chapter 6, we show the contributions, limitations, and future work of this research.

2. BACKGROUND

This chapter provides the required background to understand the remaining of this dissertation. To show the fundamental basis of this research, in Section 2.1, we address the basic concepts of rational agents, actions, and environments. Followed by Section 2.1.1, in which we describe the MDP framework for sequential decision making and stochastic environments, we provide an example of a GRIDWORLD problem and the algorithms usually used to solve MDP problems: value and policy iteration. However, these algorithms use tabular computation, and the required memory and processing grow exponentially with the number of states of the problem. So, in Section 2.1.2, we address an existing method to compact the problem to reduce computational cost: a factored representation. This method illustrates large state transition matrices into compact ones using a DBN approach. However, to leverage advances in tensor processing, we use a tensor decomposition approach, called CANDECOMP-PARAFAC, as we show its concepts in Section 2.3. We show the core idea of tensor decomposition in Section 2.2, where we illustrate tensor operations, such as *tensor product*.

2.1 Decision Theory

Decision theory, which combines probability theory (Jaynes, 2003) with utility theory (Debreu, 1959), provides a formal and complete framework for decisions made under uncertainty by rational/intelligent agents. An agent perceives its environment through sensors and acts upon that environment through actuators, as illustrated in Figure 2.1. For example, a robotic agent has cameras and infrared range finders for sensors and various motors for actuators (Russell and Norvig, 2009, Ch. 2, p. 34).

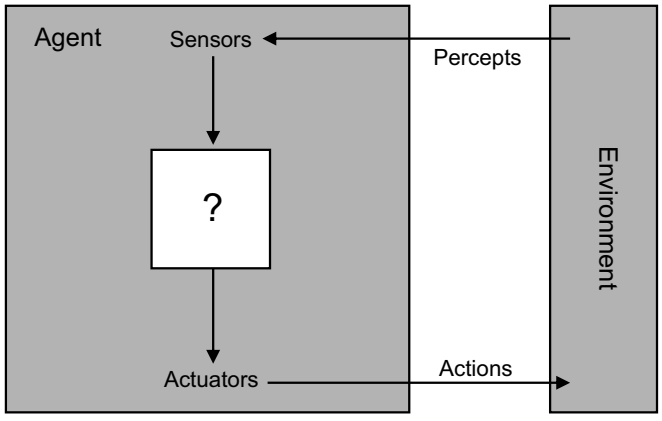


Figure 2.1: Agents interact with environments through sensors and actuators. Adapted from Russel and Norvig (Russell and Norvig, 2009, Ch. 2, p. 35).

The concept of *rationality* is applied to a wide variety of agents operating in any imaginable environment (Russell and Norvig, 2009, Ch. 2, p. 34). The fundamental idea of decision theory is that an agent is rational if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action to maximize its performance measure. This is called the principle of *maximum expected utility* (MEU). In this context, *expected* means the *average*, or *statistical mean* of the outcomes, weighted by the probability of the outcome.

The states of the environment an agent inhabits can have different definitions, such as *monolithic* and *factored* state representations. The first method, *monolithic* or *atomic* state representation, relates to where each state of the environment is indivisible and has no internal structure (Russell and Norvig, 2009, Ch. 2, p. 57). The second representation, *factored* state representation, splits up each state into a fixed set of variables or attributes, each of which can have a value. While two different atomic states have nothing in common, two different factored states can share some attributes (Russell and Norvig, 2009, Ch. 2, p. 58).

In designing an agent, we must specify the *task environment*, which is the overall problem description. The task environment is composed of four main properties: (i) the performance measure, (ii) the environment, (iii) the agent's actuators, and (iv) sensors for a given problem. We call this description PEAS, for Performance, Environment, Actuators, and Sensors (Russell and Norvig, 2009, Ch. 2, p. 40). Table 2.1 illustrates the settings of six problem examples.

Table 2.1: PEAS description of task environments. Adapted from Russel and Norvig (Russell and Norvig, 2009, Ch. 2, ps. 40,42).

Agent type	**P**erformance Measure	**E**nvironment	**A**ctuators	**S**ensors
Automated taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

As the examples show, the range of task environments that might arise in Artificial Intelligence (AI) is vast. We can, however, identify a fairly small number of dimensions for task environments, such as: (i) fully or partially observable; (ii) deterministic or stochastic; and (iii) discrete or continuous.

First, if an agent's sensors give it access to the complete state of the environment at each point in time, the task environment is (i) *fully observable*. Fully observable environ-

ments are convenient because the agent does not need to maintain any internal state to keep track of the world. An environment might be *partially observable* because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

Second, if the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is (ii) *deterministic*; otherwise, it is *stochastic* when uncertainty about outcomes is quantified in terms of probabilities. Finally, the (iii) *discrete* and *continuous* distinction applies to the state of the environment, to the way time is handled, and to the perceptions and actions of the agent. If the environment has a finite number of distinct states, the problem has a discrete set of perceptions and actions. Unlike continuous-state and continuous-time problems, where the states and actions sweep through a range of continuous values over time.

These are examples of properties that define the environment complexities and determine the appropriate agent design and the applicability for implementation (Russell and Norvig, 2009, Ch. 1, ps. 42-44). Such complexities of planning in the real world are better captured by stochastic formalisms, such as Markov Decision Processes, described in the following section.

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is a framework to model sequential decision problems for fully observable and stochastic environments with a Markovian transition model. An agent transitions through an MDP by sequential decision-making, i.e., the agent makes a series of decisions, generating a history of states in the process. MDPs have been widely used to model reinforcement learning problems, where the agent goal is to maximize the expected reward over a sequence of actions (Russell and Norvig, 2009, Ch. 17, p. 647).

An MDP is formally defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ (Puterman, 1994), where:

- $\mathcal{S}$ is the state space;
- $\mathcal{A}$ is the action space;
- $\mathcal{P}$ is a transition probability function $\mathcal{P}(s' \mid s, a)$;
- $\mathcal{R}$ is a reward function; and
- $\gamma \in [0..1]$ is a discount factor.

States in MDPs are treated as monolithic representations (Russell and Norvig, 2009, Ch. 2, p. 58), and the state transitions and decision making in MDPs are characterized by (i) a stochastic transition system, which determines the probabilities to which

state the decision-making agent reaches after taking action, and (ii) the *Markov property*, which dictates every transition between states depends exclusively on the last visited state, rather than the history of states before that (Sutton and Barton, 2018, Ch. 3, p. 49).

More specifically, at each time step t the agent interacts with the environment by taking an action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$. As a consequence, the agent receives a reward r_{t+1} (also known as r') $\in \mathcal{R}$ and reaches a new state s_{t+1} (also known as s') $\in \mathcal{S}$ with probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ given the transition probability function (Sutton and Barton, 2018, Ch. 3, p. 48). The discount factor γ is a multiplicative term that determines how much to discount the value of future decisions compared to an equivalent decision at the current time, in other words, it describes the preference of an agent for current rewards over future rewards (Russell and Norvig, 2009). A γ value close to 0 leads to "myopic" evaluation, i.e., future decisions hold no value. Whereas a γ value close to 1 leads to a "far-sighted" evaluation. Figure 2.2 illustrates the agent-environment interaction in an MDP.

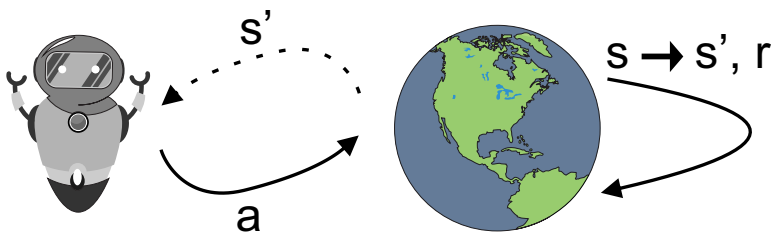
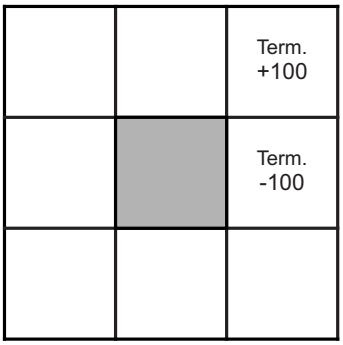


Figure 2.2: Schematic representation of an MDP. At every stage, the agent takes action and observes the resulting state s' (or s_{t+1}). Adapted from Oliehoek and Amato (Oliehoek and Amato, 2016, Ch. 2, p. 12).

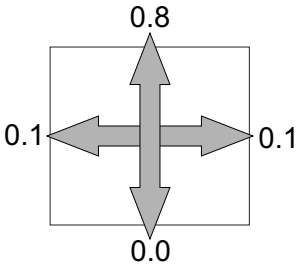
Figure 2.3 shows a GRIDWORLD representation of a simple MDP, where the interaction with the environment terminates when the agent reaches one of the terminal states, marked by +100 or –100 rewards. The agent actions in a given state, in this example, are *North*, *South*, *West*, and *East*, where the expected outcome occurs with a probability of 0.8. Still, with a probability of 0.1, the agent moves at right angles to the intended state, and with a probability of 0.0, the agent moves in the opposite direction, illustrated by Figure 2.3b. A collision with a wall (the shaded square and the limits of the environment) results in no movement (Russell and Norvig, 2009, Ch. 17, p. 645-646).

A transition model describes the stochastic outcome of each action in each state, denoted by $\mathcal{P}(s_{t+1}|s_t, a_t)$, to determine the probability of reaching state s_{t+1} if action a is taken in state s. The transition model of Figure 2.3b is given by a large state transition matrix containing probabilities, composing four 9×9 matrices, one for each action (*North*, *South*, *West*, and *East*), as we show in Table 2.2.

To complete the environment's definition, we must specify the utility function for the agent (Russell and Norvig, 2009, Ch. 17, p.646). A utility function *V(s)* is how the agent's preferences are captured, assigning a single number to express the desirability of a state (Russell and Norvig, 2009, Ch. 16, p.621). Because the decision problem is sequential, the



(a) GRIDWORLD



(b) Transition model illustration

Figure 2.3: A GRIDWORLD 3×3 environment that presents the agent with a sequential decision problem. And an illustration of the transition model of the environment. Adapted from Russel and Norvig (Russell and Norvig, 2009, Ch. 17, p. 646).

utility function depends on a sequence of states (an environment history) rather than on a single state, i.e., in each state s, the agent receives a reward $R(s)$, which may be positive or negative (Russell and Norvig, 2009, Ch. 17, p.646). An agent behaves in an environment by following a policy that maps states to actions. A policy is a function that maps states to actions for the MDP that defines the optimal (or maximum expected utility) action for every state in the domain. After taking action at a given state, an immediate reward is given to the agent as feedback from the environment (Sutton and Barton, 2018, Ch. 1, p. 6). A policy with the highest expected reward or the expected reward equal to the last generated policy is called the optimal policy. It maximizes the reward an agent receives over the long run. (π^*), i.e., the optimal solution (Sutton and Barton, 2018, Ch. 3, p. 62). Most algorithms to solve MDPs seek to find a policy that assigns an (optimal) action choice for each agent at each state (Guestrin, 2003).

Value Iteration and *Policy Iteration* are common dynamic programming algorithms to solve MDPs (Guestrin, 2003). The basic idea of value iteration is to compute each state's utility and then use the state utilities to select an optimal action in each state (Bellman, 1954). The utility of a state is the immediate reward for this state plus the expected discounted utility of neighboring states, assuming the agent chooses the optimal action, i.e., the utility is an expected value of any stochastic transition by multiplying the value of each outcome by its probability. It is estimated by a value function ($\mathcal{V}$), and it returns a value computed based on the amount of reward an agent might expect from future rewards for each state. The *Bellman Equation* (Bellman, 1957b), after Richard Bellman, formalizes this model in Equation 2.1.

$$\mathcal{V}(s) = \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s' \mid s, a)\mathcal{V}(s') \tag{2.1}$$

The Bellman equation is the basis of the value iteration algorithm for solving MDPs (Russell and Norvig, 2009, Ch. 17, p. 652). Value iteration (Algorithm 2.1) propagates

Table 2.2: State transition matrix of actions *North*, *South*, *West*, and *East* for the 3×3 grid example.

0.9	0.1	0.	0.	0.	0.	0.	0.	0.
0.1	0.8	0.1	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.8	0.	0.	0.2	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.8	0.	0.	0.1	0.1	0.
0.	0.	0.	0.	0.	0.	0.1	0.8	0.1
0.	0.	0.	0.	0.	0.8	0.	0.1	0.1

(a) North

0.1	0.1	0.	0.8	0.	0.	0.	0.	0.
0.1	0.8	0.1	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.2	0.	0.	0.8	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.9	0.1	0.
0.	0.	0.	0.	0.	0.	0.1	0.8	0.1
0.	0.	0.	0.	0.	0.	0.	0.1	0.9

(b) South

0.9	0.	0.	0.1	0.	0.	0.	0.	0.
0.8	0.2	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.	0.	0.8	0.	0.	0.1	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.1	0.	0.	0.9	0.	0.
0.	0.	0.	0.	0.	0.	0.8	0.2	0.
0.	0.	0.	0.	0.	0.1	0.	0.8	0.1

(c) West

0.1	0.8	0.	0.1	0.	0.	0.	0.	0.
0.	0.2	0.8	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.	0.	0.8	0.	0.	0.1	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.1	0.	0.	0.1	0.8	0.
0.	0.	0.	0.	0.	0.	0.	0.2	0.8
0.	0.	0.	0.	0.	0.1	0.	0.	0.9

(d) East

information through the state space iteratively through local updates until it converges to the optimal value, from which the optimal policy is extracted.

Unlike value iteration, policy iteration alternates in two steps: policy evaluation and policy improvement (Algorithm 2.2). Policy evaluation is the process of computing the value function $\mathcal{V}_\pi$ for a policy π. The algorithms for policy evaluation and finding optimal value function are highly similar except for a *max* operation. Whereas policy improvement is the process of generating a new policy, such that $\mathcal{V}_{\pi'}(s) \geq \mathcal{V}_\pi(s)$, by acting greedily to π (Sutton and Barton, 2018, Ch. 4). The algorithm terminates when the policy improvement step yields no change in the policy. At this point, the policy is a solution to the Bellman equation, and π must be an optimal policy (Russell and Norvig, 2009, Ch. 17, p. 656).

Summing up, in value iteration, the algorithm starts with a random value function. Then it finds a new and improved value function using the Bellman operator in an iterative process until it reaches the optimal value function. Basically, value iteration is composed of two steps: (i) finding an optimal value function and (ii) one policy extraction. And in policy iteration, the algorithm starts with a random policy; then, it finds the value function of that policy using the Bellman operator. This process is called the *policy evaluation* step. Then, it finds a new and improved policy based on the previous value function. In this process, which is called *policy improvement* step, each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). So, policy iteration is divided into two steps (i)

Algorithm 2.1: Value iteration algorithm for calculating utilities of states. Adapted from Russel and Norvig (Russell and Norvig, 2009, Ch. 17, p. 653).

inputs: an MDP with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$(s), transition model $\mathcal{P}(s'|s,a)$, rewards $r \in \mathcal{R}(s)$, and discount γ;
ϵ: maximum error allowed in the utility of any state.

local variables:
$\mathcal{V}$ and $\mathcal{V}'$: vectors of utilities for states in $\mathcal{S}$, initially zero;
δ: the maximum change in the utility of any state in an iteration.

procedure VALUEITERATION(mdp, ϵ) returns a utility function
  repeat
    $\mathcal{V} \leftarrow \mathcal{V}'$; $\delta \leftarrow 0$
    for each state s in $\mathcal{S}$ **do**
      $\mathcal{V}'[s] \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s'|s,a) \cdot \mathcal{V}[s']$
      if $|\mathcal{V}'[s] - \mathcal{V}[s]| > \delta$ **then** $\delta \leftarrow |\mathcal{V}'[s] - \mathcal{V}[s]|$
  until $\delta <$ ϵ (1 - γ)/γ
return $\mathcal{V}$

Algorithm 2.2: The policy iteration algorithm for calculating an optimal policy. Adapted from Russel and Norvig (Russell and Norvig, 2009, Ch. 17, p. 653).

inputs: an MDP with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$(s), transition model $\mathcal{P}(s'|s,a)$

local variables:
$\mathcal{V}$: a vector of utilities for states in $\mathcal{S}$, initially zero;
π: a policy vector indexed by state, initially random.

procedure POLICY-ITERATION(*mdp*) returns a policy
  repeat
    $\mathcal{V} \leftarrow$ Policy-Evaluation($\pi, \mathcal{V}$, *mdp*)
    unchanged? $\leftarrow$ true
    for each state *s* in $\mathcal{S}$ **do**
      if $\max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s'|s,a) \cdot \mathcal{V}[s'] > \sum_{s'} \mathcal{P}(s'|s,\pi[s]) \cdot \mathcal{V}[s']$ **then**
        $\pi[s] \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}(s'|s,a) \cdot \mathcal{V}[s']$
        unchanged? $\leftarrow$ false
  until *unchanged?*
return π

policy evaluation and (ii) policy improvement, and the two are repeated iteratively until policy converges.

Dynamic programming algorithms, such as Value and Policy Iteration, are of limited applicability because of the *curse of dimensionality* (number of states often grows exponentially with the number of state variables) (Sutton and Barton, 2018, Ch. 4, p. 87). However, these algorithms are considered computationally viable ways of solving large stochastic problems in a reasonable time compared to other programming methods, such as linear programming algorithms, which become impractical at a much smaller number of states (Sutton and Barton, 2018, Ch. 4, p. 87).

2.1.2 Factored Markov Decision Process

In this section, we briefly introduce Factored MDPs, a method to represent large MDPs in a compact representation using factored state representations instead of monolithic ones. The idea of representing a large MDP using a factored model is first proposed by Boutilier et al. (Boutilier et al., 1995), where the state in an assignment to multiple state variables and to a transition function that compactly specifies the probabilistic dependence of variables in the next state on a subset of variables in the current state.

Factored MDP allows us to represent complex, uncertain dynamic systems compactly by exploiting problem-specific structure (Guestrin, 2003). The benefit of using such a representation is the state transition model can be compactly represented using one of several methods, the most common being a Dynamic Bayesian Network (DBN). A DBN is a *directed acyclic graph* with two layers: one layer represents the current state's variables, and the other layer represents the next state (Delgado et al., 2009). This technique allows a compact representation of the transition model by exploiting the fact that a variable's transition often depends only on a small number of other variables (Guestrin et al., 2003). Factored MDPs often allow for an exponential reduction in representation complexity and can model quite substantial real-world problems.

2.2 Tensor Algebra

In the second half of the 19th century, Leopold Kronecker, a German mathematician, proposed a new tensor-based operation, a generalization of matrices where more than two dimensions are represented. This extension of Multilinear Algebra, called Tensor Algebra, was represented by an operator called the *Kronecker product* or *tensor product* (Davio, 1981). Since then, many researchers use Kronecker's product to represent operations on multidimensional structures. However, not until the late 1960s did computer scientists pay

attention to Kronecker extensions to Tensor Algebra. Related work, such as Bellman (Bellman, 1960) is one of the first studies concerning Kronecker's product applied to Computer Science.

Tensors are generalizations of matrices to higher dimensions; therefore, we represent tensors as multidimensional arrays (Kolda and Bader, 2009). An N^{th}-order tensor is an element of a tensor product of N vector spaces, each of which has its coordinate system. The *order* of a tensor is the number of dimensions. A first-order tensor is a vector that contains one index, a second-order tensor is a matrix that contains two indexes, and tensors of order three or higher are called higher-order tensors, where a third-order tensor has three indexes (Kolda and Bader, 2009). Tensor algebra contemplates tensor-based operations, being one of them the *tensor product*, also known as *Kronecker product* (Bellman, 1960). We show an example of the tensor product of two matrices $A \otimes B$ and the resulting tensor C in Equation 2.2 .

$$A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \end{pmatrix} \tag{2.2}$$

$$C = \begin{pmatrix} a_{00}B & a_{01}B \\ a_{10}B & a_{11}B \end{pmatrix} \tag{2.3}$$

$$C = \left(\begin{array}{cccc|cccc} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} & a_{00}b_{03} & a_{01}b_{00} & a_{01}b_{01} & a_{01}b_{02} & a_{01}b_{03} \\ a_{00}b_{10} & a_{00}b_{11} & a_{00}b_{12} & a_{00}b_{13} & a_{01}b_{10} & a_{01}b_{11} & a_{01}b_{12} & a_{01}b_{13} \\ a_{00}b_{20} & a_{00}b_{21} & a_{00}b_{22} & a_{00}b_{23} & a_{01}b_{20} & a_{01}b_{21} & a_{01}b_{22} & a_{01}b_{23} \\ \hline a_{10}b_{00} & a_{10}b_{01} & a_{10}b_{02} & a_{10}b_{03} & a_{11}b_{00} & a_{11}b_{01} & a_{11}b_{02} & a_{11}b_{03} \\ a_{10}b_{10} & a_{10}b_{11} & a_{10}b_{12} & a_{10}b_{13} & a_{11}b_{10} & a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} \\ a_{10}b_{20} & a_{10}b_{21} & a_{10}b_{22} & a_{10}b_{23} & a_{11}b_{20} & a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} \end{array}\right)$$

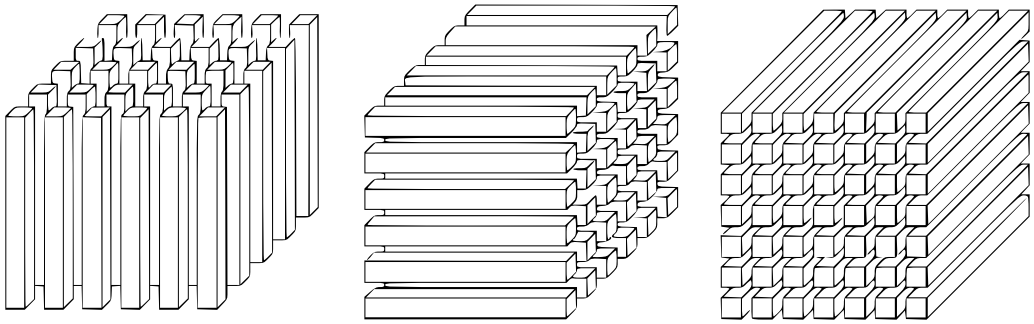
In general, to define the tensor product of two matrices with A dimensions $(\rho_1 \times \omega_1)$ and B dimensions $(\rho_2 \times \omega_2)$, a convenient observation is the resulting matrix of the tensor product: $(\rho_1\rho_2 \times \omega_1\omega_2)$ dimensions. Which can be considered as composed of $\rho_1\omega_1$ blocks, the dimensions of A, each with dimensions $(\rho_2\omega_2)$, the dimensions of B.

The tensor product $C = A \otimes B$ is defined by assigning an element value $a_{ij}\ b_{kl}$ to the position (k, l) in block (i, j), as Equation 2.4 shows. This representation of matrix elements corresponds to a tensor product.

$$c_{[ik],[jl]} = a_{ij}b_{kl}, \textit{ where } i \in [1..\alpha_1],\ j\,[1..\alpha_2],\ k\,[1..\beta_1] \textit{ and } l\,[1..\beta_2] \tag{2.4}$$

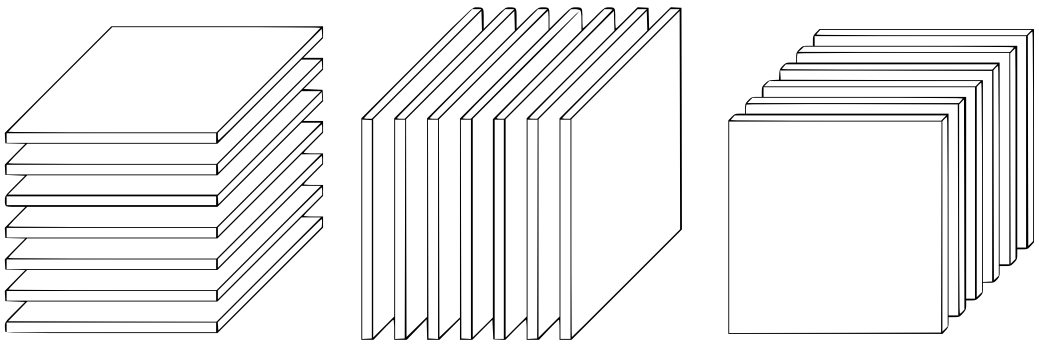
2.3 Tensor Decomposition

In order to determine the tensor decomposition definition, first, we address the (i) *fiber* and (ii) *slice* definitions. (i) *Fibers* describe tensors as a collection of vectors. Fibers are the higher-order analog of matrix rows and columns. A fiber is defined by fixing every index but one. Third-order tensors have column, row, and tube fibers, denoted by $x_{:jk}$, $x_{i:k}$, and $x_{ij:}$, respectively Kolda and Bader (2009). Figure 2.4 represents the fibers of a third-order tensor. (ii) *Slices* represent tensors as a collection of matrices. Slices are two-dimensional sections of a tensor, defined by fixing all indexes but two. Third-order tensors have horizontal, lateral, and frontal slices, denoted by $X_{i::}$, $X_{:j:}$, and $X_{::k}$, respectively (Kolda and Bader, 2009). Figure 2.5 represents the slices of a third-order tensor.



(a) Column. (b) Row. (c) Tube.

Figure 2.4: Fibers of a three-dimensional tensor. (a) First dimension: column; (b) Second dimension: row; (c) Third dimension: tube. Adapted from Kolda and Bader (Kolda and Bader, 2009).



(a) Horizontal slices. (b) Lateral slices. (c) Frontal slices.

Figure 2.5: Slices of a three-dimensional tensor. (a) First dimension: horizontal slices; (b) Second dimension: lateral slices; (c) Third dimension: frontal slices. Adapted from Kolda and Bader (Kolda and Bader, 2009).

The concepts of fibers and slices are called *tensor indexing*. Tensor indexing creates sub-arrays (or sub-fields) by indexing some of the given tensor indexes. Consequently,

it can operate with contracted indexes (Rabanser et al., 2017). The creation of sub-arrays leads to the main idea of tensor decomposition, which is a method to extract and explain data properties. We can consider these methods to be higher-order generalizations of matrix singular value decomposition (SVD) and principal component analysis (PCA) (Kolda and Bader, 2009).

Canonical Polyadic Decomposition with Parallel Factors, also known as CANDECOMP-PARAFAC decomposition or CP decomposition (Carroll and Chang, 1970; Harshman, 1972; Kiers, 1998) is an example of the tensor decomposition method. Carroll and Chang (Carroll and Chang, 1970) introduce the CANDECOMP (canonical decomposition), and Harshman (Harshman, 1970) introduces the PARAFAC (parallel factors). Kiers describes the use of both methods (CANDECOMP-PARAFAC)(Kiers, 1998). CANDECOMP-PARAFAC is the process that factorizes/decomposes a tensor into sums of individual components, providing a parallel proportional analysis and an idea of multiple axes for analysis (Kolda and Bader, 2009). This method expresses a tensor as a sum of the tensor product of vectors. We illustrate a CANDECOMP-PARAFAC decomposition of a third-order tensor $\mathcal{X}$ in Figure 2.6.

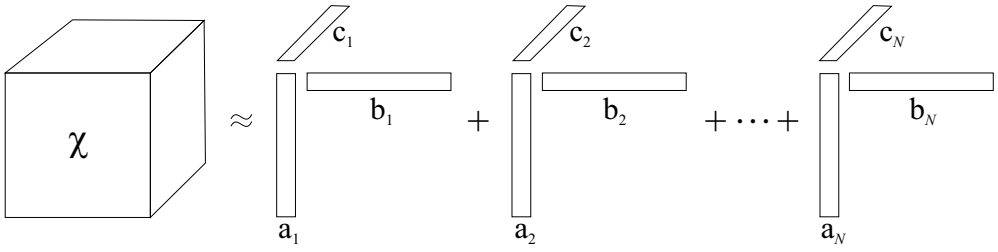


Figure 2.6: The CANDECOMP-PARAFAC decomposition representation.

As we show in the third-order tensor example $\mathcal{X} \in \mathbb{N}^{I \times J \times K}$, we can express the sum of individual components as follows:

$$\mathcal{X} \approx \sum_{n=1}^{N} (a_n \otimes b_n \otimes c_n) \tag{2.5}$$

where N is a positive integer and $a_n \in \mathbb{N}^I$, $b_n \in \mathbb{N}^J$, and $c_n \in \mathbb{N}^K$ for n = 1, ..., N. Therefore, we can describe Equation 2.5 as follows:

$$x_{ijk} \approx \sum_{n=1}^{N} (a_{in} b_{jn} c_{kn}) \text{ for } i = 1, ..., I; j = 1, ..., J; k = 1, ..., K.$$

Therefore, aiming to reduce runtime and memory usage for MDPs solvers, the state transition matrices of MDPs can be taught as sums of tensor components, as Figure 2.6 illustrates.

3. TENSOR-BASED MDP DECOMPOSITION

In this chapter, we introduce our tensor-based MDP decomposition. First, we introduce an MDP tensor algebra formalization in Section 3.1. Each definition is followed by an example using a two-dimensional GRIDWORLD problem. This formalization allows us to represent multidimensional MDP problems with discrete states. Second, we introduce a method to decompose MDP state transition matrices using CANDECOMP-PARAFAC tensor decomposition, which we call CP-MDP. We develop the tensor decomposition method to compress large state transition matrices into small arrays, which we call *tensor components*. Third, in Section 3.3, we show four algorithms to operate in the tensor decomposition representation. The first one computes a matrix of state transition probabilities. The second computes the tensor components so that these tensor components can be used later by the third and fourth algorithms: the compact value and policy iteration algorithms of our approach (CP-MDP-VI and CP-MDP-PI). Instead of performing computations over the large matrices required by tabular representations, the CP-MDP-VI and CP-MDP-PI algorithms compute the solution using compact arrays, consequently reducing solver computational cost. Section 3.4 computes the computational cost to generate the state transition matrices for the tabular representation, comparing with the computation to generate the compact tensor components. Then, we compare the computational cost of actually solving the problem using tabular value and policy iteration against CP-MDP-VI and CP-MDP-PI. Finally, in Section 3.5, to illustrate a more complex example, we show a three-dimensional GRIDWORLD problem with the tensor algebra formalization.

3.1 Tensor Algebra Formalization

We now provide a Markov Decision Process formalization in terms of *n*-dimensional values using Tensor Algebra concepts. We use the term *n-dimensional* to express problems with multiple features.

Definition 1 (Dimensions $\mathcal{D}$) *Let $\mathcal{D}$ be the set of environment dimensions, where $\mathcal{D} = \{d_1, d_2, d_3, \ldots, d_D\}$ and $D = |\mathcal{D}|$[1] is the number of dimensions. Given the set of dimensions $\mathcal{D}$, it contains $|\mathcal{D}|$ dimensions d, where $d \in [1..D]$[2].*

For example, in Figure 3.1 we show a two-dimensional MDP problem, where $\mathcal{D} = \{\mathbb{X}_1, \mathbb{Y}_2\}$ and $|\mathcal{D}| = 2$.

[1] The notation adopted is $|\mathcal{X}|$ to define the cardinality of a set.

[2] The notation adopted is $[i..j]$ referring to a number in the range from i to j, inclusive, belonging to the set of natural numbers.

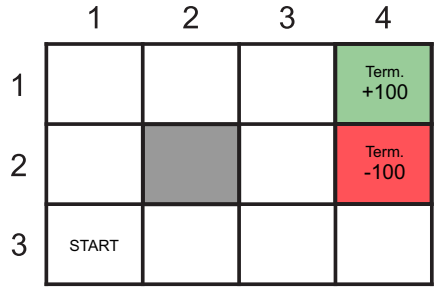


Figure 3.1: A two-dimensional 4×3 GRIDWORLD example.

Definition 2 (State-space $\mathcal{S}$) *The state-space $\mathcal{S}$ is set of states of all dimensions, where $\mathcal{S} = \{s_1, s_2, s_3, ..., s_S\}$, such that $S = |\mathcal{S}|$ is the number of states. We determine the number of states by multiplying the cardinality of each dimension, as follows:*

$$|\mathcal{S}| = |d_1| \times |d_2| \times |d_3| \times ... \times |d_D|. \tag{3.1}$$

For example, for a two-dimensional MDP problem with cardinality equal to 4×3, the number of resulting states is $|\mathcal{S}| = 12$.

$$\mathcal{S} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

Or, the state-space can be written as tuples, as follows:

$$\mathcal{S} = \{(x_1, y_1)_{s_1}, (x_2, y_1)_{s_2}, (x_3, y_1)_{s_3}, (x_4, y_1)_{s_4}, (x_1, y_2)_{s_5}, (x_2, y_2)_{s_6}, \\ (x_3, y_2)_{s_7}, (x_4, y_2)_{s_8}, (x_1, y_3)_{s_9}, (x_2, y_3)_{s_{10}}, (x_3, y_3)_{s_{11}}, (x_4, y_3)_{s_{12}}\}.$$

Definition 3 (Actions $\mathcal{A}$) *Let $\mathcal{A}$ be the set of actions, where $\mathcal{A} = \{a_1, a_2, a_3, ..., a_A\}$ and $A = |\mathcal{A}|$ is the number of actions.*

For example, for a GRIDWORLD problem, assuming a Euclidean space (De Risi, 2016), we consider two actions for each dimension. So, for a two-dimensional problem, the resulting number of actions is 4, where $\mathcal{A} = (2 \times 2)$. We name these four actions as *West* and *East* for the x-axis, and *North* and *South* for the y-axis.

Definition 4 (Obstacle states $\mathcal{O}$) *Let $\mathcal{O}$ be the set of obstacle states, where $\mathcal{O} = \{o_1, o_2, o_3, ..., o_O\}$ and $O = |\mathcal{O}|$ is the number of obstacle states. Therefore, $\mathcal{O} \subset \mathcal{S}$ and $\mathcal{O} \neq \mathcal{T}$.*

For example, as we illustrate in Figure 3.1, $\mathcal{O} = \{6\}$, and $|\mathcal{O}| = 1$.

Definition 5 (Terminal states $\mathcal{T}$) *Let $\mathcal{T}$ be the set of terminal states, where $\mathcal{T} = \{t_1, t_2, t_3, ..., t_T\}$ and $T = |\mathcal{T}|$ is the number of terminal states. Therefore, $\mathcal{T} \subset \mathcal{S}$ and $\mathcal{T} \neq \mathcal{O}$.*

For example, as we illustrate in Figure 3.1, $\mathcal{T} = \{4, 8\}$, and $|\mathcal{T}| = 2$.

Definition 6 (Rewards $\mathcal{R}$) *Let $\mathcal{R}$ be the set of rewards, which include real numbers, where $\mathcal{R}$ = {$r_1, r_2, r_3, \ldots, r_R$} and R = $|\mathcal{R}|$ is the number of rewards. $|\mathcal{R}|$ is the same number as the state-space $|\mathcal{S}|$.*

For example, taking into account a $|\mathcal{R}| = 12$, we determine a reward of -3 for non-terminal states, and 100 or -100 for terminal states, as follows:

$$\mathcal{R} = \{-3, -3, -3, 100, -3, -3, -3, -100, -3, -3, -3, -3\}.$$

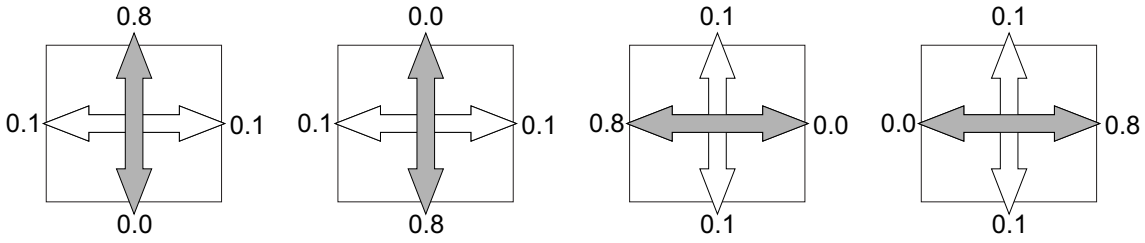
To represent the state transition probability, we formalize the *state transition probability matrix* in Definition 7.

Definition 7 (Probabilities of state transition $\mathcal{P}$) *Let $\mathcal{P}$ be the state transition probability system of going from state s to s′, where $\mathcal{P}_{ss'} = [\mathcal{S}_{t+1} = s' \mid \mathcal{S}_t = s]$. The probabilities are defined as $\mathcal{P} = \{p_1, p_2, p_3, \ldots, p_P\}$, and P = $|\mathcal{P}|$ is the number of state transition probabilities. The state transition $\mathcal{P}$ can be specified as a matrix as we show in Equation 3.2, where $|\mathcal{A}|$ is the number of actions.*

$$|\mathcal{P}| = |\mathcal{A}| \times |\mathcal{A}|. \tag{3.2}$$

For example, consider a 4-action MDP problem (*N*, *S*, *W*, and *E*) and probabilities of (i) 0.8 of ending in the intended state, (ii) 0.1 of going to the right angles of the intended state (90° angles), and (iii) 0.0 of going to the opposite direction of the intended state, as we illustrate in Figure 3.2. The resulting number of probabilities is $|\mathcal{P}| = 16$, as follows:

$$\mathcal{P} = \{0.8, 0.0, 0.1, 0.1, 0.0, 0.8, 0.1, 0.1, 0.1, 0.1, 0.8, 0.0, 0.1, 0.1, 0.0, 0.8\}.$$



(a) North. (b) South. (c) West. (d) East.

Figure 3.2: State transition matrix of actions *North*, *South*, *West*, and *East* illustration.

Definition 8 (Limits $\mathcal{L}$) *Let $\mathcal{L}$ be the set of limits to delimit the environment (collision with walls), where $\mathcal{L} = \{[\alpha_{d_1}, \Omega_{d_1}]_1, [\alpha_{d_2}, \Omega_{d_2}]_2, \ldots, [\alpha_{d_D}, \Omega_{d_D}]_L\}$, and L = $|\mathcal{L}|$ is the number of the environment limits, where $|\mathcal{L}| = |\mathcal{D}|$ is equal to the number of dimensions. We consider for each dimension a tuple of an* initial limit *defined by α (the minimum value), and a* final limit *defined by Ω (the maximum value).*

This definition follows the Euclidean assumption, where it assumes any line contains at least two points (De Risi, 2016). For example, for each dimension $\mathbb{X}$ and $\mathbb{Y}$, we define a tuple of initial and final limits as follows, where $|\mathcal{L}| = 2$:

$$\mathcal{L} = \{[1, 3], [1, 4]\}.$$

To improve computational cost, in Definition 9, we propose an optimization to eliminate any explicit representation of zero-probability of state transition, limiting the neighborhood of each state.

Definition 9 (Probabilities of state transition > 0 $\overline{\mathcal{P}_{(a)}}$) *Considering state transition probabilities where $\mathcal{P}(s'|s, a) > 0$, $\overline{\mathcal{P}_{(a)}}$ is the set of probabilities of non-zero state transition probabilities (i.e. probabilities greater than zero), where $\overline{\mathcal{P}_{(a)}} = \{p_1, p_2, p_3, ..., p_{\overline{P_{(a)}}}\}$, and $\overline{P_{(a)}} = |\overline{\mathcal{P}_{(a)}}|$ is the number of non-zero state transition probabilities. Therefore, $\overline{\mathcal{P}_{(a)}} \subset \mathcal{P}$. Consequently, this definition limits the neighborhood of each state.*

For example, using a two-dimensional problem with four actions, we consider only probabilities of state transition, where $\mathcal{P}(s'|s, a) > 0$. Therefore, $|\overline{\mathcal{P}_{(a)}}| = 3$ for each action.

$$\overline{\mathcal{P}_{(N)}} = \{0.8, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(S)}} = \{0.8, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(W)}} = \{0.1, 0.1, 0.8\};$$
$$\overline{\mathcal{P}_{(E)}} = \{0.1, 0.1, 0.8\}.$$

In order to precompute a state s_{t+1} of a current state s_t, given an action a, we define the concept of *successor states* by Definition 10.

Definition 10 (Successor states $\mathcal{S}'_{(a)}$) *Let $\mathcal{S}'_{(a)}$ be the set of successor states (s'), given an action (a), where $\mathcal{S}'_{(a)} = \{s'_1, s'_2, s'_3, ..., s'_{S'_{(a)}}\}$, and $S'_{(a)} = |\mathcal{S}'_{(a)}|$ is the number of successor states for each action (a). We define $|\mathcal{S}'_{(a)}|$ by Equation 3.3.*

$$|\mathcal{S}'_{(a)}| = |\mathcal{S}| \times |\overline{\mathcal{P}_{(a)}}|. \tag{3.3}$$

For example, considering a $|\overline{\mathcal{P}_{(a)}}| = 3$ and $\mathcal{S} = 12$, the following are the successors states for each actions a, where $|\mathcal{S}'_{(a)}| = 36$:

$$\mathcal{S}'_{(N)} = \{1', 1', 2', 2', 1', 3', 3', 2', 4', ..., 8', 11', 12'\};$$
$$\mathcal{S}'_{(S)} = \{5', 1', 2', 2', 1', 3', 7', 2', 4', ..., 12', 11', 12'\};$$
$$\mathcal{S}'_{(W)} = \{1', 5', 1', 2', 2', 1', 3', 7', 2', ..., 8', 12', 11'\};$$
$$\mathcal{S}'_{(E)} = \{1', 5', 2', 2', 2', 3', 3', 7', 4', ..., 8', 12', 12'\}.$$

Definition 11 (Successor states of s $\mathcal{S}'^{s}_{(a)}$) *Let $\mathcal{S}'^{s}_{(a)}$ be the set of successor state given a state (s) and an action (a), where $\mathcal{S}'^{s}_{(a)} = \{s'_1, s'_2, s'_3, \dots, s'_{S'^{s}_{(a)}}\}$, and $S'^{s}_{(a)} = |\mathcal{S}'^{s}_{(a)}|$ is the number of successor states of each state (s). We define $|\mathcal{S}'^{s}_{(a)}|$ as the same number of state transition probabilities greater than zero, as $|\mathcal{S}'^{s}_{(a)}| = |\overline{\mathcal{P}_{(a)}}|$. Therefore, $\mathcal{S}'^{s}_{(a)} \subset \mathcal{S}'_{(a)}$.*

For example, considering $|\overline{\mathcal{P}_{(a)}}| = 3$ and $\mathcal{S} = 12$, the following are the successors states for each state given an action *a*, where $|\mathcal{S}'^{s}_{(a)}| = 3$.

$\mathcal{S}'^{s_1}_{(N)} = \{1', 1', 2'\}$;	$\mathcal{S}'^{s_1}_{(S)} = \{5', 1', 2'\}$;	$\mathcal{S}'^{s_1}_{(W)} = \{1', 5', 1\}$;	$\mathcal{S}'^{s_1}_{(E)} = \{1', 5', 2'\}$;
$\mathcal{S}'^{s_2}_{(N)} = \{2', 1', 3'\}$;	$\mathcal{S}'^{s_2}_{(S)} = \{2', 1', 3'\}$;	$\mathcal{S}'^{s_2}_{(W)} = \{2', 2', 1'\}$;	$\mathcal{S}'^{s_2}_{(E)} = \{2', 2', 3'\}$;
$\mathcal{S}'^{s_3}_{(N)} = \{3', 2', 4'\}$;	$\mathcal{S}'^{s_3}_{(S)} = \{7', 2', 4'\}$;	$\mathcal{S}'^{s_3}_{(W)} = \{3', 7', 2'\}$;	$\mathcal{S}'^{s_3}_{(E)} = \{3', 7', 4'\}$;
$\mathcal{S}'^{s_4}_{(N)} = \{4', 4', 4'\}$;	$\mathcal{S}'^{s_4}_{(S)} = \{4', 4', 4'\}$;	$\mathcal{S}'^{s_4}_{(W)} = \{4', 4', 4'\}$;	$\mathcal{S}'^{s_4}_{(E)} = \{4', 4', 4'\}$;
$\mathcal{S}'^{s_5}_{(N)} = \{1', 5', 5'\}$;	$\mathcal{S}'^{s_5}_{(S)} = \{9', 5', 5'\}$;	$\mathcal{S}'^{s_5}_{(W)} = \{1', 9', 5'\}$;	$\mathcal{S}'^{s_5}_{(E)} = \{1', 9', 5'\}$;
$\mathcal{S}'^{s_6}_{(N)} = \{6', 6', 6'\}$;	$\mathcal{S}'^{s_6}_{(S)} = \{6', 6', 6'\}$;	$\mathcal{S}'^{s_6}_{(W)} = \{6', 6', 6'\}$;	$\mathcal{S}'^{s_6}_{(E)} = \{6', 6', 6'\}$;
$\mathcal{S}'^{s_7}_{(N)} = \{3', 7', 8'\}$;	$\mathcal{S}'^{s_7}_{(S)} = \{11', 7', 8'\}$;	$\mathcal{S}'^{s_7}_{(W)} = \{3', 11', 7'\}$;	$\mathcal{S}'^{s_7}_{(E)} = \{3', 11', 8'\}$;
$\mathcal{S}'^{s_8}_{(N)} = \{8', 8', 8'\}$;	$\mathcal{S}'^{s_8}_{(S)} = \{8', 8', 8'\}$;	$\mathcal{S}'^{s_8}_{(W)} = \{8', 8', 8'\}$;	$\mathcal{S}'^{s_8}_{(E)} = \{8', 8', 8'\}$;
$\mathcal{S}'^{s_9}_{(N)} = \{5', 9', 10'\}$;	$\mathcal{S}'^{s_9}_{(S)} = \{9', 9', 10'\}$;	$\mathcal{S}'^{s_9}_{(W)} = \{5', 9', 9'\}$;	$\mathcal{S}'^{s_9}_{(E)} = \{5', 9', 10'\}$;
$\mathcal{S}'^{s_{10}}_{(N)} = \{10', 9', 11'\}$;	$\mathcal{S}'^{s_{10}}_{(S)} = \{10', 9', 11'\}$;	$\mathcal{S}'^{s_{10}}_{(W)} = \{10', 10', 9'\}$;	$\mathcal{S}'^{s_{10}}_{(E)} = \{10', 10', 11'\}$;
$\mathcal{S}'^{s_{11}}_{(N)} = \{7', 10', 12'\}$;	$\mathcal{S}'^{s_{11}}_{(S)} = \{11', 10', 12'\}$;	$\mathcal{S}'^{s_{11}}_{(W)} = \{7', 11', 10'\}$;	$\mathcal{S}'^{s_{11}}_{(E)} = \{7', 11', 12'\}$;
$\mathcal{S}'^{s_{12}}_{(N)} = \{8', 11', 12'\}$;	$\mathcal{S}'^{s_{12}}_{(S)} = \{12', 11', 12'\}$;	$\mathcal{S}'^{s_{12}}_{(W)} = \{8', 12', 11'\}$;	$\mathcal{S}'^{s_{12}}_{(E)} = \{8', 12', 12'\}$.

Using a formalization with tensor algebra, we exploit the representation in a way MDP becomes more compact. By creating functions that precompute the actions, obstacles, terminals, rewards, the state transition probability matrix, and successor states of a problem, we minimize the computational cost to solve an MDP problem.

3.2 CP-MDP Representation

We now create a compact representation of MDP state transition matrices using tensor decomposition, which we call CP-MDP. Our method consists of decomposing MDP state transition matrices into small tensor components using the CANDECOMP-PARAFAC decomposition fundamental idea as the basic semantics of this work to represent a tensor as a sum of arrays.

Definition 12 (Tensor components $\mathcal{C}_{(a)}$) *Let $\mathcal{C}_{(a)}$ be a tensor of components of action (a). Each action constitutes a third-order tensor with tensor components, regardless of the problem's number of dimensions. Each tensor component is composed of three dimensions:*

- *(1) a current state $s \in \mathcal{S}$;*
- *(2) a successor state $s' \in \mathcal{S}'_{(a)}$; and*
- *(3) a probability of state transition $p \in \overline{\mathcal{P}_{(a)}}$, where*

$\mathcal{C}_{(a)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_2, p_2]_2, \ldots, [s_S, s'_{S'_{(a)}}, p_{\overline{P_{(a)}}}]_{C_{(a)}}\}$, and $C_{(a)} = |\mathcal{C}_{(a)}|$ is the number of tensor components of action (a). The number of tensor components $|\mathcal{C}_{(a)}|$ is a result of the number of states $|\mathcal{S}|$ multiplied by the number of state transition probabilities greater than zero $|\overline{\mathcal{P}_{(a)}}|$, as we show in Equation 3.4

$$|\mathcal{C}_{(a)}| = |\mathcal{S}| \times |\overline{\mathcal{P}_{(a)}}|. \tag{3.4}$$

Figure 3.3 illustrates the definition of tensor components of action *a*, by representing the tensor $\mathcal{C}_{(a)}$ as a sum of tensor components, as Equation 3.5 shows.

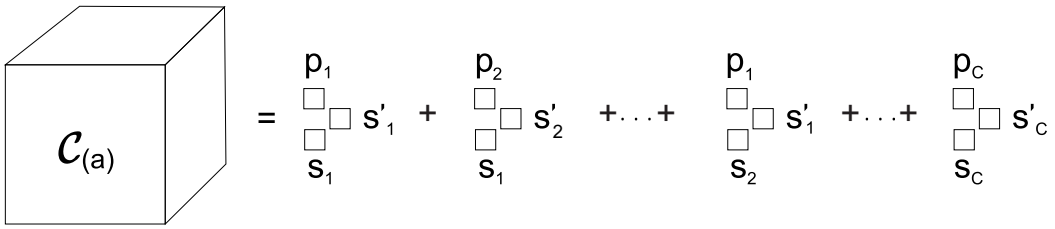


Figure 3.3: Third-order tensor components representation.

$$\mathcal{C}_{(a)} \approx \sum_{c=1}^{C} (s_c \otimes s'_c \otimes p_c) \tag{3.5}$$

Where *C* is a positive integer for c = 1, ..., *C*. Considering each dimension contains its own index (*i*, *j*, and *k*), we write Equation 3.5 as:

$$\mathcal{C}_{ijk} \approx \sum_{c=1}^{C} (s_{ic}, s'_{jc}, p_{kc}) \text{ for } i = 1, \ldots, |\mathcal{S}|;\ j = 1, \ldots, |\mathcal{S}'|;\ k = 1, \ldots, |\overline{\mathcal{P}_{(a)}}|.$$

For example, considering a 12-state and 2-dimensional MDP problem, we show a set of tensor components for each action, as follows:

$\mathcal{C}_{(N)} = \{[1, 1', 0.8]_1, [1, 1', 0.1]_2, \ldots, [12, 12', 0.1]_{36}\}$, *and* $|\mathcal{C}_{(N)}|$ = *36 tensor components*;
$\mathcal{C}_{(S)} = \{[1, 5', 0.8]_1, [1, 1', 0.1]_2, \ldots, [12, 12', 0.1]_{36}\}$, *and* $|\mathcal{C}_{(S)}|$ = *36 tensor components*;
$\mathcal{C}_{(W)} = \{[1, 1', 0.1]_1, [1, 5', 0.1]_1, \ldots, [12, 11', 0.8]_{36}\}$, *and* $|\mathcal{C}_{(W)}|$ = *36 tensor components*;
$\mathcal{C}_{(E)} = \{[1, 1', 0.1]_1, [1, 5', 0.1]_1, \ldots, [12, 12', 0.8]_{36}\}$, *and* $|\mathcal{C}_{(E)}|$ = *36 tensor components*.

In Table 3.1, we show all tensor components of the four actions, where Table 3.1a represents the tensor $\mathcal{C}_{(N)}$, Table 3.1b the tensor $\mathcal{C}_{(S)}$, Table 3.1c tensor $\mathcal{C}_{(W)}$. Table 3.1d represents the tensor $\mathcal{C}_{(E)}$.

Definition 13 (Tensor components of state s $\mathcal{C}^{s}_{(a)}$) *Let $\mathcal{C}^{s}_{(a)}$ be the set of tensor components given an action (a) for each state (s), where $\mathcal{C}^{s_S}_{(a_A)} = \{[s_S, s'_1, p_1]_1, [s_S, s'_2, p_2]_2, \dots, [s_S, s'_{S'^{S}_{(a)}}, p_{\overline{P_{(a)}}}]_{C^{s}_{(a)}}\}$, and $C^{s}_{(a)} = |\mathcal{C}^{s}_{(a)}|$ is the number of tensor components for each state (s) and for each action (a). We define $|\mathcal{C}^{s}_{(a)}|$ as the same number of Definition 9 as $|\mathcal{C}^{s}_{(a)}| = |\overline{\mathcal{P}_{(a)}}|$. Therefore, $\mathcal{C}^{s}_{(a)} \subset \mathcal{C}_{(a)}$.*

For example, for the 24-state of $|\overline{\mathcal{P}^{(a)}}| = 3$, we consider three tensor components for each state, as we show in Table 3.1.

By following the *tensor indexing* definition (see Section 2.3), we create sub-arrays by fixating indexes to be able to operate with each dimension of the tensor. We compress each dimension of the tensor $\mathcal{C}^{s}_{(a)}$ as (i) the first dimension being the states s, (ii) the second one the successor states s', and (iii) the third the state transition probabilities p. We illustrate each tensor indexing as follows:

- (i) Compressed set s: $\mathcal{C}^{s_1}_{(a_1)}[s] = \{s_1, s_1, \dots, s_1\} \dots \mathcal{C}^{s_S}_{(a_A)}[s] = \{s_S, s_S, \dots, s_S\}$.
- (ii) Compressed set s': $\mathcal{C}^{s_1}_{(a_1)}[s'] = \{s'_1, s'_2, \dots, s'_{S'^{1}_{(a)}}\} \dots \mathcal{C}^{s_S}_{(a_A)}[s'] = \{s'_1, s'_2, \dots, s'_{S'^{S}_{(a)}}\}$.
- (iii) Compressed set p: $\mathcal{C}^{s_1}_{(a_1)}[p] = \{p_1, p_2, \dots, p_{\overline{P_{(a)}}}\} \dots \mathcal{C}^{s_S}_{(a_A)}[p] = \{p_1, p_2, \dots, p_{\overline{P_{(a)}}}\}$.

For example, to illustrate the compressed tensor indexes, we show each compressed set as follows:

(i) States s:	(ii) Successor states s':	(iii) Probabilities p:
$\mathcal{C}^{s_1}_{(N)}[s] = \{1, 1, 1\}$;	$\mathcal{C}^{s_1}_{(N)}[s'] = \{1', 1', 2'\}$;	$\mathcal{C}^{s_1}_{(N)}[p] = \{0.8, 0.1, 0.1\}$;
...	...	...
$\mathcal{C}^{s_{12}}_{(N)}[s] = \{12, 12, 12\}$;	$\mathcal{C}^{s_{12}}_{(N)}[s'] = \{8', 11', 12'\}$;	$\mathcal{C}^{s_{12}}_{(N)}[p] = \{0.8, 0.1, 0.1\}$;
$\mathcal{C}^{s_1}_{(S)}[s] = \{1, 1, 1\}$;	$\mathcal{C}^{s_1}_{(S)}[s'] = \{5', 1', 2'\}$;	$\mathcal{C}^{s_1}_{(S)}[p] = \{0.8, 0.1, 0.1\}$;
...	...	...
$\mathcal{C}^{s_{12}}_{(S)}[s] = \{12, 12, 12\}$;	$\mathcal{C}^{s_{12}}_{(S)}[s'] = \{12', 11', 12'\}$;	$\mathcal{C}^{s_{12}}_{(S)}[p] = \{0.8, 0.1, 0.1\}$;
$\mathcal{C}^{s_1}_{(W)}[s] = \{1, 1, 1\}$;	$\mathcal{C}^{s_1}_{(W)}[s'] = \{1', 5', 1'\}$;	$\mathcal{C}^{s_1}_{(W)}[p] = \{0.1, 0.1, 0.8\}$;
...	...	...
$\mathcal{C}^{s_{12}}_{(W)}[s] = \{12, 12, 12\}$;	$\mathcal{C}^{s_{12}}_{(W)}[s'] = \{8', 12', 11'\}$;	$\mathcal{C}^{s_{12}}_{(W)}[p] = \{0.1, 0.1, 0.8\}$;
$\mathcal{C}^{s_1}_{(E)}[s] = \{1, 1, 1\}$;	$\mathcal{C}^{s_1}_{(E)}[s'] = \{1', 5', 2'\}$;	$\mathcal{C}^{s_1}_{(E)}[p] = \{0.1, 0.1, 0.8\}$;
...	...	...
$\mathcal{C}^{s_{12}}_{(E)}[s] = \{12, 12, 12\}$.	$\mathcal{C}^{s_{12}}_{(E)}[s'] = \{8', 12', 12'\}$.	$\mathcal{C}^{s_{12}}_{(E)}[p] = \{0.1, 0.1, 0.8\}$.

Using the CP-MDP compact representation of the state transition matrices, we represent large arrays with fewer elements than the total size. CP-MDP creates small tensor components instead of large tabular representations to solve an MDP, reducing the computational cost to generate monolithic MDPs.

Table 3.1: Tensor components of actions *North*, *South*, *West*, and *East* for each state s.

	s_1	s_2	s_3	s_4
s_1	$[1, 1', 0.8]_1$ $[1, 1', 0.1]_2$ $[1, 2', 0.1]_3$	$[2, 2', 0.8]_4$ $[2, 1', 0.1]_5$ $[2, 3', 0.1]_6$	$[3, 3', 0.8]_7$ $[3, 2', 0.1]_8$ $[3, 4', 0.1]_9$	$[4, 4', 0.8]_{10}$ $[4, 4', 0.1]_{11}$ $[4, 4', 0.1]_{12}$
s_2	$[5, 1', 0.8]_{13}$ $[5, 5', 0.1]_{14}$ $[5, 5', 0.1]_{15}$	$[6, 6', 0.8]_{16}$ $[6, 6', 0.1]_{17}$ $[6, 6', 0.1]_{18}$	$[7, 3', 0.8]_{19}$ $[7, 7', 0.1]_{20}$ $[7, 8', 0.1]_{21}$	$[8, 8', 0.8]_{22}$ $[8, 8', 0.1]_{23}$ $[8, 8', 0.1]_{24}$
s_3	$[9, 5', 0.8]_{25}$ $[9, 9', 0.1]_{26}$ $[9, 10', 0.1]_{27}$	$[10, 10', 0.8]_{28}$ $[10, 9', 0.1]_{29}$ $[10, 11', 0.1]_{30}$	$[11, 7', 0.8]_{31}$ $[11, 10', 0.1]_{32}$ $[11, 12', 0.1]_{33}$	$[12, 8', 0.8]_{34}$ $[12, 11', 0.1]_{35}$ $[12, 12', 0.1]_{36}$

(a) North.

	s_1	s_2	s_3	s_4
s_1	$[1, 5', 0.8]_1$ $[1, 1', 0.1]_2$ $[1, 2', 0.1]_3$	$[2, 2', 0.8]_4$ $[2, 1', 0.1]_5$ $[2, 3', 0.1]_6$	$[3, 7', 0.8]_7$ $[3, 2', 0.1]_8$ $[3, 4', 0.1]_9$	$[4, 4', 0.8]_{10}$ $[4, 4', 0.1]_{11}$ $[4, 4', 0.1]_{12}$
s_2	$[5, 9', 0.8]_{13}$ $[5, 5', 0.1]_{14}$ $[5, 5', 0.1]_{15}$	$[6, 6', 0.8]_{16}$ $[6, 6', 0.1]_{17}$ $[6, 6', 0.1]_{18}$	$[7, 11', 0.8]_{19}$ $[7, 7', 0.1]_{20}$ $[7, 8', 0.1]_{21}$	$[8, 8', 0.8]_{22}$ $[8, 8', 0.1]_{23}$ $[8, 8', 0.1]_{24}$
s_3	$[9, 9', 0.8]_{25}$ $[9, 9', 0.1]_{26}$ $[9, 10', 0.1]_{27}$	$[10, 10', 0.8]_{28}$ $[10, 9', 0.1]_{29}$ $[10, 11', 0.1]_{30}$	$[11, 11', 0.8]_{31}$ $[11, 10', 0.1]_{32}$ $[11, 12', 0.1]_{33}$	$[12, 12', 0.8]_{34}$ $[12, 11', 0.1]_{35}$ $[12, 12', 0.1]_{36}$

(b) South.

	s_1	s_2	s_3	s_4
s_1	$[1, 1', 0.1]_1$ $[1, 5', 0.1]_2$ $[1, 1', 0.8]_3$	$[2, 2', 0.1]_4$ $[2, 2', 0.1]_5$ $[2, 1', 0.8]_6$	$[3, 3', 0.1]_7$ $[3, 7', 0.1]_8$ $[3, 2', 0.8]_9$	$[4, 4', 0.1]_{10}$ $[4, 4', 0.1]_{11}$ $[4, 4', 0.8]_{12}$
s_2	$[5, 1', 0.1]_{13}$ $[5, 9', 0.1]_{14}$ $[5, 5', 0.8]_{15}$	$[6, 6', 0.1]_{16}$ $[6, 6', 0.1]_{17}$ $[6, 6', 0.8]_{18}$	$[7, 3', 0.1]_{19}$ $[7, 11', 0.1]_{20}$ $[7, 7', 0.8]_{21}$	$[8, 8', 0.1]_{22}$ $[8, 8', 0.1]_{23}$ $[8, 8', 0.8]_{24}$
s_3	$[9, 5', 0.1]_{25}$ $[9, 9', 0.1]_{26}$ $[9, 9', 0.8]_{27}$	$[10, 10', 0.1]_{28}$ $[10, 10', 0.1]_{29}$ $[10, 9', 0.8]_{30}$	$[11, 7', 0.1]_{31}$ $[11, 11', 0.1]_{32}$ $[11, 10', 0.8]_{33}$	$[12, 8', 0.1]_{34}$ $[12, 12', 0.1]_{35}$ $[12, 11', 0.8]_{36}$

(c) West.

	s_1	s_2	s_3	s_4
s_1	$[1, 1', 0.1]_1$ $[1, 5', 0.1]_2$ $[1, 2', 0.8]_3$	$[2, 2', 0.1]_4$ $[2, 2', 0.1]_5$ $[2, 3', 0.8]_6$	$[3, 3', 0.1]_7$ $[3, 7', 0.1]_8$ $[3, 4', 0.8]_9$	$[4, 4', 0.1]_{10}$ $[4, 4', 0.1]_{11}$ $[4, 4', 0.8]_{12}$
s_2	$[5, 1', 0.1]_{13}$ $[5, 9', 0.1]_{14}$ $[5, 5', 0.8]_{15}$	$[6, 6', 0.1]_{16}$ $[6, 6', 0.1]_{17}$ $[6, 6', 0.8]_{18}$	$[7, 3', 0.1]_{19}$ $[7, 11', 0.1]_{20}$ $[7, 8', 0.8]_{21}$	$[8, 8', 0.1]_{22}$ $[8, 8', 0.1]_{23}$ $[8, 8', 0.8]_{24}$
s_3	$[9, 5', 0.1]_{25}$ $[9, 9', 0.1]_{26}$ $[9, 10', 0.8]_{27}$	$[10, 10', 0.1]_{28}$ $[10, 10', 0.1]_{29}$ $[10, 11', 0.8]_{30}$	$[11, 7', 0.1]_{31}$ $[11, 11', 0.1]_{32}$ $[11, 12', 0.8]_{33}$	$[12, 8', 0.1]_{34}$ $[12, 12', 0.1]_{35}$ $[12, 12', 0.8]_{36}$

(d) East

3.3 CP-MDP Algorithms

This section develops an efficient algorithm to compute the transition model matrix (see Definition 7) in Section 3.3.1. In Section 3.3.2, we develop the algorithm to compute tensor components (see Definition 12). Then, in Sections 3.3.3 and 3.3.4, we develop the compact value iteration and policy iteration and show how they compute the solution using the tensor components.

3.3.1 Transition Model Matrix Generator

Concerning Definition 7, which is the state transition probabilities $\mathcal{P}$, we create an algorithm to compute the environment's transition models. Algorithm 3.1 consists of composing a matrix of size $|\mathcal{A} \times \mathcal{A}|$ with probabilities of state transition. We consider three main probabilities:

- *probability_intended_angle*: probability of transitioning towards the intended state;
- *probability_opposite_angle*: probability of state transition to the opposite state; and
- *probability_right_angle*: probability of going to a 90° angle of the intended state.

Algorithm 3.1: Transition model matrix generator

inputs: actions $a \in \mathcal{A}$.

local variables: TM: a matrix $|\mathcal{A} \times \mathcal{A}|$ to illustrate the transition model;
probability_intended_angle: the probability of transitioning to the intended state;
probability_opposite_angle: the probability of state transition to the opposite direction;
probability_right_angle: the probability of going to right angles of the intended state;
a_opposite: opposite action/direction.

```
1:  procedure GENERATETM(mdp) returns a Transition Model Matrix
2:      for each action a in 𝒜 do
3:          for each action a′ in 𝒜 do
4:              if a == a′ then
5:                  TM[a, a′] = probability_intended_angle
6:              else if a′ == a_opposite then
7:                  TM[a, a′] = probability_opposite_angle
8:              else
9:                  TM[a, a′] = probability_right_angle
10:     return TM
```

Table 3.2 shows the representation of the resulting matrix. For example, considering a 4-action problem, in Table 3.3, we illustrate the 16 probabilities in matrix form, which are the stochastic probabilities of taking action a in a given state s.

Table 3.2: Representation of a transition model matrix.

	a'_1	a'_2	a'_3	...	a'_A
a_1	p_{11}	p_{12}	p_{13}	...	p_{1A}
a_2	p_{21}	p_{22}	p_{23}	...	p_{2A}
a_3	p_{31}	p_{32}	p_{33}	...	p_{3A}
...	...	...	...	...	...
a_A	p_{A1}	p_{A2}	p_{A3}	...	p_{AA}

Table 3.3: An example of a transition model matrix of four actions.

	North	*South*	*West*	*East*
North	0.8	0.0	0.1	0.1
South	0.0	0.8	0.1	0.1
West	0.1	0.1	0.8	0.0
East	0.1	0.1	0.0	0.8

3.3.2 Tensor Components Generator

To compute all tensor components, we develop an algorithm to generate compact state transition matrices, as we modeled in our formalization (see Definition 12). Algorithm 3.2 consists of composing the set of components $\mathcal{C}_{(a)}$ at each iteration with a set of values $[s, s', p]$, where s is the current state, s' is the target state, and p is the probability of state transition.

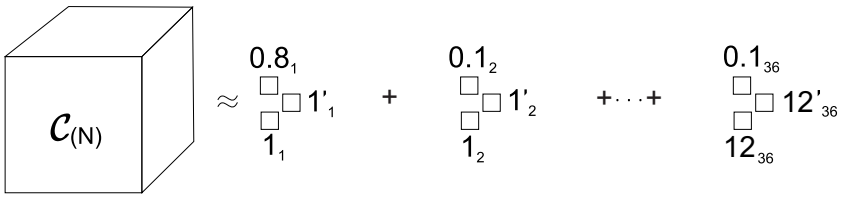
Iterating through each action $a \in \mathcal{A}$, each state $s \in \mathcal{S}$ and considering only probabilities where $\mathcal{P}(s'|s, a) > 0$, the successor state $s' \in \mathcal{S}'^{s}_{(a)}$ is computed by the function $succ_s$ (Algorithm 3.3). Then, we analyze if the resulting successor state $s' \notin \mathcal{O}$ or $s' \in \mathcal{O}$. If not, the set $\mathcal{C}_{(a)}$ receives the tuple with its respective state s, successor state s', and the probability of state transition p, or if so, the set $\mathcal{C}_{(a)}$ receives the successor state as being the same state s. The resulting set $\mathcal{C}_{(a)}$ is a tensor of components of action a, which enables a compact representation of the state transition matrices. Figure 3.4 illustrates the tensors of action *North*, *South*, *West*, and *East* to demonstrate the idea of tensor components that Algorithm 3.2 generates.

Algorithm 3.2: Tensor components generator

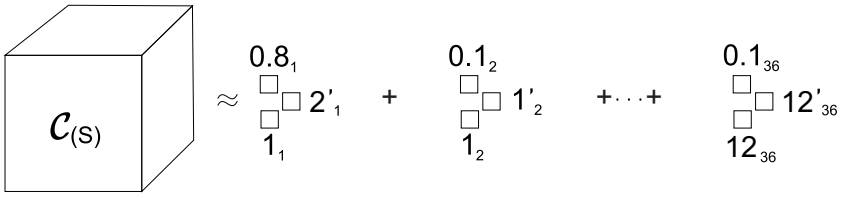
inputs: states $\mathcal{S}$, obstacles $\mathcal{O}$

local variables: $\mathcal{C}_{(a)}$ tensor containing tensor components of action *a*.

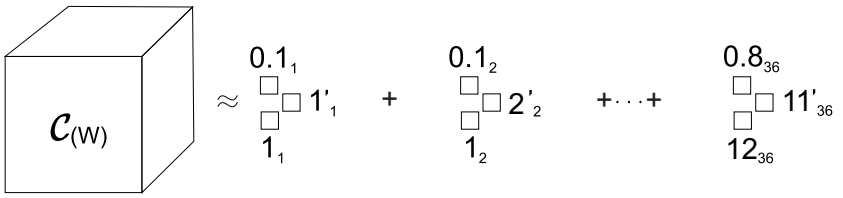
procedure GENERATETENSORCOMPONENTS(mdp) returns a tensor of components
  for each state *a* in $\mathcal{A}$ **do**
    for each state *s* in $\mathcal{S}$ **do**
      for each state *a'* in $\mathcal{A}$, where $\mathcal{P}(s'|s, a) > 0$ **do**
        $s' = succ_s$
        if s' $\notin \mathcal{O}$ **then**
          $\mathcal{C}_{(a)} \leftarrow \mathcal{C}_{(a)} \cup [s, s', p]$
        else
          $\mathcal{C}_{(a)} \leftarrow \mathcal{C}_{(a)} \cup [s, s, p]$
  return $\mathcal{C}_{(a)}$



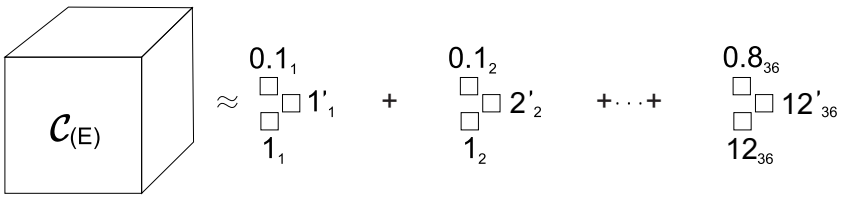
(a) North.



(b) South.



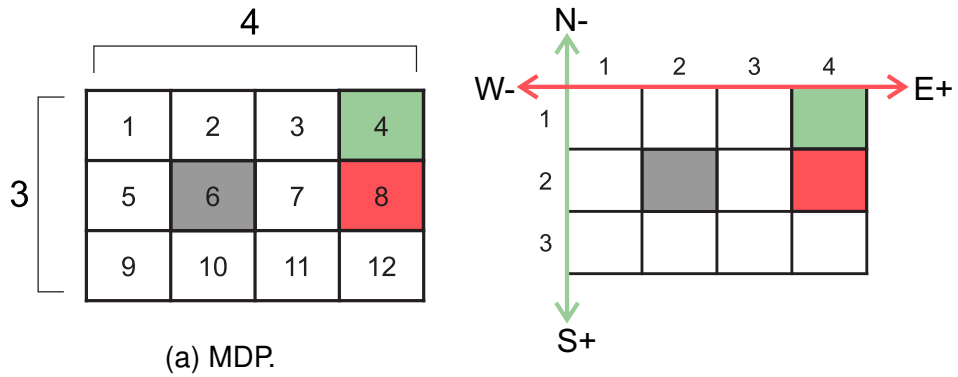
(c) West.



(d) East.

Figure 3.4: Illustration of tensor components of actions *North*, *South*, *West*, and *East*.

Algorithm 3.3 consists of iterating all dimensions of a given state s to generate its respective successor state s'. We are assuming a Euclidean space where actions are operations to increase or decrease each dimension's numeric value. In Figure 3.5b we illustrate a Cartesian plane idea to a 4×3 grid (Figure 3.5a), where actions *West* and *East* belong to the x-axis, i.e., to the first dimension, and actions *North* and *South* belong to the y-axis, i.e., to the second dimension. Actions *North* and *West* are negative directions because, in the Cartesian plane, we shift the state to smaller numbers. And *South* and *East* are positive directions because, in the Cartesian plane, we shift the state to larger numbers.



(a) MDP.

(b) Cartesian plane.

Figure 3.5: Example of a Cartesian plane for a two-dimensional 4×3 GRIDWORLD.

Algorithm 3.3: Successor state s' of state s generator

inputs: states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, initial limit $\alpha \in \mathcal{L}$ and final limit $\Omega \in \mathcal{L}$.

local variables: s': successor state of state s.

```
1:  procedure succ_s(mdp) returns a successor state of s
2:      for each dim in state s do
3:          if dim is equal to action axis then
4:              if action a is a negative direction then
5:                  if s[dim] ∉ α then
6:                      s' ← s[dim] − 1
7:                  else
8:                      s' ← s[dim]
9:              else
10:                 if s[dim] ∉ Ω then
11:                     s' ← s[dim] + 1
12:                 else
13:                     s' ← s[dim]
14:         else
15:             s' ← s[dim]
16:     return s'
```

3.3.3 CP-MDP-VI

The compact Value Iteration algorithm, which we call CP-MDP-VI, computes an optimal policy by iterating through each tensor component instead of the large state transition matrices in tabular value iteration algorithms. At each iteration, the algorithm chooses an optimal action (or maximum expected utility) based on the inner product between the compressed set of tensor components ($\mathcal{C}^{s}_{(a)}[p]$) and a vector of utilities ($\mathcal{V}[s']$) of the same size, when the solution converges, i.e., when the result yields no change in the utilities, the policy is extracted.

Algorithm 3.4: CP-MDP-VI, a compact value iteration algorithm for calculating state utilities.

inputs: an MDP with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$(s), rewards $r \in \mathcal{R}(s)$, and discount γ;
$\mathcal{C}^{s}_{(a)}$: tensor of components of state s;
ϵ: maximum error allowed in the utility of any state.

local variables:
$\mathcal{V}$ and $\mathcal{V}'$: vectors of utilities for states in $\mathcal{S}$, initially zero;
δ: the maximum change in the utility of any state in an iteration.

procedure VALUEITERATION(mdp, ϵ) returns a utility function
  repeat
    $\mathcal{V} \leftarrow \mathcal{V}'$; $\delta \leftarrow 0$
    for each state s in $\mathcal{S}$ **do**
      $\mathcal{V}'[s] \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} (\mathcal{C}^{s}_{(a)}[p] \cdot \mathcal{V}[s'])$
      if $|\mathcal{V}'[s] - \mathcal{V}[s]| > \delta$ **then** $\delta \leftarrow |\mathcal{V}'[s] - \mathcal{V}[s]|$
  until $\delta < \epsilon$ (1 - γ)/γ
return $\mathcal{V}$

For example, for a 4×3 GRIDWORLD problem, where $|\mathcal{A}| = 4$ (**N**orth, **S**outh, **W**est, and **E**ast), Figure 3.6a illustrates the vector of maximum expected utilities of each state. And in Figure 3.6b, we illustrate the optimal policy extracted from the utilities.

85.18	89.40	93.15	100.0
81.43	-3.0	68.35	-100.0
77.21	73.46	69.56	47.38

(a) Utilities.

E	E	E	Term. +100
N		N	Term. -100
N	W	W	W

(b) Policy.

Figure 3.6: Utilities computed by CP-MDP-VI. And policy extracted from the utilities.

3.3.4 CP-MDP-PI

Policy iteration (Howard, 1960) is an optimal policy construction algorithm that produces exact policies and value functions. Unlike tabular policy iteration algorithms, the CP-MDP-PI (Algorithm 3.5) computes a policy by a dot product between a set of probabilities ($\mathcal{C}^{s}_{(a)}[p]$) and a vector of utilities ($\mathcal{V}[s']$).

Algorithm 3.5: CP-MDP-PI, a compact policy iteration algorithm for calculating an optimal policy.

inputs: an MDP with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$(s);
$\mathcal{C}_{(a)}$: tensor of components;
$\mathcal{C}^{s}_{(a)}$: tensor of components of state s.

local variables:
$\mathcal{V}$: a vector of utilities for states in $\mathcal{S}$, initially zero;
π: a policy vector indexed by state, initially random.

1: **procedure** POLICY-ITERATION(*mdp*) returns a policy
2: **repeat**
3: $\mathcal{V} \leftarrow$ Policy-Evaluation(π, $\mathcal{V}$, *mdp*)
4: *unchanged?* $\leftarrow$ true
5: **for** each state s in $\mathcal{S}$ **do**
6: **if** $\max_{a \in \mathcal{A}(s)} (\mathcal{C}^{s}_{(a)}[p] \cdot \mathcal{V}[s']) > (\mathcal{C}^{s}_{(a)}[p] \cdot \mathcal{V}[s'])$ **then**
7: $\pi[s] \leftarrow \arg\max_{a \in \mathcal{A}(s)} (\mathcal{C}^{s}_{(a)}[p] \cdot \mathcal{V}[s'])$
8: *unchanged?* $\leftarrow$ false
9: **until** *unchanged?*
10: **return** π

The algorithm begins with an arbitrary policy and repeatedly alternates between an evaluation phase (Line 6) to evaluate the current policy and an improvement phase (Line 7) to improve the policy. This continues until no local policy improvement is possible. The optimal policy generated by CP-MDP-PI for a 4×3 grid is the same as CP-MDP-VI previously mentioned because the value functions of both CP-MDP algorithms are identical to each other.

3.4 Complexity and Computational Cost Analysis

This section analyzes computational cost (number of multiplications) and the complexity of the CP-MDP and tabular approaches. In Section 3.4.1, we show the computational cost to generate the transition state probabilities both for tabular and compact approaches. And in Section 3.4.2, we show the computational cost to compute the solution.

3.4.1 Precomputation

We begin by computing the computational cost to generate state transition matrices for tabular representations, taking into account that a transition model system for tabular approaches requires multiplications equal to $O(|\mathcal{A}| \times |\mathcal{S}|^2)$. The resulting computational cost is exponential in the number of states. In Equation 3.6, we illustrate the computational cost required to compute state transition matrices for tabular approaches.

$$\prod_{i=1}^{A} a_i \times \prod_{j=1}^{S} s_j \times \prod_{k=1}^{S} s_k \tag{3.6}$$

For example, considering a 4×3 GRIDWORLD problem, where $|\mathcal{S}| = 12$ and $|\mathcal{A}| = 4$, we show the required number of multiplications in the following, and in Table 3.4 we illustrate the state transition matrices of actions *North*, *South*, *West*, and *East*.

$$\prod_{i=1}^{4} a_i \times \prod_{j=1}^{12} s_j \times \prod_{k=1}^{12} s_k = 576 \text{ multiplications,}$$

With regard to compare a larger example, we now consider a 100×100 GRIDWORLD, where $|\mathcal{S}| = 10,000$ and $|\mathcal{A}| = 4$. Therefore:

$$\prod_{i=1}^{4} a_i \times \prod_{j=1}^{10,000} s_j \times \prod_{k=1}^{10,000} s_k = 400{,}000{,}000 \text{ multiplications.}$$

Now, we define the computational cost of CP-MDP to generate compact state transition matrices (Algorithm 3.2), which we call *tensor components*, by Equation 3.7, and its respective complexity of $O(|\mathcal{A}| \times |\mathcal{C}_{(a)}| \times 3)$, where 3 is a constant to represent the third-order tensor $\mathcal{C}_{(a)}$.

$$\prod_{i=1}^{A} a_i \times \prod_{j=1}^{C_{(a_i)}} c_j \times \prod_{k=1}^{3} k. \tag{3.7}$$

In order to illustrate an example, we consider a two-dimensional GRIDWORLD example of size 4×3, where the computational cost is:

$$\prod_{i=1}^{4} a_i \times \prod_{j=1}^{36} c_j \times \prod_{k=1}^{3} k = 432 \text{ multiplications.}$$

To show a larger example, now we consider a 100×100 GRIDWORLD, where $|\mathcal{S}| = 10,000$ and $|\mathcal{A}| = 4$. Therefore:

$$\prod_{i=1}^{4} a_i \times \prod_{j=1}^{30,000} s_j \times \prod_{k=1}^{3} k = 360{,}000 \text{ multiplications.}$$

Table 3.4: State transition matrices of actions *North*, *South*, *West*, and *East* for the 4×3 grid example.

0.9	0.1	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.8	0.1	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.1	0.8	0.1	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.8	0.	0.	0.	0.2	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.8	0.	0.	0.	0.1	0.1	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.8	0.	0.	0.	0.1	0.1	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.1	0.8	0.1	0.
0.	0.	0.	0.	0.	0.	0.8	0.	0.	0.1	0.	0.1
0.	0.	0.	0.	0.	0.	0.	0.8	0.	0.	0.1	0.1

(a) North

0.1	0.1	0.	0.	0.8	0.	0.	0.	0.	0.	0.	0.
0.1	0.8	0.1	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.8	0.1	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.2	0.	0.	0.	0.8	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.1	0.1	0.	0.	0.8	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.9	0.1	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.1	0.8	0.1	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.1	0.8	0.1
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.1	0.9

(b) South

0.9	0.	0.	0.	0.1	0.	0.	0.	0.	0.	0.	0.
0.8	0.2	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.8	0.1	0.	0.	0.	0.1	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.	0.	0.	0.8	0.	0.	0.	0.1	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.1	0.	0.	0.	0.8	0.	0.	0.	0.1	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.1	0.	0.	0.	0.9	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.8	0.2	0.	0.
0.	0.	0.	0.	0.	0.	0.1	0.	0.	0.8	0.1	0.
0.	0.	0.	0.	0.	0.	0.	0.1	0.	0.	0.8	0.1

(c) West

0.1	0.8	0.	0.	0.1	0.	0.	0.	0.	0.	0.	0.
0.	0.2	0.8	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.1	0.8	0.	0.	0.1	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.1	0.	0.	0.	0.8	0.	0.	0.	0.1	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.1	0.	0.	0.	0.	0.8	0.	0.	0.1	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.1	0.	0.	0.	0.1	0.8	0.	0.
0.	0.	0.	0.	0.	0.	0.	0.	0.	0.2	0.8	0.
0.	0.	0.	0.	0.	0.	0.1	0.	0.	0.	0.1	0.8
0.	0.	0.	0.	0.	0.	0.	0.1	0.	0.	0.	0.9

(d) East

To compare Equation 3.6, which generates tabular state transition matrices, against Equation 3.7, which computes tensor components for the CP-MDP method, in Table 3.5 we show a comparison of 2 and 3-dimensional grids between the tabular approach and our compact model.

Table 3.5: Computational cost comparison to generate state transition matrices (tabular) and tensor components (CP-MDP).

Grid size	Tabular	CP-MDP	Complexity reduction
4×3	576	432	25.00%
50×50	25,000,000	90,000	99.64%
100×100	400,000,000	360,000	99.91%
1000×1000	4,000,000,000,000	36,000,000	99.99%
$4 \times 3 \times 2$	3,456	2,160	37.50%
$50 \times 50 \times 5$	937,500,000	1,125,000	99.88%
$100 \times 100 \times 10$	60,000,000,000	9,000,000	99.99%
$1000 \times 1000 \times 100$	6E+16	9,000,000,000	99.99%

3.4.2 Computation

In this section, we analyze the computational cost to compute an MDP solution using (i) tabular value iteration and (ii) tabular policy iteration algorithms (Algorithm 2.1 and 2.2) and using (iii) CP-MDP-VI and (iv) CP-MDP-PI compact algorithms (Algorithms 3.4 and 3.5).

(i) We begin analyzing the value iteration algorithm in tabular form, which we call TABULAR-VI. Each iteration of value iteration requires $O(|\mathcal{A}| \times |\mathcal{S}|^2)$ computation time, and the number of iterations is polynomial in $|\mathcal{S}|$ (Littman et al., 1995). Equation 3.8 shows the resulting computational cost of TABULAR-VI.

$$\prod_{i=1}^{A} a_i \times \prod_{j=1}^{S} s_j \times \prod_{k=1}^{S} s_k. \tag{3.8}$$

For example, considering the same 4×3 GRIDWORLD problem previously addressed, where $|\mathcal{S}| = 12$ and $|\mathcal{A}| = 4$, the required number of multiplications for TABULAR-VI is equal to:

$$\prod_{i=1}^{4} a_i \times \prod_{j=1}^{12} s_j \times \prod_{k=1}^{12} s_k = 576 \text{ multiplications.}$$

(ii) And the tabular policy iteration, which we call TABULAR-PI, requires roughly $O(|\mathcal{S}|^3)$ computations for each evaluation step, and requires $O(|\mathcal{A}| \times |\mathcal{S}|^2)$ for each improvement step (Littman et al., 1995), resulting in a complexity of $O(|\mathcal{S}|^3 + |\mathcal{A}| \times |\mathcal{S}|^2)$. Therefore, Equation 3.9 shows the computational cost of TABULAR-PI.

$$\prod_{i=1}^{S} s_i \times \prod_{j=1}^{S} s_j \times \prod_{k=1}^{S} s_k + \prod_{l=1}^{A} a_l \times \prod_{m=1}^{S} s_m \times \prod_{n=1}^{S} s_n. \tag{3.9}$$

Policy Iteration converges quadratically and in practice tends to do so in relatively few iterations compared to value iteration (Puterman, 1994). For example, TABULAR-PI computational cost for an example with $|\mathcal{S}| = 12$ and $|\mathcal{A}| = 4$ is:

$$\prod_{i=1}^{12} s_i \times \prod_{j=1}^{12} s_j \times \prod_{k=1}^{12} s_k + \prod_{l=1}^{4} a_l \times \prod_{m=1}^{12} s_m \times \prod_{n=1}^{12} s_n = 2{,}304 \text{ multiplications.}$$

(iii) Now, we address the CP-MDP-VI computational cost, which requires a number of multiplications equal to:

$$\prod_{i=1}^{S} s_i \times \prod_{j=1}^{C^s_{(a)}} c_j \times \prod_{k=1}^{A} a_k, \tag{3.10}$$

we arrive at the complexity of $O(|\mathcal{S}| \times |\mathcal{C}^s_{(a)}| \times |\mathcal{A}|)$, where $|\mathcal{C}^s_{(a)}|$ is the number of tensor components for each state s.

For example, for an MDP problem where $|\mathcal{S}| = 12$, $|\mathcal{C}^s_{(a)}| = 3$, and $|\mathcal{A}| = 4$, we show the required number of multiplications to solve this problem, as follows:

$$\prod_{i=1}^{12} a_i \times \prod_{j=1}^{3} c_j \times \prod_{k=1}^{4} a_k = 144 \text{ multiplications.}$$

(iv) By contrast, CP-MDP-PI requires a number of multiplications equal to Equation 3.11, which results in a $O(|\mathcal{S}| \times |\mathcal{C}^s_{(a)}|^2 + |\mathcal{S}| \times |\mathcal{C}^s_{(a)}| \times |\mathcal{A}|)$ complexity.

$$\prod_{i=1}^{S} s_i \times \prod_{j=1}^{C^s_{(a)}} c_j \times \prod_{k=1}^{C^s_{(a)}} c_k + \prod_{l=1}^{S} s_l \times \prod_{m=1}^{C^s_{(a)}} c_m \times \prod_{n=1}^{A} a_n. \tag{3.11}$$

To illustrate the computational cost of CP-MDP-PI, we compute the same example ($|\mathcal{S} = 12|$, $|\mathcal{C}^s_{(a)}| = 3$ and $|\mathcal{A}| = 4$) using this algorithm.

$$\prod_{i=1}^{12} s_i \times \prod_{j=1}^{3} c_j \times \prod_{k=1}^{3} c_k + \prod_{l=1}^{12} s_l \times \prod_{m=1}^{3} c_m \times \prod_{n=1}^{4} a_n = 252 \text{ multiplications.}$$

To compare Equation 3.8 that computes the solution using tabular value iteration (TABULAR-VI), with Equation 3.10, which uses the compact value iteration (CP-MDP-VI), in Table 3.6 we show a computational cost comparison of 2 and 3-dimensional grids. And in Table 3.7, we show a computational cost comparison, using several grid sizes, between Equation 3.9 that solves problems using tabular policy iteration (TABULAR-PI), with Equation 3.11, which uses CP-MDP-PI, the compact policy iteration algorithm.

Table 3.6: Computational cost comparison between TABULAR-VI and CP-MDP-VI algorithms.

Grid size	TABULAR-VI	CP-MDP-VI	Complexity reduction
4×3	576	144	75.00%
50×50	25,000,000	30,000	99.88%
100×100	400,000,000	120,000	99.97%
1000×1000	4,000,000,000,000	12,000,000	99.99%
$4 \times 3 \times 2$	3,456	720	79.17%
$50 \times 50 \times 5$	937,500,000	375,000	99.96%
$100 \times 100 \times 10$	60,000,000,000	3,000,000	99.99%
$1000 \times 1000 \times 100$	6E+16	3,000,000,000	99.99%

Table 3.7: Computational cost comparison between TABULAR-PI and CP-MDP-PI algorithms.

Grid size	TABULAR-PI	CP-MDP-PI	Complexity reduction
4×3	2,304	252	89,06%
50×50	15,650,000,000	52,500	99.99%
100×100	1,000,400,000,000	210,000	99.99%
1000×1000	1E+18	21,000,000	99.99%
$4 \times 3 \times 2$	17,280	1,320	92.36%
$50 \times 50 \times 5$	1,954,062,500,000	687,500	99.99%
$100 \times 100 \times 10$	1,00006E+15	5,500,000	99.99%
$1000 \times 1000 \times 100$	1E+24	5,500,000,000	99.99%

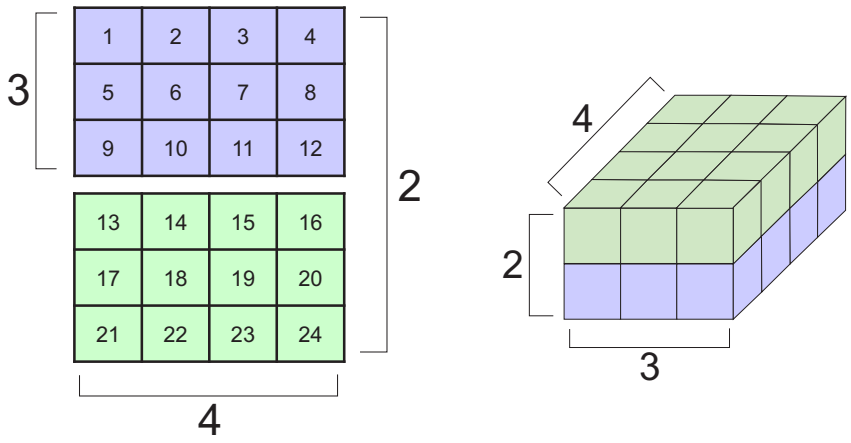
Consequently, CP-MDP-VI and CP-MDP-PI require much less computation and substantially less memory than traditional tabular methods. So now, in Table 3.8, we show a comparison between the compact algorithms: CP-MDP-VI and CP-MDP-PI.

Table 3.8: Computational cost comparison between CP-MDP-VI and CP-MDP-PI algorithms.

Grid size	CP-MDP-VI	CP-MDP-PI	Complexity reduction
4×3	576	252	42,86%
50×50	25,000,000	52,500	42,86%
100×100	400,000,000	210,000	42,86%
1000×1000	4,000,000,000,000	21,000,000	42,86%
$4 \times 3 \times 2$	3,456	1,320	45,45%
$50 \times 50 \times 5$	937,500,000	687,500	45,45%
$100 \times 100 \times 10$	60,000,000,000	5,500,000	45,45%
$1000 \times 1000 \times 100$	6E+16	5,500,000,000	45,45%

3.5 Three-dimensional GRIDWORLD Example

To show a more complex example, in this section, we illustrate a 3-dimensional GRIDWORLD MDP example using the tensor algebra formalization and the CP-MDP representation. In Figure 3.7 we show a three-dimensional MDP problem, where $\mathcal{D} = \{\mathbb{X}_1, \mathbb{Y}_2, \mathbb{Z}_3\}$ and $|\mathcal{D}| = 3$ (see Definition 1).



(a) MDP.

(b) Tensor.

Figure 3.7: A three-dimensional 4×3×2 GRIDWORLD example.

Each dimension comprises a set of states, and the resulting number of states is $|\mathcal{S}| = (4 \times 3 \times 2) = 24$ states (Definition 2), as follows:

$$\begin{aligned}\mathcal{S} = \{&(x_1, y_1, z_1)_{s_1}, (x_1, y_1, z_2)_{s_2}, (x_2, y_1, z_1)_{s_3}, (x_2, y_1, z_2)_{s_4},\\ &(x_3, y_1, z_1)_{s_5}, (x_3, y_1, z_2)_{s_6}, (x_4, y_1, z_1)_{s_7}, (x_4, y_1, z_2)_{s_8},\\ &(x_1, y_2, z_1)_{s_9}, (x_1, y_2, z_2)_{s_{10}}, (x_2, y_2, z_1)_{s_{11}}, (x_2, y_2, z_2)_{s_{12}},\\ &(x_3, y_2, z_1)_{s_{13}}, (x_3, y_2, z_2)_{s_{14}}, (x_4, y_2, z_1)_{s_{15}}, (x_4, y_2, z_2)_{s_{16}},\\ &(x_1, y_3, z_1)_{s_{17}}, (x_1, y_3, z_2)_{s_{18}}, (x_2, y_3, z_1)_{s_{19}}, (x_2, y_3, z_2)_{s_{20}},\\ &(x_3, y_3, z_1)_{s_{21}}, (x_3, y_3, z_2)_{s_{22}}, (x_4, y_3, z_1)_{s_{23}}, (x_4, y_3, z_2)_{s_{24}}\}.\end{aligned}$$

For the three-dimensional problem, the resulting number of actions is $\mathcal{A} = (2 \times 3) = 6$ (Definition 3). We name these six actions as *West* and *East* for the x-axis, *North* and *South* for the y-axis, and *Forward* and *Backward* for the z-axis.

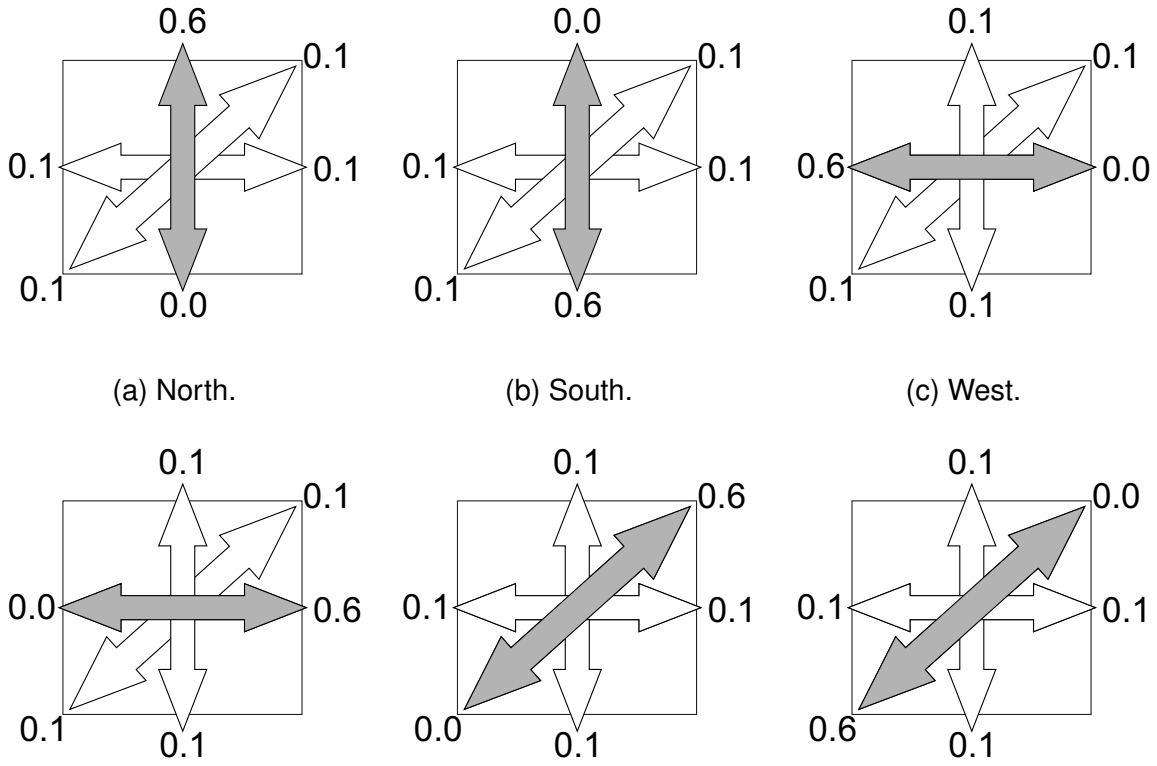
We now define an obstacle set (Definition 4), a terminal set (Definition 5), and a reward set (Definition 6). For example, $\mathcal{O} = \{4, 7, 20\}$, where $|\mathcal{O}| = 3$, and $\mathcal{T} = \{5, 15, 17\}$, where $|\mathcal{T}| = 3$. Taking into account a $|\mathcal{R}| = 24$, we determine a reward of -4 for non-terminal states, and 100 or -100 for terminal states:

$$\begin{aligned}\mathcal{R} = \{&-4, -4, -4, -4, 100, -4, -4, -4, -4, -4, -4, -4,\\ &-4, -100, -4, 100, -4, -4, -4, -4, -4, -4, -4, -4\}.\end{aligned}$$

Concerning the Definition 7, we consider a 0.6 probability of ending in the intended state, 0.1 of going to the right angles (90° angles) of the intended state, and 0.0 of going to the opposite state, as we show in Figure 3.8. The resulting number of probabilities is given

by $|\mathcal{P}| = (6 \times 6) = 36$, as follows:

$$\mathcal{P} = \{0.6, 0.0, 0.1, 0.1, 0.1, 0.1, 0.0, 0.6, 0.1, 0.1, 0.1, 0.1,$$
$$0.1, 0.1, 0.6, 0.0, 0.1, 0.1, 0.1, 0.1, 0.0, 0.6, 0.1, 0.1,$$
$$0.1, 0.1, 0.1, 0.1, 0.6, 0.0, 0.1, 0.1, 0.1, 0.1, 0.0, 0.6\}.$$



(a) North. (b) South. (c) West.

(d) East. (e) Forward. (f) Backward.

Figure 3.8: Transition models of actions *North*, *South*, *West*, *East*, *Forward*, and *Backward* illustration.

Table 3.9 illustrates the 36 probabilities in matrix form: the stochastic probability of taking action *a* in a given state *s*.

Table 3.9: Example of a transition model matrix of six actions.

$\mathcal{P}$ =		*North*	*South*	*West*	*East*	*Forward*	*Backward*
	North	0.6	0.0	0.1	0.1	0.1	0.1
	South	0.0	0.6	0.1	0.1	0.1	0.1
	West	0.1	0.1	0.6	0.0	0.1	0.1
	East	0.1	0.1	0.0	0.6	0.1	0.1
	Forward	0.1	0.1	0.1	0.1	0.6	0.0
	Backward	0.1	0.1	0.1	0.1	0.0	0.6

The environment limits of this example are $\mathcal{L} = \{[1,3]_1, [1,4]_2, [1,2]_3\}$, and $|\mathcal{L}| = 3$, where each component is a tuple of initial and final limits (Definition 8). The initial limit and final limit of dimension $\mathbb{X}$ are $\alpha(1)$ and $\Omega(3)$, respectively, of dimension $\mathbb{Y}$ are $\alpha(1)$ and $\Omega(4)$, and of dimension $\mathbb{Z}$ are $\alpha(1)$ and $\Omega(2)$.

Considering only probabilities of state transition where $\mathcal{P}(s'|s,a) > 0$ (Definition 9), $|\overline{\mathcal{P}_{(a)}}| = 5$ for each action, as follows:

$$\overline{\mathcal{P}_{(N)}} = \{0.6, 0.1, 0.1, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(S)}} = \{0.6, 0.1, 0.1, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(W)}} = \{0.1, 0.1, 0.6, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(E)}} = \{0.1, 0.1, 0.6, 0.1, 0.1\};$$
$$\overline{\mathcal{P}_{(F)}} = \{0.1, 0.1, 0.1, 0.1, 0.6\};$$
$$\overline{\mathcal{P}_{(B)}} = \{0.1, 0.1, 0.1, 0.1, 0.6\}.$$

As $|\overline{\mathcal{P}_{(a)}}| = 5$ and $\mathcal{S} = 24$, the following are the successors states for each action a, where $|\mathcal{S}'_{(a)}| = 120$ (Definition 10).

$$\mathcal{S}'_{(N)} = \{1', 1', 5', 1', 2', 2', 2', 6', 1', 3', 3', 3', 3', 2', 3', \ldots, 24', 24', 23', 24', 13'\};$$
$$\mathcal{S}'_{(S)} = \{1', 5', 1', 2', 14', 2', 6', 1', 3', 15', 3', 3', 2', 3', 4', \ldots, 24', 24', 24', 23', 24'\};$$
$$\mathcal{S}'_{(W)} = \{1', 13', 1', 1', 2', 2', 14', 2', 1', 3', 3', 15', 3', 2', 3', \ldots, 12', 24', 24', 23', 24'\};$$
$$\mathcal{S}'_{(E)} = \{1', 13', 5', 1', 2', 2', 14', 6', 1', 3', 3', 15', 3', 2', 3', \ldots, 12', 24', 24', 23', 24'\};$$
$$\mathcal{S}'_{(F)} = \{1', 13', 1', 5', 1', 2', 14', 2', 6', 1', 3', 15', 3', 3', 2', \ldots, 12', 24', 24', 24', 23'\};$$
$$\mathcal{S}'_{(B)} = \{1', 13', 1', 5', 2', 2', 14', 2', 6', 3', 3', 15', 3', 3', 3', \ldots, 12', 24', 24', 24', 24'\}.$$

As $|\overline{\mathcal{P}_{(a)}}| = 5$, the following are the successors states for each state given an action a, where $|\mathcal{S}'^{s}_{(a)}| = 5$ (Definition 11).

$$\mathcal{S}'^{s_1}_{(N)} = \{1', 1', 5', 1', 2'\}; \quad \mathcal{S}'^{s_1}_{(W)} = \{1', 13', 1', 1', 2'\}; \quad \mathcal{S}'^{s_1}_{(F)} = \{1', 13', 1', 5', 1'\};$$
$$\mathcal{S}'^{s_2}_{(N)} = \{2', 2', 6', 1', 3'\}; \quad \mathcal{S}'^{s_2}_{(W)} = \{2', 14', 2', 1', 3'\}; \quad \mathcal{S}'^{s_2}_{(F)} = \{2', 14', 2', 6', 1'\};$$
$$\mathcal{S}'^{s_3}_{(N)} = \{3', 3', 3', 2', 3'\}; \quad \mathcal{S}'^{s_3}_{(W)} = \{3', 15', 3', 2', 3'\}; \quad \mathcal{S}'^{s_3}_{(F)} = \{3', 15', 3', 3', 2'\};$$
$$\ldots \qquad \ldots \qquad \ldots$$
$$\mathcal{S}'^{s_{24}}_{(N)} = \{24', 24', 23', 24', 13'\}; \quad \mathcal{S}'^{s_{24}}_{(W)} = \{12', 24', 24', 23', 24'\}; \quad \mathcal{S}'^{s_{24}}_{(F)} = \{12', 24', 24', 24', 23'\};$$
$$\mathcal{S}'^{s_1}_{(S)} = \{1', 5', 1', 2', 14'\}; \quad \mathcal{S}'^{s_1}_{(E)} = \{1', 13', 5', 1', 2'\}; \quad \mathcal{S}'^{s_1}_{(B)} = \{1', 13', 1', 5', 2'\};$$
$$\mathcal{S}'^{s_2}_{(S)} = \{2', 6', 1', 3', 15'\}; \quad \mathcal{S}'^{s_2}_{(E)} = \{2', 14', 6', 1', 3'\}; \quad \mathcal{S}'^{s_2}_{(B)} = \{2', 14', 2', 6', 3'\};$$
$$\mathcal{S}'^{s_3}_{(S)} = \{3', 3', 2', 3', 4'\}; \quad \mathcal{S}'^{s_3}_{(E)} = \{3', 15', 3', 2', 3'\}; \quad \mathcal{S}'^{s_3}_{(B)} = \{3', 15', 3', 3', 3'\};$$
$$\ldots \qquad \ldots \qquad \ldots$$
$$\mathcal{S}'^{s_{24}}_{(S)} = \{24', 24', 24', 23', 24'\}; \quad \mathcal{S}'^{s_{24}}_{(E)} = \{12', 24', 24', 23', 24'\}; \quad \mathcal{S}'^{s_{24}}_{(B)} = \{12', 24', 24', 24', 24'\}.$$

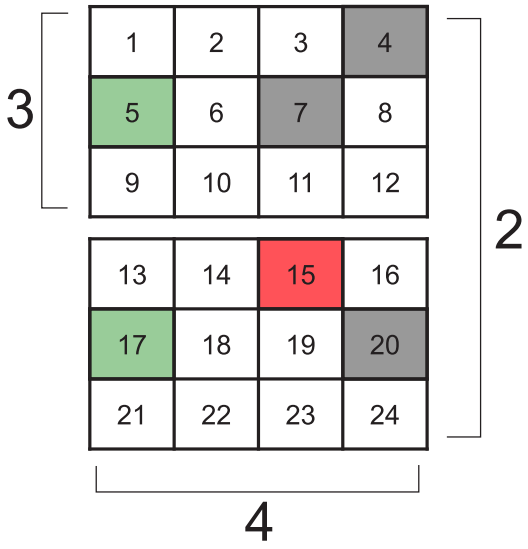
For the $\mathcal{C}_{(a)}$ tensor (Definition 12), we compute a set of tensor components for each action, as follows:

- $\mathcal{C}_{(N)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_1, p_2]_2, \ldots, [s_{24}, s'_{24}, p_5]_{120}\}$, and $|\mathcal{C}_{(N)}| = 120$ tensor components;
- $\mathcal{C}_{(S)} = \{[s_1, s'_5, p_1]_1, [s_1, s'_1, p_2]_2, \ldots, [s_{24}, s'_{24}, p_5]_{120}\}$, and $|\mathcal{C}_{(S)}| = 120$ tensor components;
- $\mathcal{C}_{(W)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_5, p_2]_1, \ldots, [s_{24}, s'_{24}, p_5]_{120}\}$, and $|\mathcal{C}_{(W)}| = 120$ tensor components;
- $\mathcal{C}_{(E)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_5, p_2]_1, \ldots, [s_{24}, s'_{24}, p_5]_{120}\}$, and $|\mathcal{C}_{(E)}| = 120$ tensor components;
- $\mathcal{C}_{(F)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_5, p_2]_1, \ldots, [s_{24}, s'_{23}, p_5]_{120}\}$, and $|\mathcal{C}_{(F)}| = 120$ tensor components;
- $\mathcal{C}_{(B)} = \{[s_1, s'_1, p_1]_1, [s_1, s'_5, p_2]_1, \ldots, [s_{24}, s'_{24}, p_5]_{120}\}$, and $|\mathcal{C}_{(B)}| = 120$ tensor components.

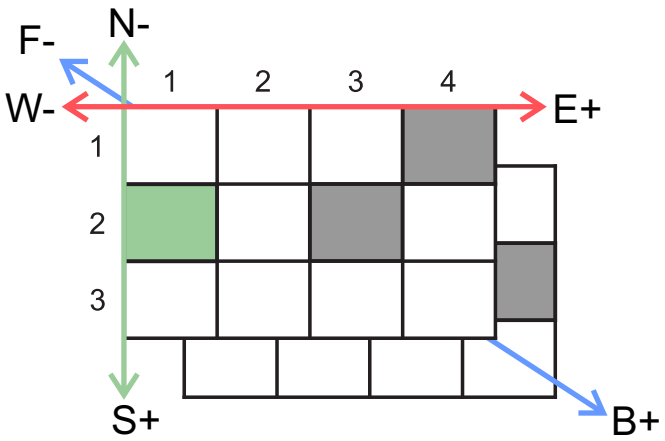
As previously mentioned, $|\overline{\mathcal{P}_{(a)}}| = 5$, so we consider five tensor components for each state and for each action (Definition 13), as follows:

- $\mathcal{C}^{s_1}_{(N)} = \{[1, 1, 0.6]_1, [1, 1, 0.1]_2, [1, 5, 0.1]_3, [1, 1, 0.1]_4, [1, 2, 0.1]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(N)} = \{[24, 12, 0.6]_1, [24, 20, 0.1]_2, [24, 24, 0.1]_3, [24, 23, 0.1]_4, [24, 24, 0.1]_5\}$
- $\mathcal{C}^{s_1}_{(S)} = \{[1, 13, 0.6]_1, [1, 1, 0.1]_2, [1, 5, 0.1]_3, [1, 1, 0.1]_4, [1, 2, 0.1]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(S)} = \{[24, 24, 0.6]_1, [24, 20, 0.1]_2, [24, 24, 0.1]_3, [24, 23, 0.1]_4, [24, 24, 0.1]_5\}$
- $\mathcal{C}^{s_1}_{(W)} = \{[1, 1, 0.1]_1, [1, 13, 0.1]_2, [1, 1, 0.6]_3, [1, 1, 0.1]_4, [1, 2, 0.1]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(W)} = \{[24, 12, 0.1]_1, [24, 24, 0.1]_2, [24, 20, 0.6]_3, [24, 23, 0.1]_4, [24, 24, 0.1]_5\}$
- $\mathcal{C}^{s_1}_{(E)} = \{[1, 1, 0.1]_1, [1, 13, 0.1]_2, [1, 5, 0.6]_3, [1, 1, 0.1]_4, [1, 2, 0.1]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(E)} = \{[24, 12, 0.1]_1, [24, 24, 0.1]_2, [24, 24, 0.6]_3, [24, 23, 0.1]_4, [24, 24, 0.1]_5\}$
- $\mathcal{C}^{s_1}_{(F)} = \{[1, 1, 0.1]_1, [1, 13, 0.1]_2, [1, 1, 0.1]_3, [1, 5, 0.1]_4, [1, 1, 0.6]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(F)} = \{[24, 12, 0.1]_1, [24, 24, 0.1]_2, [24, 20, 0.1]_3, [24, 24, 0.1]_4, [24, 23, 0.6]_5\}$
- $\mathcal{C}^{s_1}_{(B)} = \{[1, 1, 0.1]_1, [1, 13, 0.1]_2, [1, 1, 0.1]_3, [1, 5, 0.1]_4, [1, 2, 0.6]_5\}$
  ...
- $\mathcal{C}^{s_{24}}_{(B)} = \{[24, 12, 0.1]_1, [24, 24, 0.1]_2, [24, 20, 0.1]_3, [24, 24, 0.1]_4, [24, 24, 0.6]_5\}$.

In Figure 3.9b we illustrate the Cartesian plane idea to a $4 \times 3 \times 2$ grid (Figure 3.9a), where actions *West* and *East* belong to the x-axis (first dimension), actions *North* and *South* belong to the y-axis (second dimension), and actions *Forward* and *Backward* belong to the z-axis (third dimension). Actions *North*, *West*, and *Forward* are negative directions because, in the Cartesian plane, we shift the state to smaller numbers. And actions *South*, *East*, and *Backward* are positive directions because, in the Cartesian plane, we shift the state to larger numbers.



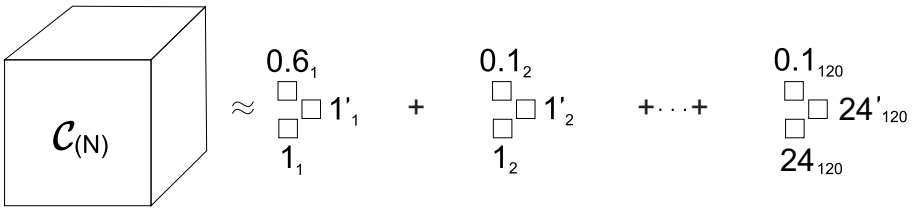
(a) MDP.



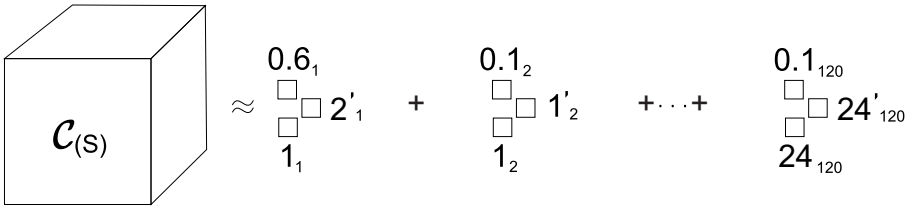
(b) Cartesian plane.

Figure 3.9: Example of a Cartesian plane for a three-dimensional $4 \times 3 \times 2$ GRIDWORLD.

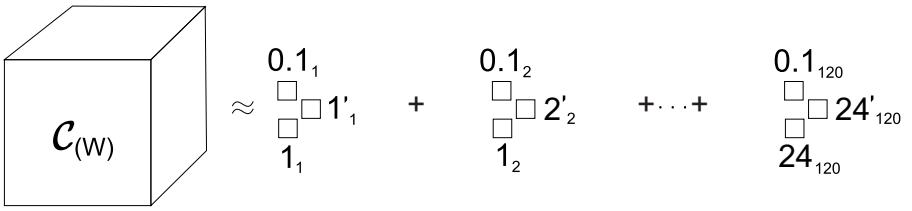
Finally, to demonstrate the idea of tensor components generated by Algorithm 3.2, we illustrate the tensors of action *North*, *South*, *West*, *East*, *Forward*, and *Backward* in Figure 3.10.



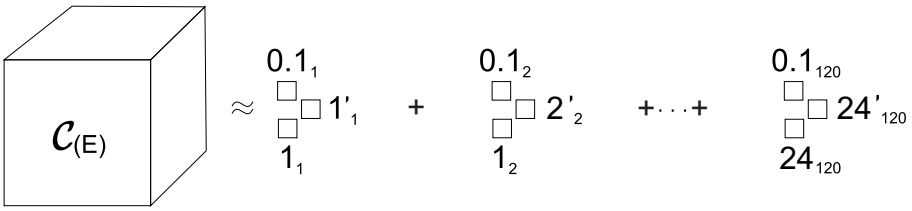
(a) North.



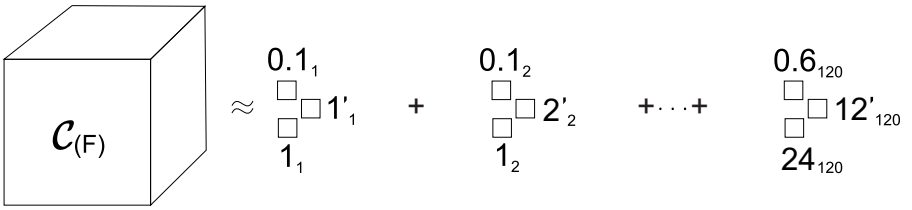
(b) South.



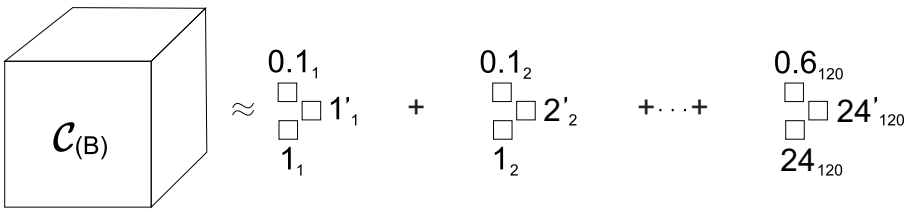
(c) West.



(d) East.



(e) Forward.



(f) Backward.

Figure 3.10: Illustration of tensor components of actions *North*, *South*, *West*, *East*, *Forward*, and *Backward*.

4. EXPERIMENTS

In this chapter, we report the experiments using the CP-MDP compact method. First, we address the problem we solve in Section 4.1. Then, in Section 4.2, we introduce the experimental setup and the tabular implementation we compare against the CP-MDP method. Finally, in Sections 4.3 and 4.4, we report the runtime and memory usage results of solving a GRIDWORLD problem using tabular approaches in comparison to CP-MDP-VI and CP-MDP-PI algorithms.

4.1 MDP Scenario

We evaluate our approach using standard GRIDWORLD single-agent MDP problems, which are classic navigation problems in artificial intelligence. The grid cells correspond to the states of the environment, and the grid size is defined by the number of states $\mathcal{S}$. We determine the number of actions that an agent can perform by $2 \times |\mathcal{D}|$, i.e., we consider two actions for each Cartesian plane. Obstacles $\mathcal{O}$ are states an agent has no access to. The interaction with the environment terminates when the agent reaches one of the terminal states $\mathcal{T}$. For example, for a two-dimensional grid, the environment is defined by $x \times y$ states, and the agent actions in a given state are *North* and *South* in the x-axis, and in the y-axis are *West* and *East*. For a three-dimensional grid, the environment is defined by $x \times y \times z$ states, and the actions are *Forward* and *Backward* in the z-axis, and so on.

4.2 Experimental Setup

We now detail the execution setup we use for the experiments. We evaluate the Python implementation of our approach[1] using an Intel(R) Xeon(R) CPU @ 2.30GHz with 13 GB RAM from a Python 3.6.9 Google Colaboratory notebook[2].We compare our results against a standard tabular implementation of value iteration and policy iteration algorithms in Python *pymdptoolbox*[3].

The plotted values consist of the average of 6 executions of the GRIDWORLD problem for each grid size configuration. We use a *discount factor* of γ = 0.9 and *maxIter* = 1,000. We define non-terminal states with a -3 reward. The grid contains randomly placed obstacles and terminal states, where half of the terminal states set receives additive +100 rewards, and the other half receives discounted -100 rewards. Table 4.1 shows the number

[1] Available at: https://github.com/danielakuinchtner/cp-mdp
[2] https://colab.research.google.com/
[3] https://pypi.org/project/pymdptoolbox/

of actions $\mathcal{A}$, states $\mathcal{S}$, obstacles $\mathcal{O}$, and terminals $\mathcal{T}$ we consider for each test environment (#). For example, for a 2D test with 4 actions, we use 6 terminals and 50 obstacles to solve a 4,900-state problem, where the total number of states are divided into two dimensions (e.g., $70 \times 70 = 4,900$). We describe the distribution of states into dimensions in Appendix A.

Table 4.1: Experimental setup for each execution: "#": number of each configuration; "$|\mathcal{S}|$": number of states; "$|\mathcal{T}|$": number of terminals; "$|\mathcal{O}|$": number of obstacles; "$|\mathcal{D}|$": number of dimensions; and "$|\mathcal{A}|$": number of actions.

#	**Number of States** $\|\mathcal{S}\|$					$\|\mathcal{T}\|$	$\|\mathcal{O}\|$
1)	4,900	4,000	3,125	2,048	3,888	6	50
2)	10,000	8,000	7,000	5,184	5,832	8	100
3)	14,400	12,500	10,000	9,216	8,748	10	200
4)	19,600	18,750	12,500	10,368	9,216	12	300
5)	22,500	24,000	19,200	18,432	17,496	14	400
6)	90,000	60,000	100,000	78,125	82,944	16	500
7)	250,000	125,000	200,000	233,280	196,008	18	600
8)	640,000	512,000	600,000	605,052	491,520	20	700
9)	1,000,000	1,000,000	1,200,000	823,543	800,000	22	800
$\|\mathcal{D}\|$	**2**	**3**	**5**	**7**	**9**		
$\|\mathcal{A}\|$	**4**	**6**	**10**	**14**	**18**		

4.3 Runtime Analysis

In this section, we show a comparison of runtime to solve the GRIDWORLD problem between the (i) TABULAR-VI and CP-MDP-VI, (ii) TABULAR-PI and CP-MDP-PI, and (iii) CP-MDP-VI and CP-MDP-PI algorithms.

Figure 4.1 shows the runtime in seconds to solve the GRIDWORLD problem using tabular value iteration (TABULAR-VI) and tabular policy iteration (TABULAR-PI) algorithms, and using the compact value iteration (CP-MDP-VI) and compact policy iteration (CP-MDP-PI) algorithms for problems with 2, 3, 5, 7 and 9 dimensions. While CP-MDP does not significantly outperform tabular methods for smaller problems (up to 5,000 states), it achieves a runtime improvement between 60% and 80% in most larger cases. For a two-dimensional 19,600-state problem, CP-MDP-VI achieves a runtime improvement of 82.45% compared to TABULAR-VI, as the number of dimensions increases, the runtime improvement between the two algorithms decreases in an average of 8.55%. And CP-MDP-PI achieves 68.99% runtime improvement compared to TABULAR-PI for 10,000-state and 14,400-state problems, as the number of dimensions increases, the improvement reduces to an average of 9.95%.

In all cases, CP-MDP-PI is substantially less efficient than CP-MDP-VI, taking an average of runtime 5 times longer than the compact value iteration, either TABULAR-PI compared to TABULAR-VI for a lower number of dimensions. And TABULAR-PI is unable to solve larger problems, e.g., tests with 18,750, 19,600, 12,500, 10,368, and 9,216, states compared to TABULAR-VI. To show results without a limit of visualization, in Appendix B (Figure B.1), we show the results of Figure 4.1 on a logarithmic scale.

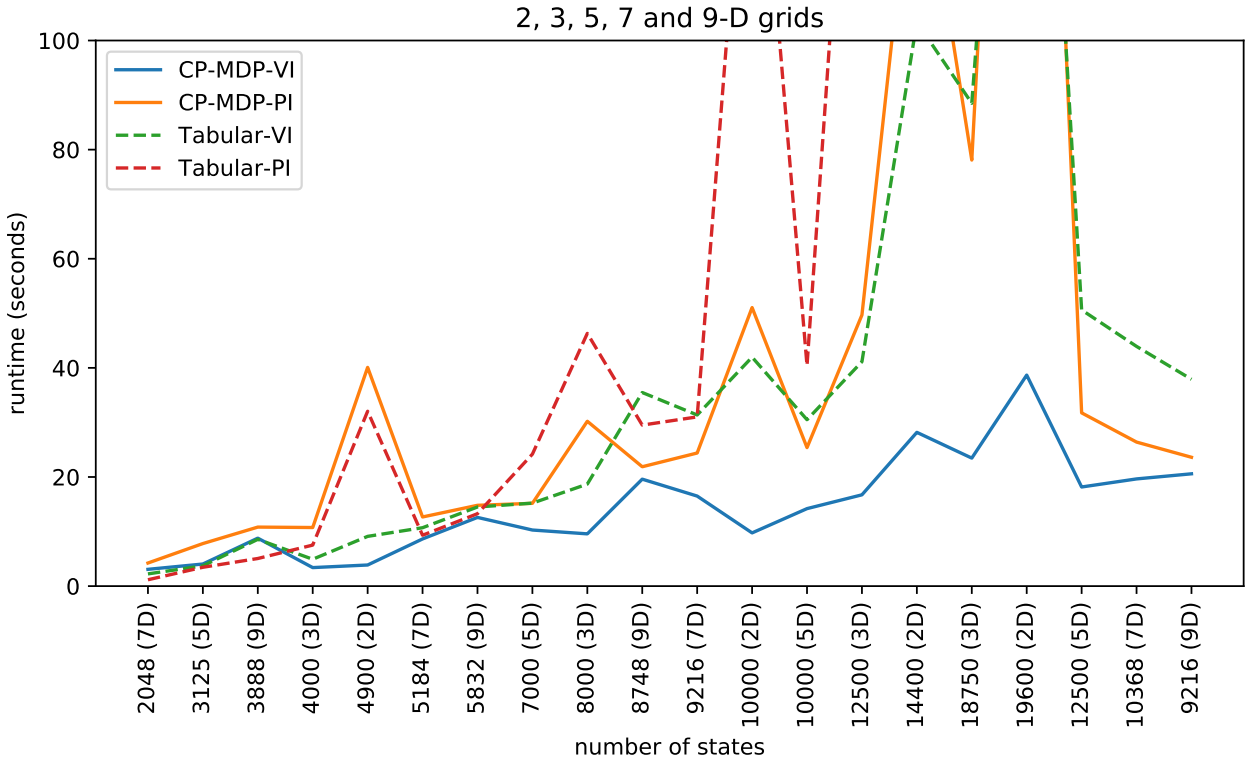


Figure 4.1: Runtime (in seconds) of CP-MDP-VI and CP-MDP-PI methods against the tabular value iteration (TABULAR-VI) and policy iteration (TABULAR-PI) algorithms of 2, 3, 5, 7, and 9 dimensions.

Figure 4.2 shows runtime results for tests performed only by the CP-MDP-VI and CP-MDP-PI algorithms, due to memory limitations to compute the solution for these grid sizes using tabular approaches. In some cases, problems with more than 250,000 states, CP-MDP-PI takes longer than 4 hours, so we interrupted the execution, and problems with more than 600,000 states, CP-MDP-PI is unable to run due to memory limitations, as well. To show results without a limit of visualization, in Appendix B (Figure B.2), we show the results of Figure 4.2 on a logarithmic scale.

In Figure 4.3, we show all runs performed by tabular and CP-MDP algorithms on a logarithmic scale. Comparing the four algorithms, CP-MDP-VI is the approach that outperforms the remaining algorithms since it can solve larger problems and in a more efficient way.

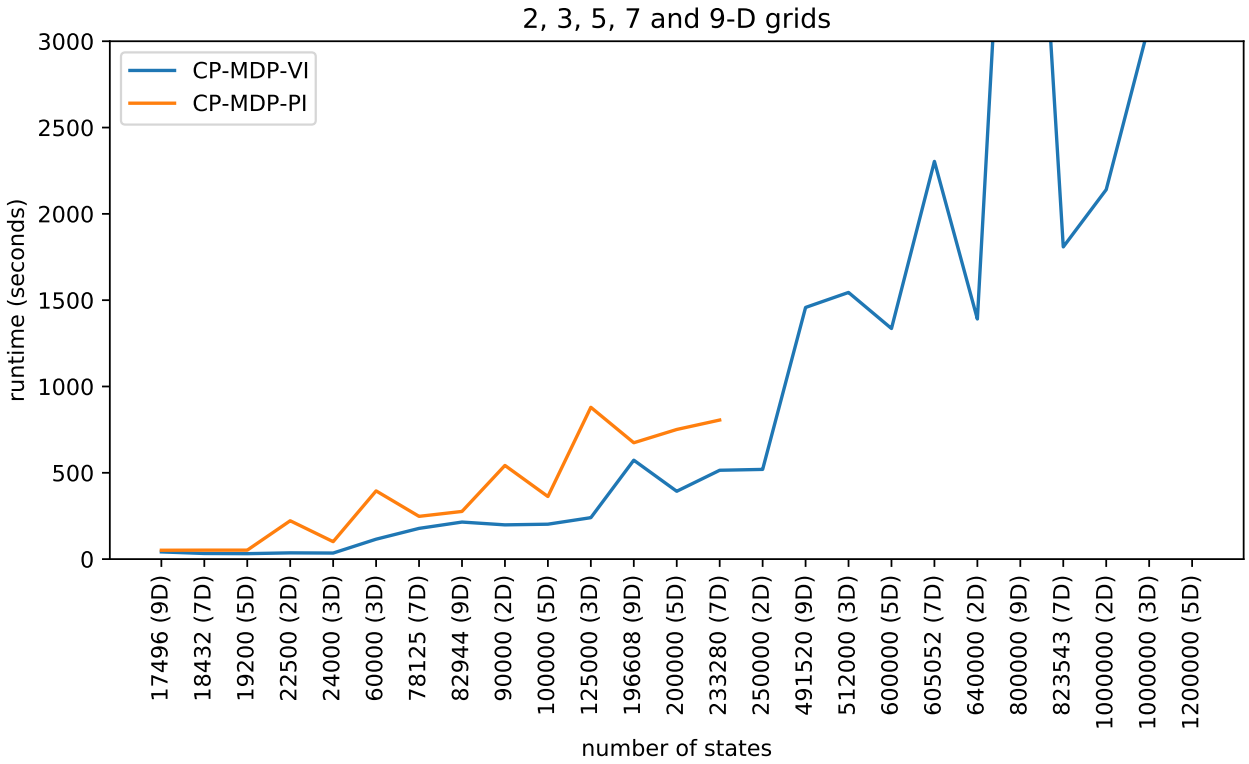


Figure 4.2: CP-MDP runtime to compute large grid sizes using the compact value and policy iteration.

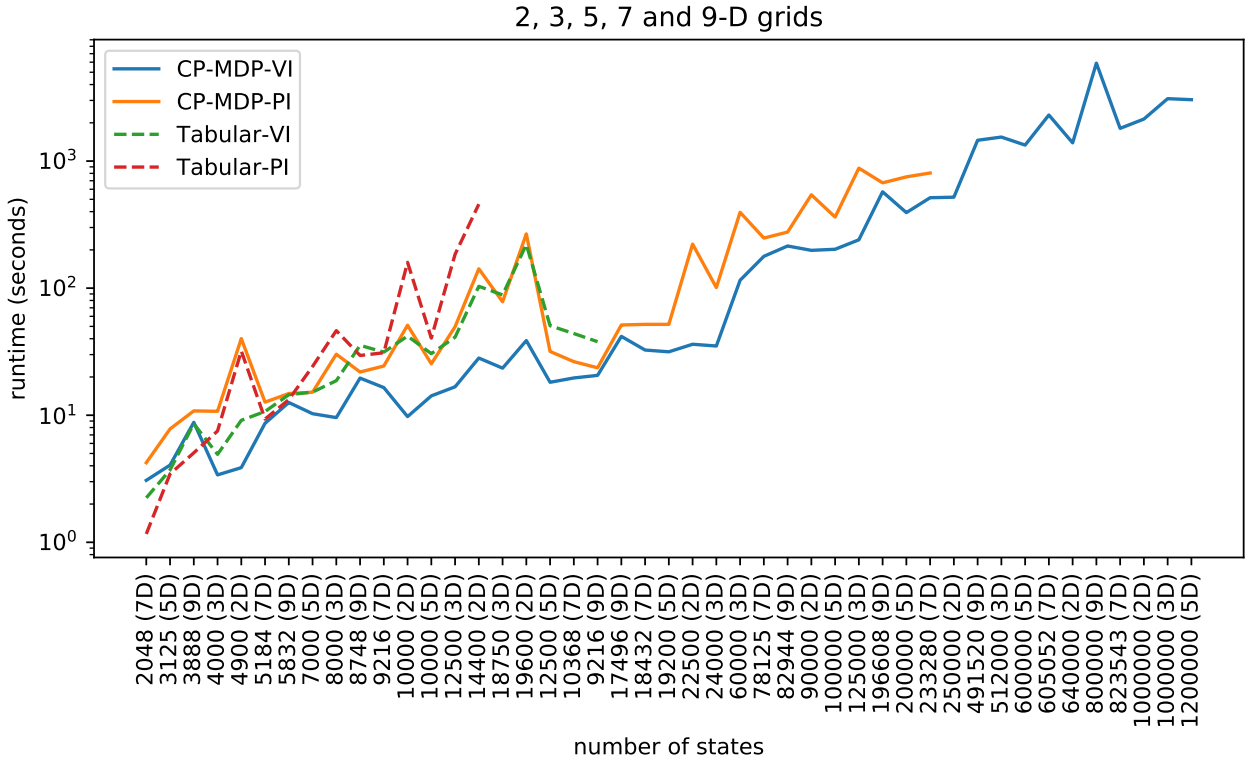


Figure 4.3: Runtime comparison between tabular and compact algorithms on a logarithmic scale.

Finally, in Table 4.2 we show a runtime improvement average between (i) CP-MDP-VI and TABULAR-VI, (ii) CP-MDP-PI and TABULAR-PI, and (iii) CP-MDP-VI and CP-MDP-PI. The table consists of an average of five sizes of grids, which we describe each size in Table 4.1 previously mentioned. For more details, we describe each test with its respective runtime improvement in Appendix C (Table C.1). About the first comparison, (i) CP-MDP-VI does not significantly exceed the tabular form runtime for runs of an average of 3,592 states. However, as the number of states grows, the average runtime improves from 7.94% up to 64.21%. Whereas, TABULAR-VI is not able to solve problems with an average of 20,326 up to 964,709 states, and TABULAR-PI from 14,087 up to 964,709 states. Consequently, we are not able to compare the improvement against the CP-MDP compact methods. In the second comparison, (ii) CP-MDP-PI is substantially slower than the TABULAR-PI for small problems. However, for larger problems with 7,203 up to 10,973 states, CP-MDP-PI starts to overcome the tabular form. Finally, (iii) the comparison between the two CP-MDP compact algorithms shows that CP-MDP-VI maintains a pattern of time improvement compared to CP-MDP-PI for small and larger problems.

# States	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI
1) 3,592	7.94%	-114.22%	50.59%
2) 7,203	38.07%	18.51%	45.65%
3) 10,973	55.53%	45.30%	46.66%
4) 14,087	64.21%	-	47.34%
5) 20,326	-	-	48.81%
6) 82,214	-	-	45.78%
7) 200,858	-	-	42.87%
8) 569,714	-	-	-
9) 964,709	-	-	-

Table 4.2: Average of runtime improvement comparison between (i) CP-MDP-VI and TABULAR-VI, (ii) CP-MDP-PI and TABULAR-PI, and (iii) CP-MDP-VI and CP-MDP-PI.

The standard deviation of time between runs is small; that is why we performed only 6 executions. For example, the standard deviation for tests between 4,900 to 18,750 states varies from 0.81 to 1.96. For larger problems (24,000 to 1,000,000 states), the dispersion ranges from 4.10 to 51.68, which is almost insignificant for larger tests that take more than 2,000 seconds to run.

4.4 Memory Analysis

In this section, we show a comparison of memory usage to solve the GridWorld problem between (i) Tabular-VI and Cp-Mdp-VI, (ii) Tabular-PI and Cp-Mdp-PI, and (iii) Cp-Mdp-VI and Cp-Mdp-PI. As we show in Figure 4.4, the memory requirements for the Cp-Mdp method are substantially lower than tabular ones, ranging from 131.54 to 246.31 MB. The Tabular-VI and Tabular-PI methods require much more memory than Cp-Mdp algorithms to the extent that we cannot run problems with more than 19,600 states using tabular forms. For a 19,600-state test, Tabular-VI uses 12.45 GB and Tabular-PI uses more memory than our set limit of 13 GB, whereas both Cp-Mdp-VI and Cp-Mdp-PI compute the solution using 151.33 MB and 149.66 MB, respectively. Consequently, the memory improvement of Cp-Mdp is more than 90% in all cases with more than 5,000 states than tabular value and policy iteration. To show results without a limit of visualization, in Appendix B (Figure B.3), we show the results of Figure 4.4 on a logarithmic scale.

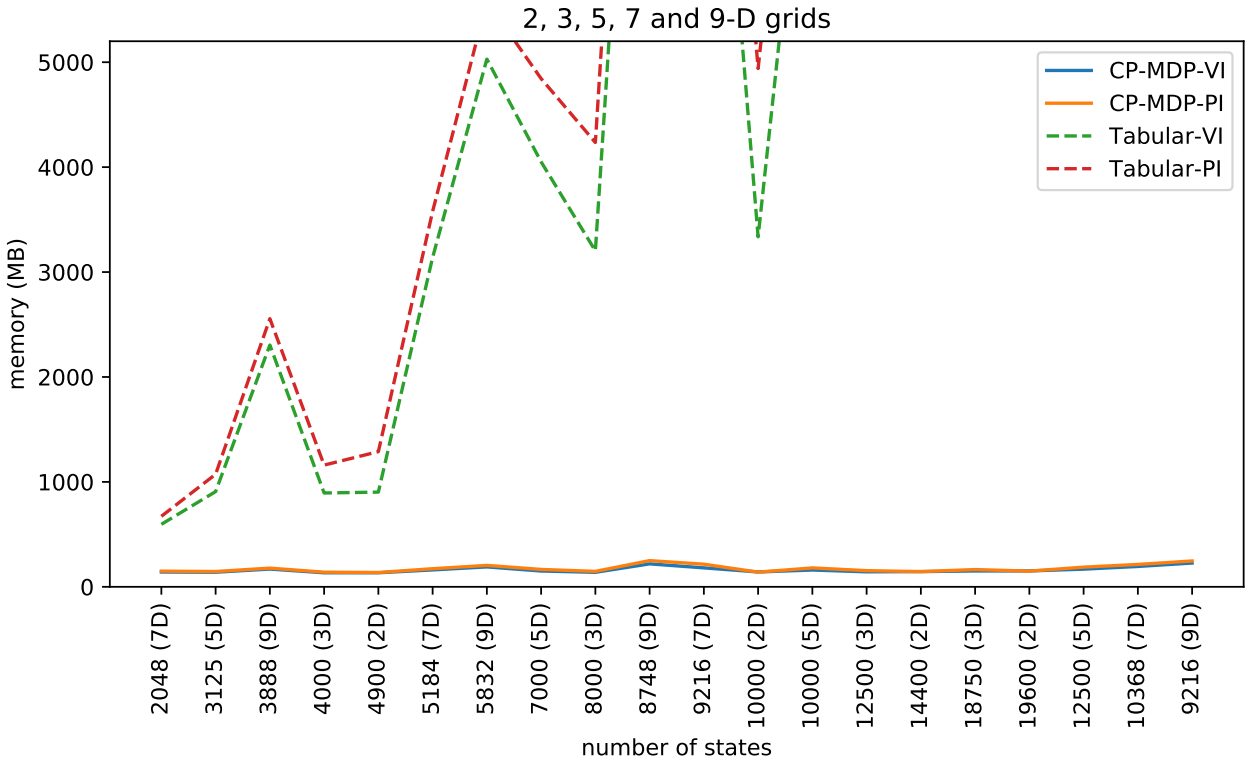


Figure 4.4: Memory (in MB) of Cp-Mdp-VI and Cp-Mdp-PI methods against the tabular value iteration (Tabular-VI) and policy iteration (Tabular-PI) algorithms of 2, 3, 5, 7, and 9 dimensions.

Due to memory limitations to compute using tabular algorithms, Figure 4.5 shows memory results for tests performed only by the Cp-Mdp methods. In some cases, Cp-Mdp-PI cannot compute due to memory limitations, and in some cases, the algorithm takes more than 4 hours to compute the solution, as we previously stated. To show results without a

limit of visualization, in Appendix B (Figure B.4), we show the results of Figure 4.5 on a logarithmic scale.

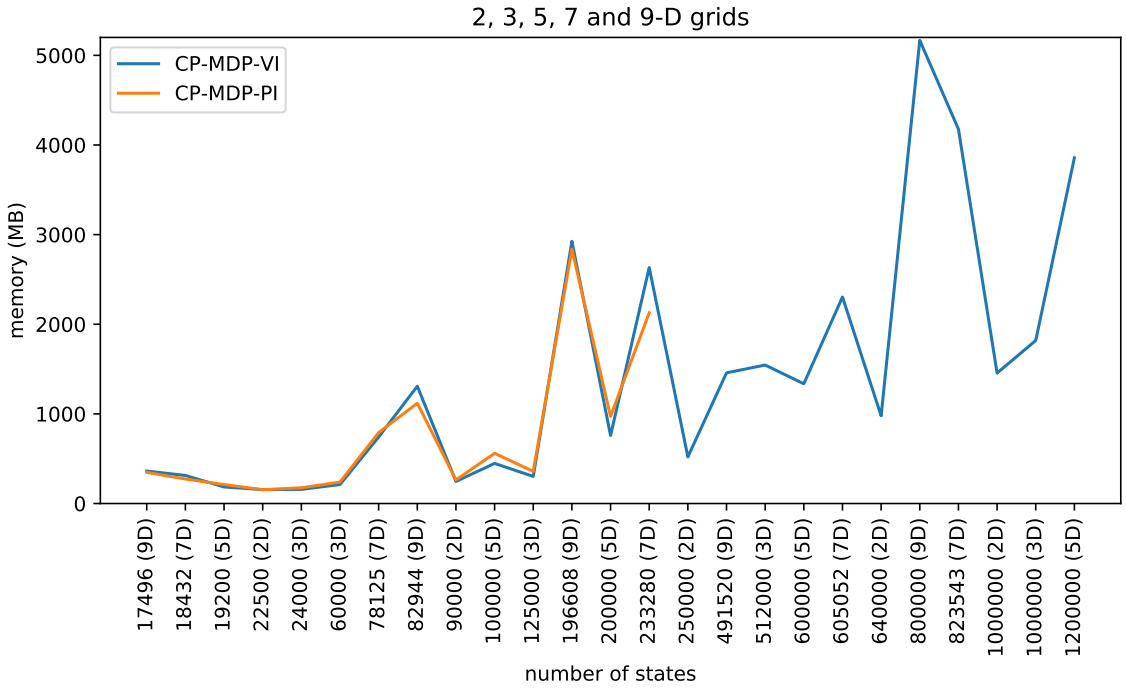


Figure 4.5: CP-MDP necessary memory to compute large grid sizes using the compact value and policy iteration.

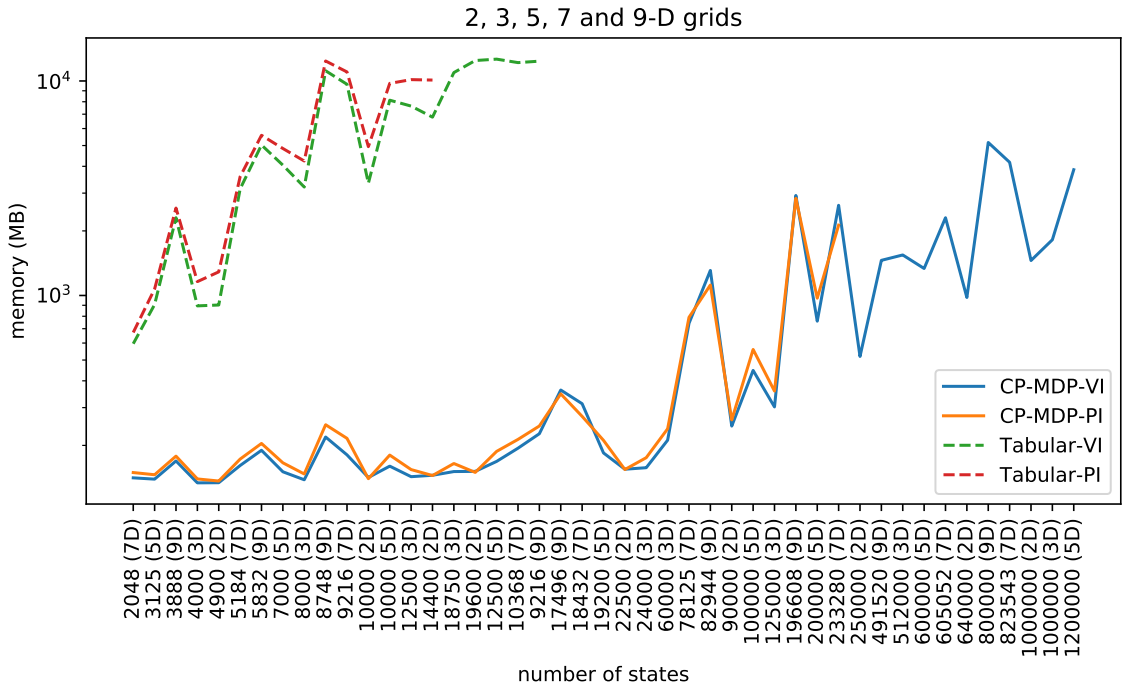


Figure 4.6: Memory comparison between tabular and compact algorithms on a logarithmic scale.

Now, on a logarithmic scale we show all runs performed by tabular algorithms and CP-MDP compact ones. The dispersion distance between tabular and compact methods is surprisingly large. However, the memory usage between CP-MDP-VI and CP-MDP-PI is quite similar. For example, CP-MDP-VI computes the solution of a 200,000-state 5-dimensional problem in 392.95 seconds using 758.86 MB, whereas, CP-MDP-PI takes 750.95 seconds using 971.28 MB. This pattern behavior is also valid for TABULAR-VI and TABULAR-PI. For example, for a 10,000-state 5-dimensional problem TABULAR-VI in 30.49 seconds uses 8,135.09 GB, whereas TABULAR-PI uses 9,744.13 GB in 40.42 seconds.

The number of dimensions affects memory usage, as well as runtime. For example, for a 1,000,000-state problem distributed into 2 dimensions takes 2140.34 seconds to compute using 1455.18 GB with CP-MDP-VI, whereas a 1,000,000-state problem distributed into 3 dimensions takes 3097.55 seconds to run using 1818.77 GB.

Finally, in Table 4.3 we show an average of memory usage improvement between (i) CP-MDP-VI and TABULAR-VI, (ii) CP-MDP-PI and TABULAR-PI, and (iii) CP-MDP-VI and CP-MDP-PI. We detail each test with its respective memory usage improvement in Appendix C. About the first comparison, (i) CP-MDP-VI significantly overcomes TABULAR-VI memory usage for all comparable runs. In the second comparison, (ii) CP-MDP-PI is also substantially faster than the TABULAR-PI for all problems. Finally, (iii) the comparison between the two CP-MDP compact algorithms shows the improvement of using CP-MDP-VI is merely bigger than CP-MDP-PI. In Appendix C (Table C.2), we show a detailed table containing all memory usage improvements for each grid size.

# States	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI
1) 3,592	84.76%	86.92%	4.21%
2) 7,203	95.76%	96.35%	5.54%
3) 10,973	98.04%	98.25%	9.37%
4) 14,087	98.53%	-	6.86%
5) 20,326	-	-	2.43%
6) 82,214	-	-	5.47%
7) 200,858	-	-	2.67%
8) 569,714	-	-	-
9) 964,709	-	-	-

Table 4.3: Average of memory usage improvement comparison between (i) CP-MDP-VI and TABULAR-VI, (ii) CP-MDP-PI and TABULAR-PI, and (iii) CP-MDP-VI and CP-MDP-PI.

5. RELATED WORK

This chapter addresses state-of-art papers for the factored MDP field: (i) Boutilier et al. (Boutilier et al., 2000), and (ii) Guestrin et al. (Guestrin et al., 2003). Then, we describe a work that uses a tensor decomposition method called KKT.

(i) The first work of Boutilier et al. (Boutilier et al., 2000) uses dynamic Bayesian networks to represent stochastic actions in an MDP, together with a decision-tree representation of rewards. Based on this representation, the authors develop versions of standard dynamic programming algorithms: Modified Policy Iteration (MPI) (Puterman and Shin, 1978) and Structured Policy Iteration (SPI) (Boutilier et al., 1995), which directly manipulate decision tree representations of policies and value functions. They perform tests using manufacturing problems with up to 1.8 million states. In the largest of these problems, MPI is unable to run to completion due to memory limitations, but SPI solves the problem in one-third of the time required by MPI.

(ii) The second work of Guestrin et al. (Guestrin et al., 2003) develops a DBN approach that presents two approximate solution algorithms, which exploit structure in factored MDPs. Both algorithms use an approximate value function represented as a linear combination of basis functions, where each basis function involves only a small subset of the domain variables. One algorithm uses approximate linear programming, and the second uses approximate dynamic programming (value and policy iteration). A central element of these algorithms is a novel linear program decomposition technique, which reduces an exponentially large problem to a provably equivalent, polynomial-sized one. The work provides experimental results on system administrator (SysAdmin) problems with over 10^{40} states, demonstrating a promising indication of the approach's scalability and exponential gains in computation time.

In general, these papers' methods and algorithms prove that decomposing the representation into smaller subsets (a factored representation) allows an exponential reduction in the representation size of structured MDPs. However, these works are based on factored state representations that use DBN-based approaches. In contrast, our research is focused on monolithic ones leveraging advances in tensor decomposition methods to increase MDPs solvers' efficiency. In our approach, instead of using dynamic Bayesian networks to represent state transition matrices, we provide a formalization of multidimensional monolithic MDPs using tensor algebra. In our formalization, we represent the state transition matrices by CANDECOMP-PARAFAC decomposition. Moreover, our approach focuses on grid problems, and the state-of-art of DBN approaches uses different kinds of problems, like *system administrator* problems and *manufacturing* problems.

To show a related work of MDPs addressing tensor decomposition, the research that comes closest to our proposal is Smart's dissertation (Smart, 2016). His work is based

on an unpublished formalization that is not available online. The author focuses on developing an MDP tensor decomposition algorithm and analyzes its computational performance and the optimality of its resultant solutions for obtaining optimal MDP policies. The author mentions a couple of tensor decomposition methods, such as CANDECOMP-PARAFAC (Carroll and Chang, 1970) and Tucker decomposition (Tucker, 1966). But, he concludes (erroneously) these methods are not viable candidates for MDP decomposition because they are not clear enough to transform an MDP problem to a CANDECOMP-PARAFAC or Tucker decomposition. Still, our work proves it otherwise for CANDECOMP-PARAFAC. So, he uses Karush-Kuhn-Tucker (KKT) (Kuhn and Tucker, 1951), a method to determine constraint qualifications for local optimality. However, the procedure does not appear to generalize well to MDPs without exact tensor decomposition components. In other words, the required optimization problem does not have a sufficient number of degrees of freedom to generate decomposition sub-problems, which reconstruct the state transition and reward matrices of the MDP. For tests, the author uses a small number of states, and the memory impact of the KKT method appears to be worse than the standard method.

Unlike Smart's work, we prove CANDECOMP-PARAFAC decomposition is a valid candidate to represent MDP transition matrices, and we show the CP-MDP method performs better than tabular algorithms for solver runtime and memory usage.

6. CONCLUSION

In this dissertation, we developed the CP-MDP, a tensor decomposition method that uses the CANDECOMP-PARAFAC decomposition to solve an MDP multidimensional problem using value iteration and policy iteration algorithms. To our knowledge, our work is the first one employing CANDECOMP-PARAFAC tensor decomposition to solve MDPs. Our empirical analysis shows CP-MDP-VI method solves more efficiently than CP-MDP-PI and tabular methods in both runtime and memory. First CP-MDP-VI achieves runtime improvements of up to 80% in the best cases than tabular approaches. Second, both CP-MDP-VI and CP-MDP-PI require substantially less memory to compute these solutions, decreasing memory usage by more than 90% for large multidimensional problems. The available memory is the bottleneck to carry out tests with many states for tabular approaches. Nevertheless, memory usage improvements for compact algorithms are the leading results in this research since we proved that by construction, the value function is identical to the tabular approaches. In other words, the policies of CP-MDP-VI and CP-MDP-PI are guaranteed to be optimal, much like the equivalent tabular methods.

Our main contributions are: (i) a formalization of MDPs multidimensional problems using tensor algebra, (ii) a novel implementation of a GRIDWORLD problem using the CANDECOMP-PARAFAC decomposition method, (iii) a compact value iteration (CP-MDP-VI) algorithm, (iv) a compact policy iteration (CP-MDP-PI) algorithm, (v) a runtime, memory and complexity analysis of both CP-MDP methods compared to tabular approaches, and as result of this research, (vi) we published a paper at the Mexican International Conference on Artificial Intelligence (MICAI) (Kuinchtner et al., 2020).

To leverage advances in GPU computation libraries, we tried several different ways to parallelize our approach to run on GPUs with Tensorflow[1], Pytorch[2] and Numba[3]. However, the runtime did not improve in any of these libraries. Such negative result seems to stem from the communication overhead between CPU and GPU, as our methods rely on multiplications between small tensor components.

As future work, we intend to use other tensor decomposition methods, such as Tensor-Train Decomposition (Oseledets, 2011) and Tucker Decomposition (Tucker, 1966) to perform the state transition matrices decomposition, aiming to improve runtime on GPUs. We also intend to generalize the CP-MDP implementation to solve several types of problems already described in RDDL (Sanner, 2010), aiming to compare our approach against the state-of-art of factored MDPs solvers for different problems.

[1] https://www.tensorflow.org/
[2] https://pytorch.org/
[3] https://numba.pydata.org/

REFERENCES

Amari, S. V. McLaughlin, L. and Hoang, P. (2006). Cost-effective condition-based maintenance using markov decision processes. In: *Proceesings of the Annual Reliability and Maintainability Symposium (RAMS '06)*, pp. 464–469, Greensburg, USA. IEEE.

Bellman, R. (1957a). *Dynamic Programming*. Dover Publications, Inc., New York, USA.

Bellman, R. (1960). *Introduction to Matrix Analysis*. McGraw-Hill, New York, USA.

Bellman, R. (Apr, 1954). Some applications of the theory of dynamic programming - a review. *Operational Research*, 2–3:275–288.

Bellman, R. (Apr, 1957b). A markovian decision process. *Journal of Mathematics and Mechanics*, 6–5:679–684.

Berhili, K. Koulali, M. and Berrehili, Y. (2020). Iiot-based prognostic health management using a Markov decision process approach. In: *Proceedings of the 5th Ubiquitous Networking (UNet'20)*, pp. 146–157, Cham, Switzerland. Springer International Publishing.

Boger, J. Hoey, J. Poupart, P. Boutilier, C. Fernie, G. and Mihailidis, A. (Apr, 2006). A planning system based on markov decision processes to guide people with dementia through activities of daily living. *IEEE Transactions on Information Technology in Biomedicine*, 10–2:323–333.

Boutilier, C. Dearden, R. and Goldszmidt, M. (1995). Exploiting structure in policy construction. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pp. 1104–1111, San Francisco, USA. Elsevier: Morgan Kaufmann Publishers Inc.

Boutilier, C. Dearden, R. and Goldszmidt, M. (Aug, 2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121–1:49—-107.

Carroll, J. and Chang, J. (Sep, 1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35:283–319.

Cassandra, A. R. Kaelbling, L. P. and Kurien, J. A. (1996). Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '96)*, pp. 963–972, Providence, USA. IEEE.

Davio, M. (Feb, 1981). Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, 30–2:116–125.

De Risi, V. (Feb, 2016). The development of euclidean axiomatics. *Archive for History of Exact Sciences*, 70–6:591—676.

Debreu, G. (1959). *Topological methods in cardinal utility theory*. Stanford University Press, New Haven, USA.

Delgado, K. V. Sanner, S. de Barros, L. N. and Cozman, F. G. (2009). Efficient solutions to factored mdps with imprecise transition probabilities. In: *Proceedings of the 19th International Conference on International Conference on Automated Planning and Scheduling (ICAPS'09)*, pp. 98–105, Thessaloniki, Greece. AAAI Press.

Durazo-Cardenas, I. Starr, A. Turner, C. J. Tiwari, A. Kirkwood, L. Bevilacqua, M. Tsourdos, A. Shehab, E. Baguley, P. Xu, Y. and Emmanouilidis, C. (Apr, 2018). An autonomous system for maintenance scheduling data-rich complex infrastructure: Fusing the railways' condition, planning and cost. *Transportation Research Part C: Emerging Technologies*, 89:234–253.

Guestrin, C. Koller, D. Parr, R. and Venkataraman, S. (Oct, 2003). Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research (JAIR)*, 19–1:399–468.

Guestrin, C. E. (2003). *Planning Under Uncertainty in Complex Structured Environments*. (ph.d. thesis), Stanford University, Stanford, USA.

Harshman, R. A. (Dec, 1970). Foundations of the parafac procedure: Models and conditions for an explanatory multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84.

Harshman, R. A. (Mar, 1972). Parafac2: Mathematical and technical notes. *UCLA Working Papers in Phonetics*, 22:30–44.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.

Iantovics, L. B. (Jan, 2008). Agent-based medical diagnosis systems. *Computing and Informatics*, 27:593–625.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press, St. Louis, USA.

Joly, M. Moro, L. F. L. and Pinto, J. M. (Jun, 2002). Planning and scheduling for petroleum refineries using mathematical programming. *Brazilian Journal of Chemical Engineering*, 19:207–228.

Jónsson, A. K. Morris, P. H. Muscettola, N. Rajan, K. and Smith, B. (2000). Planning in interplanetary space: Theory and practice. In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pp. 177—-186, Breckenridge, USA. AAAI Press.

Kiers, H. (May, 1998). A three-step algorithm for candecomp/parafac analysis of large data sets with multicollinearity. *Journal of Chemometrics*, 12:155–171.

Kolda, T. G. and Bader, B. W. (Aug, 2009). Tensor decompositions and applications. *SIAM Review*, 51–3:455–500.

Kuhn, H. W. and Tucker, A. W. (1951). Nonlinear programming. In: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492, Berkeley, USA. University of California Press.

Kuinchtner, D. Meneguzzi, F. and Sales, A. (2020). A tensor-based markov decision process representation. In: *Proceedings onf the 19th Mexican International Conference on Artificial Intelligence (MICAI'20): Advances in Soft Computing*, pp. 313–324, Porto Alegre, RS. Springer International Publishing.

Lamini, C. Benhlima, S. and Elbekri, A. (Mar, 2018). Genetic algorithm based approach for autonomous mobile robot path planning. *Procedia Computer Science*, 127:180–189.

Littman, M. L. Dean, T. L. and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. In: *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI'95)*, pp. 394–402, San Francisco, USA. Elsevier: Morgan Kaufmann Publishers Inc.

Liu, Q. Dong, M. and Chen, F. F. (Jun, 2018). Single-machine-based joint optimization of predictive maintenance planning and production scheduling. *Robotics and Computer-Integrated Manufacturing*, 51:238–247.

Oliehoek, F. A. and Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated.

Oseledets, I. (Jan, 2011). Tensor-train decomposition. *SIAM Journal Scientific Computing*, 33:2295–2317.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, USA.

Puterman, M. L. and Shin, M. C. (Jul, 1978). Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24–11:1127–1137.

Rabanser, S. Shchur, O. and Gunnemann, S. (Nov, 2017). Introduction to tensor decompositions and their applications in machine learning. *ArXiv*, abs/1711.10781.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, USA.

Sahe, K. E. Farah, I. R. and Ahmed, M. B. (2004). Multi-agent system for detecting and analysing changes on satellite image sequence. In: *Proceesings of the International Conference on Industrial Technology (ICIT'04)*, pp. 1579–1584, Manouba, Tunisie. IEEE.

Sanner, S. (2010). Relational dynamic influence diagram language (rddl): Language description. Retrieved from URL http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf. November 2020.

Schaefer, A. J. Bailey, M. D. Shechter, S. M. and Roberts, M. S. (2004). *Modeling Medical Treatment Using Markov Decision Processes*, chap. 23, pp. 593–612. Springer US, Boston, MA.

Sidiropoulos, N. D. De Lathauwer, L. Fu, X. Huang, K. Papalexakis, E. E. and Faloutsos, C. (Jul, 2017). Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65–13:3551–3582.

Smart, D. P. (2016). *Tensor decomposition and parallelization of Markov Decision Processes*. (ph.d. thesis), Massachusetts Institute of Technology, Cambridge, USA.

Sutton, R. S. and Barton, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, USA.

Tominac, P. and Mahalec, V. (Jul, 2017). A game theoretic framework for petroleum refinery strategic production planning. *AIChE Journal*, 63:2751–2763.

Toomey, C. and Mark, W. (Oct, 1995). Satellite image dissemination via software agents. *IEEE Expert*, 10–5:44–51.

Tucker, L. R. (Sep, 1966). Some mathematical notes on three-mode factor analysis. *Journal Psychometrika*, 31:279—-311.

APPENDIX A – GRID CONFIGURATION

In Table A.1, we show how many states we consider for each dimension. We use this configuration to run the tests for each grid size. For example, we assign a grid size of 3,125 states into 5 dimensions as: "5×5×5×5×5".

Number of Dimensions and States		
2	**3**	**5**
4,900 (70×70)	4,000 (10×20×20)	3,125 (5×5×5×5×5)
10,000 (100×100)	8,000 (20×20×20)	7,000 (5×5×5×7×8)
14,400 (120×120)	12,500 (20×25×25)	10,000 (5×5×5×8×10)
19,600 (140×140)	18,750 (25×25×30)	12,500 (5×5×5×10×10)
22,500 (150×150)	24,000 (20×30×40)	19,200 (5×6×8×8×10)
90,000 (300×300)	60,000 (30×40×50)	100,000 (10×10×10×10×10)
250,000 (500×500)	125,000 (50×50×50)	200,000 (5×10×10×20×20)
640,000 (800×800)	512,000 (80×80×80)	600,000 (10×10×15×20×20)
1,000,000 (1000×1000)	1,000,000 (100×100×100)	1,200,000 (10×15×20×20×20)

7	**9**
2,048 (2×2×2×4×4×4×4)	3,888 (2×2×2×2×3×3×3×3×3)
5,184 (3×3×3×3×4×4×4)	5,832 (2×2×2×3×3×3×3×3×3)
9,216 (3×3×4×4×4×4×4)	8,748 (2×2×3×3×3×3×3×3×3)
10,368 (3×3×3×4×4×4×6)	9,216 (2×2×2×2×3×3×4×4×4)
18,432 (3×4×4×4×4×4×6)	17,496 (2×3×3×3×3×3×3×3×4)
78,125 (5×5×5×5×5×5×5)	82,944 (3×3×3×3×4×4×4×4×4)
233,280 (5×6×6×6×6×6×6)	196,008 (3×4×4×4×4×4×4×4×4)
605,052 (6×6×7×7×7×7×7)	491,520 (4×4×4×4×4×4×4×5×6)
823,543 (7×7×7×7×7×7×7)	800,000 (4×4×4×4×5×5×5×5×5)

Table A.1: Number of states of each dimension.

APPENDIX B – LOGARITHMIC SCALE

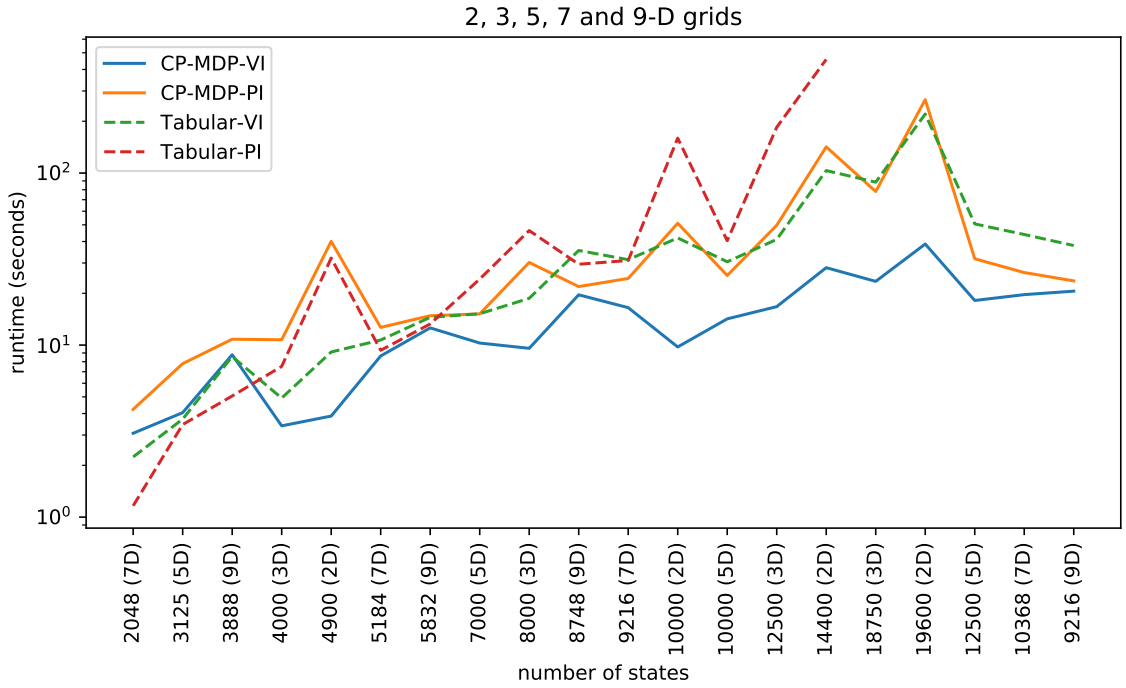


Figure B.1: Runtime of CP-MDP-VI and CP-MDP-PI methods against TABULAR-VI and TABULAR-PI algorithms of 2, 3, 5, 7, and 9 dimensions on a logarithmic scale.

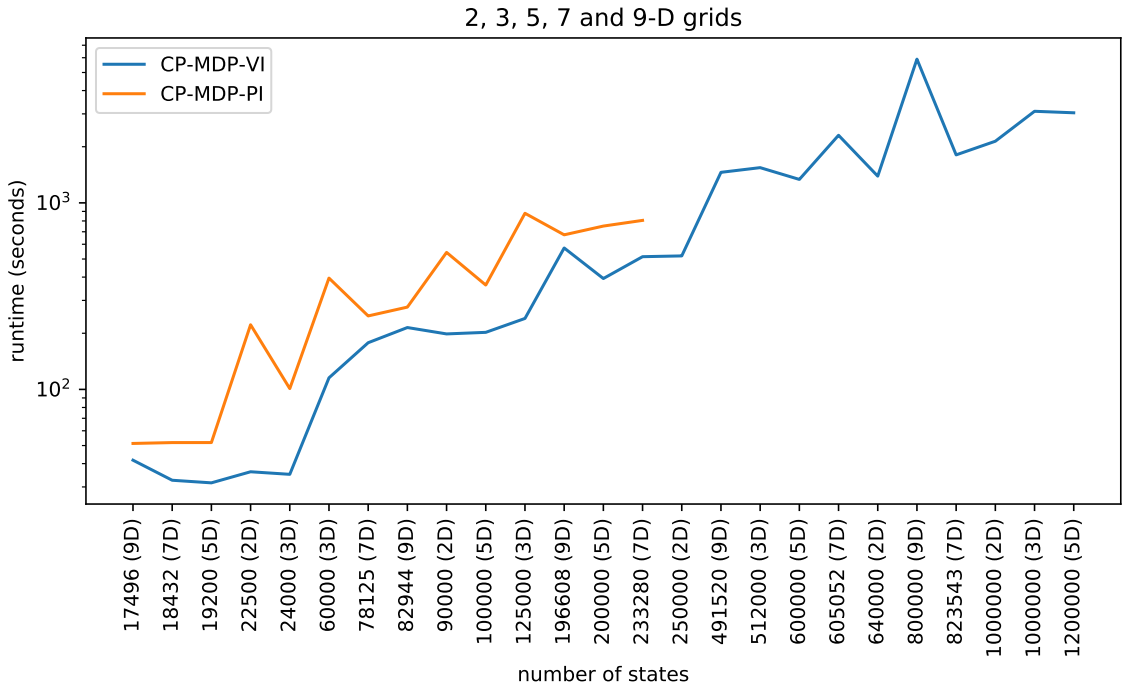


Figure B.2: CP-MDP-VI and CP-MDP-PI runtime to compute large grid sizes on a logarithmic scale.

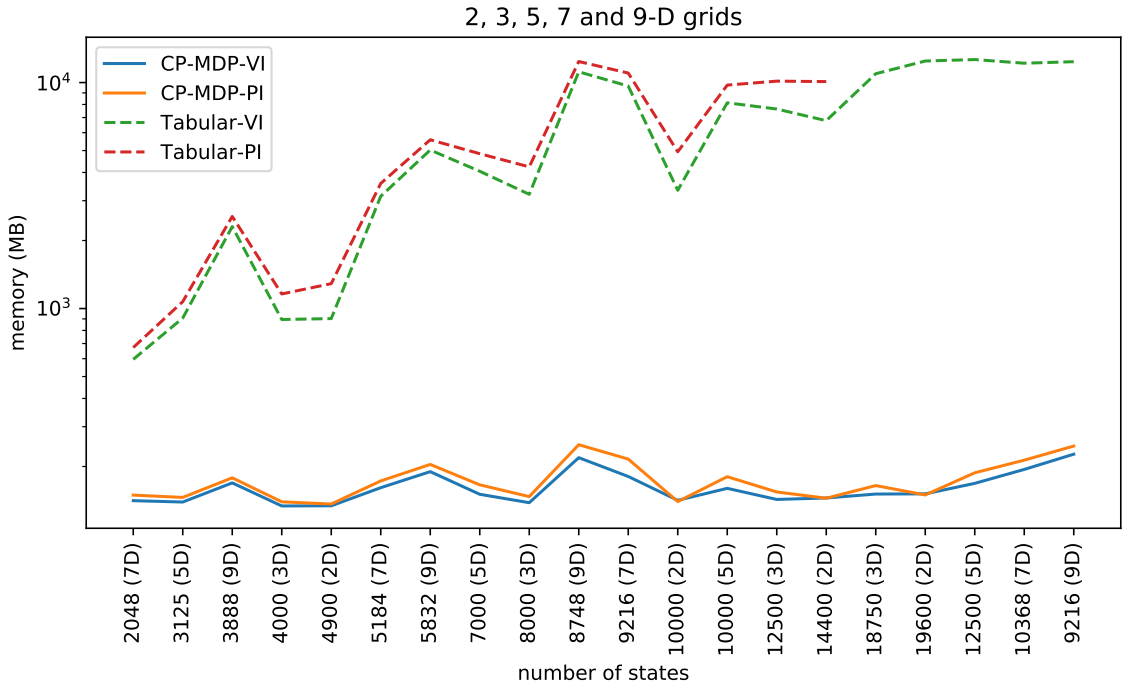


Figure B.3: Memory requirements of CP-MDP-VI and CP-MDP-PI methods against TABULAR-VI and TABULAR-PI algorithms of 2, 3, 5, 7, and 9 dimensions on a logarithmic scale.

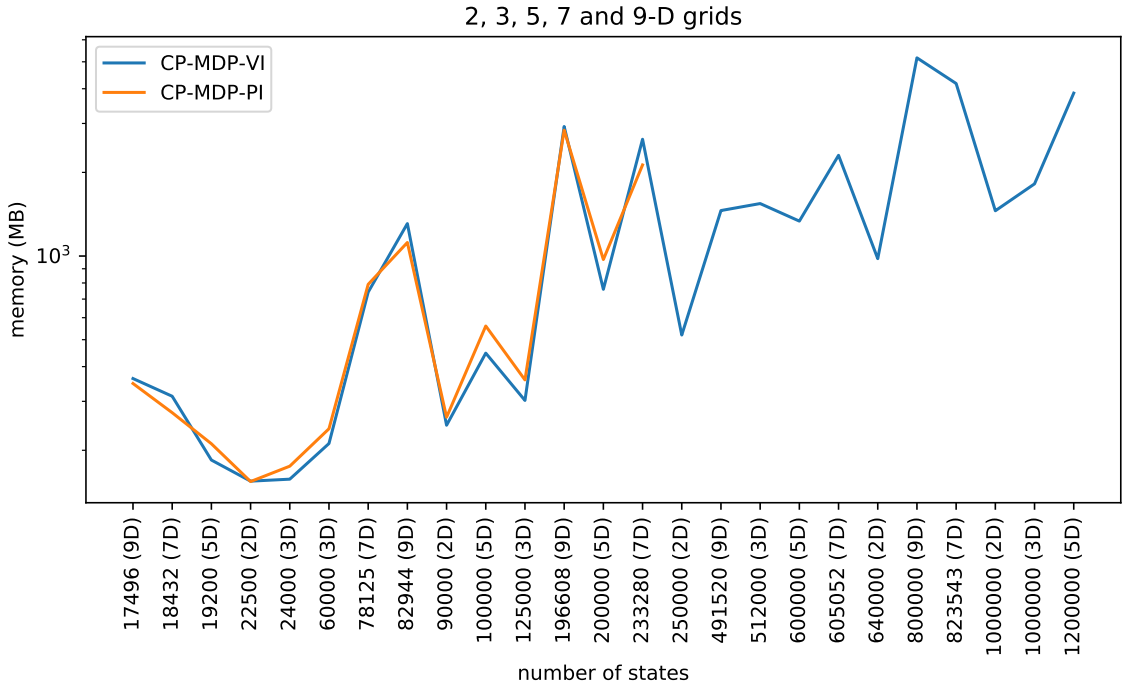


Figure B.4: CP-MDP-VI and CP-MDP-PI memory requirements to compute large grid sizes on a logarithmic scale.

APPENDIX C – RUNTIME AND MEMORY IMPROVEMENT

#	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI
1)		**4,900**			**4,000**			**3,125**			**2,048**			**3,888**	
	57.57%	-25.07%	90.35%	31.07%	-42.83%	68.41%	-9.08%	-125.88%	48.07%	-37.29%	-263.53%	27.37%	-2.56%	-113.81%	18.73%
2)		**10,000**			**8,000**			**7,000**			**5,184**			**5,832**	
	76.76%	68.01%	80.90%	48.78%	34.77%	68.31%	32.42%	37.27%	32.26%	19.22%	-35.88%	31.78%	13.18%	-11.60%	14.99%
3)		**14,400**			**12,500**			**10,000**			**9,216**			**8,748**	
	72.74%	68.99%	80.15%	59.32%	73.07%	66.33%	53.43%	37.21%	44.05%	47.39%	21.34%	32.35%	44.76%	25.87%	10.40%
4)		**19,600**			**18,750**			**12,500**			**10,368**			**9,216**	
	82.45%	-	85.53%	73.49%	-	69.95%	64.11%	-	42.79%	55.28%	-	25.58%	45.70%	-	12.83%
5)		**22,500**			**24,000**			**19,200**			**18,432**			**17,496**	
	-	-	83.71%	-	-	65.30%	-	-	39.19%	-	-	37.18%	-	-	18.66%
6)		**90,000**			**60,000**			**100,000**			**78,125**			**82,944**	
	-	-	63.45%	-	-	70.81%	-	-	44.22%	-	-	28.15%	-	-	22.29%
7)		**250,000**			**125,000**			**200,000**			**233,280**			**196,008**	
	-	-	-	-	-	72.70%	-	-	47.67%	-	-	36.12%	-	-	14.97%
8)		**640,000**			**512,000**			**600,000**			**605,052**			**491,520**	
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9)		**1,000,000**			**1,000,000**			**1,200,000**			**823,543**			**800,000**	
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$\lvert\mathcal{D}\rvert$		**2**			**3**			**5**			**7**			**9**	
$\lvert\mathcal{A}\rvert$		**4**			**6**			**10**			**14**			**18**	

Table C.1: Runtime improvement of each grid size.

#	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI	CP-MDP-VI x TABULAR-VI	CP-MDP-PI x TABULAR-PI	CP-MDP-VI x CP-MDP-PI
1)		**4,900**			**4,000**			**3,125**			**2,048**			**3,888**	
	85.16%	89.41%	1.79%	85.04%	87.97%	4.11%	84.66%	86.40%	4.61%	76.30%	77.77%	5.53%	92.66%	93.03%	5.03%
2)		**10,000**			**8,000**			**7,000**			**5,184**			**5,832**	
	95.75%	97.17%	-1.27%	95.68%	96.52%	6.03%	96.28%	96.57%	9.18%	94.87%	95.17%	6.67%	96.23%	96.34%	7.07%
3)		**14,400**			**12,500**			**10,000**			**9,216**			**8,748**	
	97.86%	98.57%	-0.27%	98.13%	98.48%	7.34%	98.03%	98.15%	11.19%	98.13%	98.04%	16.24%	98.04%	97.99%	12.35%
4)		**19,600**			**18,750**			**12,500**			**10,368**			**9,216**	
	98.78%	-	-1.12%	98.62%	-	8.27%	98.67%	-	10.19%	98.41%	-	9.00%	98.17%	-	7.96%
5)		**22,500**			**24,000**			**19,200**			**18,432**			**17,496**	
	-	-	-0.32%	-	-	10.21%	-	-	12.63%	-	-	-14.57%	-	-	-4.19%
6)		**90,000**			**60,000**			**100,000**			**78,125**			**82,944**	
	-	-	6.42%	-	-	11.50%	-	-	20.13%	-	-	6.24%	-	-	-16.95%
7)		**250,000**			**125,000**			**200,000**			**233,280**			**196,008**	
	-	-	-	-	-	15.65%	-	-	21.87%	-	-	-23.74%	-	-	-3.09%
8)		**640,000**			**512,000**			**600,000**			**605,052**			**491,520**	
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9)		**1,000,000**			**1,000,000**			**1,200,000**			**823,543**			**800,000**	
	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$\lvert\mathcal{D}\rvert$		**2**			**3**			**5**			**7**			**9**	
$\lvert\mathcal{A}\rvert$		**4**			**6**			**10**			**14**			**18**	

Table C.2: Memory improvement of each grid size.